```php
<?php
/**
  * Yii bootstrap file.
  *
  * This file is automatically generated using 'build lite' command.
  * It is the result of merging commonly used Yii class files with
  * comments and trace statements removed away.
  *
  * By using this file instead of yii.php, an Yii application may
  * improve performance due to the reduction of PHP parsing time.
  * The performance improvement is especially obvious when PHP APC extension
  * is enabled.
  *
  * DO NOT modify this file manually.
  *
  * @author Qiang Xue <qiang.xue@gmail.com>
  * @link http://www.yiiframework.com/
  * @copyright Copyright &copy; 2008-2012 Yii Software LLC
  * @license http://www.yiiframework.com/license/
  * @version $Id: $
  * @since 1.0
  */


defined('YII_BEGIN_TIME') or define('YII_BEGIN_TIME',microtime(true));
defined('YII_DEBUG') or define('YII_DEBUG',false);
defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',0);
defined('YII_ENABLE_EXCEPTION_HANDLER') or define('YII_ENABLE_EXCEPTION_HANDLER',true);
defined('YII_ENABLE_ERROR_HANDLER') or define('YII_ENABLE_ERROR_HANDLER',true);
defined('YII_PATH') or define('YII_PATH',dirname(__FILE__));
defined('YII_ZII_PATH') or define('YII_ZII_PATH',YII_PATH.DIRECTORY_SEPARATOR.'zii');
class YiiBase
{
	public static $classMap=array();
	public static $enableIncludePath=true;
	private static $_aliases=array('system'=>YII_PATH,'zii'=>YII_ZII_PATH); // alias => path
	private static $_imports=array();					// alias => class name or directory
	private static $_includePaths;						// list of include paths
	private static $_app;
	private static $_logger;
	public static function getVersion()
	{
		return '1.1.13';
	}
	public static function createWebApplication($config=null)
	{
		return self::createApplication('CWebApplication',$config);
	}
	public static function createConsoleApplication($config=null)
	{
		return self::createApplication('CConsoleApplication',$config);
	}
	public static function createApplication($class,$config=null)
	{
		return new $class($config);
	}
	public static function app()
	{
		return self::$_app;
	}
	public static function setApplication($app)
	{
		if(self::$_app===null || $app===null)
			self::$_app=$app;
		else
			throw new CException(Yii::t('yii','Yii application can only be created once.'));
	}
	public static function getFrameworkPath()
	{
		return YII_PATH;
	}
	public static function createComponent($config)
	{
		if(is_string($config))
		{
			$type=$config;
			$config=array();
		}
		elseif(isset($config['class']))
		{
			$type=$config['class'];
			unset($config['class']);
		}
		else
			throw new CException(Yii::t('yii','Object configuration must be an array containing
a "class" element.'));
		if(!class_exists($type,false))
			$type=Yii::import($type,true);
		if(($n=func_num_args())>1)
		{
			$args=func_get_args();
			if($n===2)
				$object=new $type($args[1]);
			elseif($n===3)
				$object=new $type($args[1],$args[2]);
			elseif($n===4)
				$object=new $type($args[1],$args[2],$args[3]);
			else
			{
				unset($args[0]);
				$class=new ReflectionClass($type);
				// Note: ReflectionClass::newInstanceArgs() is available for PHP 5.1.3+
				// $object=$class->newInstanceArgs($args);
				$object=call_user_func_array(array($class,'newInstance'),$args);
			}
		}
		else
			$object=new $type;
		foreach($config as $key=>$value)
			$object->$key=$value;
		return $object;
	}
	public static function import($alias,$forceInclude=false)
	{
		if(isset(self::$_imports[$alias]))	// previously imported
			return self::$_imports[$alias];
		if(class_exists($alias,false) || interface_exists($alias,false))
			return self::$_imports[$alias]=$alias;
		if(($pos=strrpos($alias,'\\'))!==false) // a class name in PHP 5.3 namespace format
		{
			$namespace=str_replace('\\','.',ltrim(substr($alias,0,$pos),'\\'));
			if(($path=self::getPathOfAlias($namespace))!==false)
			{
				$classFile=$path.DIRECTORY_SEPARATOR.substr($alias,$pos+1).'.php';
				if($forceInclude)
				{
					if(is_file($classFile))
						require($classFile);
					else
						throw new CException(Yii::t('yii','Alias "{alias}" is invalid. Make sure
it points to an existing PHP file and the file is readable.',array('{alias}'=>$alias)));
					self::$_imports[$alias]=$alias;
```

```
                    }
                    else
                        self::$classMap[$alias]=$classFile;
                    return $alias;
                }
                else
                    throw new CException(Yii::t('yii','Alias "{alias}" is invalid. Make sure it points
to an existing directory.',
                        array('{alias}'=>$namespace)));
            }
            if(($pos=strrpos($alias,'.'))===false)    // a simple class name
            {
                if($forceInclude && self::autoload($alias))
                    self::$_imports[$alias]=$alias;
                return $alias;
            }
            $className=(string)substr($alias,$pos+1);
            $isClass=$className!=='*';
            if($isClass && (class_exists($className,false) || interface_exists($className,false)))
                return self::$_imports[$alias]=$className;
            if(($path=self::getPathOfAlias($alias))!==false)
            {
                if($isClass)
                {
                    if($forceInclude)
                    {
                        if(is_file($path.'.php'))
                            require($path.'.php');
                        else
                            throw new CException(Yii::t('yii','Alias "{alias}" is invalid. Make sure
it points to an existing PHP file and the file is readable.',array('{alias}'=>$alias)));
                        self::$_imports[$alias]=$className;
                    }
                    else
                        self::$classMap[$className]=$path.'.php';
                    return $className;
                }
                else    // a directory
                {
                    if(self::$_includePaths===null)
                    {

    self::$_includePaths=array_unique(explode(PATH_SEPARATOR,get_include_path()));
                        if(($pos=array_search('.',self::$_includePaths,true))!==false)
                        unset(self::$_includePaths[$pos]);
                    }
                    array_unshift(self::$_includePaths,$path);
                    if(self::$enableIncludePath                                                &&
set_include_path('.'.PATH_SEPARATOR.implode(PATH_SEPARATOR,self::$_includePaths))===false)
                        self::$enableIncludePath=false;
                    return self::$_imports[$alias]=$path;
                }
            }
            else
                throw new CException(Yii::t('yii','Alias "{alias}" is invalid. Make sure it points to an
existing directory or file.',
                    array('{alias}'=>$alias)));
        }
        public static function getPathOfAlias($alias)
        {
            if(isset(self::$_aliases[$alias]))
                return self::$_aliases[$alias];
            elseif(($pos=strpos($alias,'.'))!==false)
            {
                $rootAlias=substr($alias,0,$pos);
                if(isset(self::$_aliases[$rootAlias]))
                    return
self::$_aliases[$alias]=rtrim(self::$_aliases[$rootAlias].DIRECTORY_SEPARATOR.str_replace('.',DIR
ECTORY_SEPARATOR,substr($alias,$pos+1)),'*'.DIRECTORY_SEPARATOR);
                elseif(self::$_app instanceof CWebApplication)
                {
                    if(self::$_app->findModule($rootAlias)!==null)
                        return self::getPathOfAlias($alias);
                }
            }
            return false;
        }
        public static function setPathOfAlias($alias,$path)
        {
            if(empty($path))
                unset(self::$_aliases[$alias]);
            else
                self::$_aliases[$alias]=rtrim($path,'\\/');
        }
        public static function autoload($className)
        {
            // use include so that the error PHP file may appear
            if(isset(self::$classMap[$className]))
                include(self::$classMap[$className]);
            elseif(isset(self::$_coreClasses[$className]))
                include(YII_PATH.self::$_coreClasses[$className]);
            else
            {
                // include class file relying on include_path
                if(strpos($className,'\\')===false)    // class without namespace
                {
                    if(self::$enableIncludePath===false)
                    {
                        foreach(self::$_includePaths as $path)
                        {
                            $classFile=$path.DIRECTORY_SEPARATOR.$className.'.php';
                            if(is_file($classFile))
                            {
                                include($classFile);
                                if(YII_DEBUG                                    &&
basename(realpath($classFile))!==$className.'.php')
                                throw   new   CException(Yii::t('yii','Class   name   "{class}"
does not match class file "{file}".', array(
                                    '{class}'=>$className,
                                    '{file}'=>$classFile,
                                )));
                                break;
                            }
                        }
                    }
                    else
                        include($className.'.php');
                }
                else    // class name with namespace in PHP 5.3
                {
                    $namespace=str_replace('\\','.',ltrim($className,'\\'));
                    if(($path=self::getPathOfAlias($namespace))!==false)
                        include($path.'.php');
                    else
                        return false;
                }
                return class_exists($className,false) || interface_exists($className,false);
            }
            return true;
        }
        public static function trace($msg,$category='application')
        {
        if(YII_DEBUG)
                self::log($msg,CLogger::LEVEL_TRACE,$category);
}
public static function log($msg,$level=CLogger::LEVEL_INFO,$category='application')
{
        if(self::$_logger===null)
                self::$_logger=new CLogger;
        if(YII_DEBUG && YII_TRACE_LEVEL>0 && $level!==CLogger::LEVEL_PROFILE)
        {
                $traces=debug_backtrace();
                $count=0;
                foreach($traces as $trace)
                {
                        if(isset($trace['file'],$trace['line']) && strpos($trace['file'],YII_PATH)!==0)
                        {
                                $msg.="\nin ".$trace['file'].' ('.$trace['line'].')';
                                if(++$count>=YII_TRACE_LEVEL)
                                        break;
                        }
                }
        }
        self::$_logger->log($msg,$level,$category);
}
public static function beginProfile($token,$category='application')
{
        self::log('begin:'.$token,CLogger::LEVEL_PROFILE,$category);
}
public static function endProfile($token,$category='application')
{
        self::log('end:'.$token,CLogger::LEVEL_PROFILE,$category);
}
public static function getLogger()
{
        if(self::$_logger!==null)
                return self::$_logger;
        else
                return self::$_logger=new CLogger;
}
public static function setLogger($logger)
{
        self::$_logger=$logger;
}
public static function powered()
{
        return Yii::t('yii','Powered by {yii}.', array('{yii}'=>'<a
href="http://www.yiiframework.com/" rel="external">Yii Framework</a>'));
    }
    public static function t($category,$message,$params=array(),$source=null,$language=null)
    {
        if(self::$_app!==null)
        {
            if($source===null)
                $source=($category==='yii'||$category==='zii')?'coreMessages':'messages';
            if(($source=self::$_app->getComponent($source))!==null)
                $message=$source->translate($category,$message,$language);
        }
        if($params===array())
            return $message;
        if(!is_array($params))
            $params=array($params);
        if(isset($params[0])) // number choice
        {
            if(strpos($message,'|')!==false)
            {
                if(strpos($message,'#')===false)
                {
                    $chunks=explode('|',$message);
                    $expressions=self::$_app->getLocale($language)->getPluralRules();
                    if($n=min(count($chunks),count($expressions)))
                    {
                        for($i=0;$i<$n;$i++)
                            $chunks[$i]=$expressions[$i].'#'.$chunks[$i];
                        $message=implode('|',$chunks);
                    }
                }
                $message=CChoiceFormat::format($message,$params[0]);
            }
            if(!isset($params['{n}']))
                $params['{n}']=$params[0];
            unset($params[0]);
        }
        return $params!==array() ? strtr($message,$params) : $message;
    }
    public static function registerAutoloader($callback, $append=false)
    {
        if($append)
        {
            self::$enableIncludePath=false;
				spl_autoload_register($callback);
			}
			else
			{
				spl_autoload_unregister(array('YiiBase','autoload'));
				spl_autoload_register($callback);
				spl_autoload_register(array('YiiBase','autoload'));
			}
		}
		private static $_coreClasses=array(
			'CApplication' => '/base/CApplication.php',
			'CApplicationComponent' => '/base/CApplicationComponent.php',
			'CBehavior' => '/base/CBehavior.php',
			'CComponent' => '/base/CComponent.php',
			'CErrorEvent' => '/base/CErrorEvent.php',
			'CErrorHandler' => '/base/CErrorHandler.php',
			'CException' => '/base/CException.php',
			'CExceptionEvent' => '/base/CExceptionEvent.php',
			'CHttpException' => '/base/CHttpException.php',
			'CModel' => '/base/CModel.php',
			'CModelBehavior' => '/base/CModelBehavior.php',
			'CModelEvent' => '/base/CModelEvent.php',
			'CModule' => '/base/CModule.php',
			'CSecurityManager' => '/base/CSecurityManager.php',
			'CStatePersister' => '/base/CStatePersister.php',
			'CApcCache' => '/caching/CApcCache.php',
			'CCache' => '/caching/CCache.php',
			'CDbCache' => '/caching/CDbCache.php',
			'CDummyCache' => '/caching/CDummyCache.php',
			'CEAcceleratorCache' => '/caching/CEAcceleratorCache.php',
			'CFileCache' => '/caching/CFileCache.php',
			'CMemCache' => '/caching/CMemCache.php',
			'CWinCache' => '/caching/CWinCache.php',
			'CXCache' => '/caching/CXCache.php',
			'CZendDataCache' => '/caching/CZendDataCache.php',
			'CCacheDependency' => '/caching/dependencies/CCacheDependency.php',
			'CChainedCacheDependency'                                                      =>
'/caching/dependencies/CChainedCacheDependency.php',
			'CDbCacheDependency' => '/caching/dependencies/CDbCacheDependency.php',
			'CDirectoryCacheDependency'                                                    =>
'/caching/dependencies/CDirectoryCacheDependency.php',
			'CExpressionDependency' => '/caching/dependencies/CExpressionDependency.php',
			'CFileCacheDependency' => '/caching/dependencies/CFileCacheDependency.php',
			'CGlobalStateCacheDependency'                                                  =>
```

```
'/caching/dependencies/CGlobalStateCacheDependency.php',
		'CAttributeCollection' => '/collections/CAttributeCollection.php',
		'CConfiguration' => '/collections/CConfiguration.php',
		'CList' => '/collections/CList.php',
		'CListIterator' => '/collections/CListIterator.php',
		'CMap' => '/collections/CMap.php',
		'CMapIterator' => '/collections/CMapIterator.php',
		'CQueue' => '/collections/CQueue.php',
		'CQueueIterator' => '/collections/CQueueIterator.php',
		'CStack' => '/collections/CStack.php',
		'CStackIterator' => '/collections/CStackIterator.php',
		'CTypedList' => '/collections/CTypedList.php',
		'CTypedMap' => '/collections/CTypedMap.php',
		'CConsoleApplication' => '/console/CConsoleApplication.php',
		'CConsoleCommand' => '/console/CConsoleCommand.php',
		'CConsoleCommandBehavior' => '/console/CConsoleCommandBehavior.php',
		'CConsoleCommandEvent' => '/console/CConsoleCommandEvent.php',
		'CConsoleCommandRunner' => '/console/CConsoleCommandRunner.php',
		'CHelpCommand' => '/console/CHelpCommand.php',
		'CDbCommand' => '/db/CDbCommand.php',
		'CDbConnection' => '/db/CDbConnection.php',
		'CDbDataReader' => '/db/CDbDataReader.php',
		'CDbException' => '/db/CDbException.php',
		'CDbMigration' => '/db/CDbMigration.php',
		'CDbTransaction' => '/db/CDbTransaction.php',
		'CActiveFinder' => '/db/ar/CActiveFinder.php',
		'CActiveRecord' => '/db/ar/CActiveRecord.php',
		'CActiveRecordBehavior' => '/db/ar/CActiveRecordBehavior.php',
		'CDbColumnSchema' => '/db/schema/CDbColumnSchema.php',
		'CDbCommandBuilder' => '/db/schema/CDbCommandBuilder.php',
		'CDbCriteria' => '/db/schema/CDbCriteria.php',
		'CDbExpression' => '/db/schema/CDbExpression.php',
		'CDbSchema' => '/db/schema/CDbSchema.php',
		'CDbTableSchema' => '/db/schema/CDbTableSchema.php',
		'CMssqlColumnSchema' => '/db/schema/mssql/CMssqlColumnSchema.php',
		'CMssqlCommandBuilder' => '/db/schema/mssql/CMssqlCommandBuilder.php',
		'CMssqlPdoAdapter' => '/db/schema/mssql/CMssqlPdoAdapter.php',
		'CMssqlSchema' => '/db/schema/mssql/CMssqlSchema.php',
		'CMssqlSqlsrvPdoAdapter' => '/db/schema/mssql/CMssqlSqlsrvPdoAdapter.php',
		'CMssqlTableSchema' => '/db/schema/mssql/CMssqlTableSchema.php',
		'CMysqlColumnSchema' => '/db/schema/mysql/CMysqlColumnSchema.php',
		'CMysqlCommandBuilder' => '/db/schema/mysql/CMysqlCommandBuilder.php',
		'CMysqlSchema' => '/db/schema/mysql/CMysqlSchema.php',
		'CMysqlTableSchema' => '/db/schema/mysql/CMysqlTableSchema.php',
'COciColumnSchema' => '/db/schema/oci/COciColumnSchema.php',
'COciCommandBuilder' => '/db/schema/oci/COciCommandBuilder.php',
'COciSchema' => '/db/schema/oci/COciSchema.php',
'COciTableSchema' => '/db/schema/oci/COciTableSchema.php',
'CPgsqlColumnSchema' => '/db/schema/pgsql/CPgsqlColumnSchema.php',
'CPgsqlSchema' => '/db/schema/pgsql/CPgsqlSchema.php',
'CPgsqlTableSchema' => '/db/schema/pgsql/CPgsqlTableSchema.php',
'CSqliteColumnSchema' => '/db/schema/sqlite/CSqliteColumnSchema.php',
'CSqliteCommandBuilder' => '/db/schema/sqlite/CSqliteCommandBuilder.php',
'CSqliteSchema' => '/db/schema/sqlite/CSqliteSchema.php',
'CChoiceFormat' => '/i18n/CChoiceFormat.php',
'CDateFormatter' => '/i18n/CDateFormatter.php',
'CDbMessageSource' => '/i18n/CDbMessageSource.php',
'CGettextMessageSource' => '/i18n/CGettextMessageSource.php',
'CLocale' => '/i18n/CLocale.php',
'CMessageSource' => '/i18n/CMessageSource.php',
'CNumberFormatter' => '/i18n/CNumberFormatter.php',
'CPhpMessageSource' => '/i18n/CPhpMessageSource.php',
'CGettextFile' => '/i18n/gettext/CGettextFile.php',
'CGettextMoFile' => '/i18n/gettext/CGettextMoFile.php',
'CGettextPoFile' => '/i18n/gettext/CGettextPoFile.php',
'CChainedLogFilter' => '/logging/CChainedLogFilter.php',
'CDbLogRoute' => '/logging/CDbLogRoute.php',
'CEmailLogRoute' => '/logging/CEmailLogRoute.php',
'CFileLogRoute' => '/logging/CFileLogRoute.php',
'CLogFilter' => '/logging/CLogFilter.php',
'CLogRoute' => '/logging/CLogRoute.php',
'CLogRouter' => '/logging/CLogRouter.php',
'CLogger' => '/logging/CLogger.php',
'CProfileLogRoute' => '/logging/CProfileLogRoute.php',
'CWebLogRoute' => '/logging/CWebLogRoute.php',
'CDateTimeParser' => '/utils/CDateTimeParser.php',
'CFileHelper' => '/utils/CFileHelper.php',
'CFormatter' => '/utils/CFormatter.php',
'CMarkdownParser' => '/utils/CMarkdownParser.php',
'CPropertyValue' => '/utils/CPropertyValue.php',
'CTimestamp' => '/utils/CTimestamp.php',
'CVarDumper' => '/utils/CVarDumper.php',
'CBooleanValidator' => '/validators/CBooleanValidator.php',
'CCaptchaValidator' => '/validators/CCaptchaValidator.php',
'CCompareValidator' => '/validators/CCompareValidator.php',
'CDateValidator' => '/validators/CDateValidator.php',
'CDefaultValueValidator' => '/validators/CDefaultValueValidator.php',
'CEmailValidator' => '/validators/CEmailValidator.php',
'CExistValidator' => '/validators/CExistValidator.php',
'CFileValidator' => '/validators/CFileValidator.php',
'CFilterValidator' => '/validators/CFilterValidator.php',
'CInlineValidator' => '/validators/CInlineValidator.php',
'CNumberValidator' => '/validators/CNumberValidator.php',
'CRangeValidator' => '/validators/CRangeValidator.php',
'CRegularExpressionValidator' => '/validators/CRegularExpressionValidator.php',
'CRequiredValidator' => '/validators/CRequiredValidator.php',
'CSafeValidator' => '/validators/CSafeValidator.php',
'CStringValidator' => '/validators/CStringValidator.php',
'CTypeValidator' => '/validators/CTypeValidator.php',
'CUniqueValidator' => '/validators/CUniqueValidator.php',
'CUnsafeValidator' => '/validators/CUnsafeValidator.php',
'CUrlValidator' => '/validators/CUrlValidator.php',
'CValidator' => '/validators/CValidator.php',
'CActiveDataProvider' => '/web/CActiveDataProvider.php',
'CArrayDataProvider' => '/web/CArrayDataProvider.php',
'CAssetManager' => '/web/CAssetManager.php',
'CBaseController' => '/web/CBaseController.php',
'CCacheHttpSession' => '/web/CCacheHttpSession.php',
'CClientScript' => '/web/CClientScript.php',
'CController' => '/web/CController.php',
'CDataProvider' => '/web/CDataProvider.php',
'CDataProviderIterator' => '/web/CDataProviderIterator.php',
'CDbHttpSession' => '/web/CDbHttpSession.php',
'CExtController' => '/web/CExtController.php',
'CFormModel' => '/web/CFormModel.php',
'CHttpCookie' => '/web/CHttpCookie.php',
'CHttpRequest' => '/web/CHttpRequest.php',
'CHttpSession' => '/web/CHttpSession.php',
'CHttpSessionIterator' => '/web/CHttpSessionIterator.php',
'COutputEvent' => '/web/COutputEvent.php',
'CPagination' => '/web/CPagination.php',
'CSort' => '/web/CSort.php',
'CSqlDataProvider' => '/web/CSqlDataProvider.php',
'CTheme' => '/web/CTheme.php',
'CThemeManager' => '/web/CThemeManager.php',
'CUploadedFile' => '/web/CUploadedFile.php',
'CUrlManager' => '/web/CUrlManager.php',
'CWebApplication' => '/web/CWebApplication.php',
'CWebModule' => '/web/CWebModule.php',
'CWidgetFactory' => '/web/CWidgetFactory.php',
'CAction' => '/web/actions/CAction.php',
'CInlineAction' => '/web/actions/CInlineAction.php',
'CViewAction' => '/web/actions/CViewAction.php',
'CAccessControlFilter' => '/web/auth/CAccessControlFilter.php',
'CAuthAssignment' => '/web/auth/CAuthAssignment.php',
'CAuthItem' => '/web/auth/CAuthItem.php',
'CAuthManager' => '/web/auth/CAuthManager.php',
'CBaseUserIdentity' => '/web/auth/CBaseUserIdentity.php',
'CDbAuthManager' => '/web/auth/CDbAuthManager.php',
'CPhpAuthManager' => '/web/auth/CPhpAuthManager.php',
'CUserIdentity' => '/web/auth/CUserIdentity.php',
'CWebUser' => '/web/auth/CWebUser.php',
'CFilter' => '/web/filters/CFilter.php',
'CFilterChain' => '/web/filters/CFilterChain.php',
'CHttpCacheFilter' => '/web/filters/CHttpCacheFilter.php',
'CInlineFilter' => '/web/filters/CInlineFilter.php',
'CForm' => '/web/form/CForm.php',
'CFormButtonElement' => '/web/form/CFormButtonElement.php',
'CFormElement' => '/web/form/CFormElement.php',
'CFormElementCollection' => '/web/form/CFormElementCollection.php',
'CFormInputElement' => '/web/form/CFormInputElement.php',
'CFormStringElement' => '/web/form/CFormStringElement.php',
'CGoogleApi' => '/web/helpers/CGoogleApi.php',
'CHtml' => '/web/helpers/CHtml.php',
'CJSON' => '/web/helpers/CJSON.php',
'CJavaScript' => '/web/helpers/CJavaScript.php',
'CJavaScriptExpression' => '/web/helpers/CJavaScriptExpression.php',
'CPradoViewRenderer' => '/web/renderers/CPradoViewRenderer.php',
'CViewRenderer' => '/web/renderers/CViewRenderer.php',
'CWebService' => '/web/services/CWebService.php',
'CWebServiceAction' => '/web/services/CWebServiceAction.php',
'CWsdlGenerator' => '/web/services/CWsdlGenerator.php',
'CActiveForm' => '/web/widgets/CActiveForm.php',
'CAutoComplete' => '/web/widgets/CAutoComplete.php',
'CClipWidget' => '/web/widgets/CClipWidget.php',
'CContentDecorator' => '/web/widgets/CContentDecorator.php',
'CFilterWidget' => '/web/widgets/CFilterWidget.php',
'CFlexWidget' => '/web/widgets/CFlexWidget.php',
'CHtmlPurifier' => '/web/widgets/CHtmlPurifier.php',
'CInputWidget' => '/web/widgets/CInputWidget.php',
'CMarkdown' => '/web/widgets/CMarkdown.php',
'CMaskedTextField' => '/web/widgets/CMaskedTextField.php',
'CMultiFileUpload' => '/web/widgets/CMultiFileUpload.php',
'COutputCache' => '/web/widgets/COutputCache.php',
'COutputProcessor' => '/web/widgets/COutputProcessor.php',
'CStarRating' => '/web/widgets/CStarRating.php',
		'CTabView' => '/web/widgets/CTabView.php',
		'CTextHighlighter' => '/web/widgets/CTextHighlighter.php',
		'CTreeView' => '/web/widgets/CTreeView.php',
		'CWidget' => '/web/widgets/CWidget.php',
		'CCaptcha' => '/web/widgets/captcha/CCaptcha.php',
		'CCaptchaAction' => '/web/widgets/captcha/CCaptchaAction.php',
		'CBasePager' => '/web/widgets/pagers/CBasePager.php',
		'CLinkPager' => '/web/widgets/pagers/CLinkPager.php',
		'CListPager' => '/web/widgets/pagers/CListPager.php',
	);
}
spl_autoload_register(array('YiiBase','autoload'));
class Yii extends YiiBase
{
}
class CComponent
{
	private $_e;
	private $_m;
	public function __get($name)
	{
		$getter='get'.$name;
		if(method_exists($this,$getter))
			return $this->$getter();
		elseif(strncasecmp($name,'on',2)===0 && method_exists($this,$name))
		{
			// duplicating getEventHandlers() here for performance
			$name=strtolower($name);
			if(!isset($this->_e[$name]))
				$this->_e[$name]=new CList;
			return $this->_e[$name];
		}
		elseif(isset($this->_m[$name]))
			return $this->_m[$name];
		elseif(is_array($this->_m))
		{
			foreach($this->_m as $object)
			{
				if($object->getEnabled()	&&	(property_exists($object,$name)	||
$object->canGetProperty($name)))
					return $object->$name;
			}
		}
		throw new CException(Yii::t('yii','Property "{class}.{property}" is not defined.',
                array('{class}'=>get_class($this), '{property}'=>$name)));
    }
    public function __set($name,$value)
    {
        $setter='set'.$name;
        if(method_exists($this,$setter))
            return $this->$setter($value);
        elseif(strncasecmp($name,'on',2)===0 && method_exists($this,$name))
        {
            // duplicating getEventHandlers() here for performance
            $name=strtolower($name);
            if(!isset($this->_e[$name]))
                $this->_e[$name]=new CList;
            return $this->_e[$name]->add($value);
        }
        elseif(is_array($this->_m))
        {
            foreach($this->_m as $object)
            {
                if($object->getEnabled()        &&        (property_exists($object,$name)        ||
$object->canSetProperty($name)))
                    return $object->$name=$value;
            }
        }
        if(method_exists($this,'get'.$name))
            throw new CException(Yii::t('yii','Property "{class}.{property}" is read only.',
                array('{class}'=>get_class($this), '{property}'=>$name)));
        else
            throw new CException(Yii::t('yii','Property "{class}.{property}" is not defined.',
                array('{class}'=>get_class($this), '{property}'=>$name)));
    }
    public function __isset($name)
    {
        $getter='get'.$name;
        if(method_exists($this,$getter))
            return $this->$getter()!==null;
        elseif(strncasecmp($name,'on',2)===0 && method_exists($this,$name))
        {
            $name=strtolower($name);
            return isset($this->_e[$name]) && $this->_e[$name]->getCount();
        }
        elseif(is_array($this->_m))
        {
            if(isset($this->_m[$name]))
```

```
                    return true;
                foreach($this->_m as $object)
                {
                    if($object->getEnabled()        &&        (property_exists($object,$name)        ||
$object->canGetProperty($name)))
                        return $object->$name!==null;
                }
            }
            return false;
        }
        public function __unset($name)
        {
            $setter='set'.$name;
            if(method_exists($this,$setter))
                $this->$setter(null);
            elseif(strncasecmp($name,'on',2)===0 && method_exists($this,$name))
                unset($this->_e[strtolower($name)]);
            elseif(is_array($this->_m))
            {
                if(isset($this->_m[$name]))
                    $this->detachBehavior($name);
                else
                {
                    foreach($this->_m as $object)
                    {
                        if($object->getEnabled())
                        {
                            if(property_exists($object,$name))
                                return $object->$name=null;
                            elseif($object->canSetProperty($name))
                                return $object->$setter(null);
                        }
                    }
                }
            }
            elseif(method_exists($this,'get'.$name))
                throw new CException(Yii::t('yii','Property "{class}.{property}" is read only.',
                    array('{class}'=>get_class($this), '{property}'=>$name)));
        }
        public function __call($name,$parameters)
        {
            if($this->_m!==null)
            {
                foreach($this->_m as $object)
                {
                        if($object->getEnabled() && method_exists($object,$name))
                                return call_user_func_array(array($object,$name),$parameters);
                }
        }
        if(class_exists('Closure',  false)  &&  $this->canGetProperty($name)  &&  $this->$name
instanceof Closure)
                return call_user_func_array($this->$name, $parameters);
        throw new CException(Yii::t('yii','{class} and its behaviors do not have a method or
closure named "{name}".',
                array('{class}'=>get_class($this), '{name}'=>$name)));
    }
    public function asa($behavior)
    {
        return isset($this->_m[$behavior]) ? $this->_m[$behavior] : null;
    }
    public function attachBehaviors($behaviors)
    {
        foreach($behaviors as $name=>$behavior)
                $this->attachBehavior($name,$behavior);
    }
    public function detachBehaviors()
    {
        if($this->_m!==null)
        {
                foreach($this->_m as $name=>$behavior)
                        $this->detachBehavior($name);
                $this->_m=null;
        }
    }
    public function attachBehavior($name,$behavior)
    {
        if(!($behavior instanceof IBehavior))
                $behavior=Yii::createComponent($behavior);
        $behavior->setEnabled(true);
        $behavior->attach($this);
        return $this->_m[$name]=$behavior;
    }
    public function detachBehavior($name)
    {
        if(isset($this->_m[$name]))
        {
                $this->_m[$name]->detach($this);
                $behavior=$this->_m[$name];
                unset($this->_m[$name]);
                return $behavior;
        }
}
public function enableBehaviors()
{
        if($this->_m!==null)
        {
                foreach($this->_m as $behavior)
                        $behavior->setEnabled(true);
        }
}
public function disableBehaviors()
{
        if($this->_m!==null)
        {
                foreach($this->_m as $behavior)
                        $behavior->setEnabled(false);
        }
}
public function enableBehavior($name)
{
        if(isset($this->_m[$name]))
                $this->_m[$name]->setEnabled(true);
}
public function disableBehavior($name)
{
        if(isset($this->_m[$name]))
                $this->_m[$name]->setEnabled(false);
}
public function hasProperty($name)
{
        return method_exists($this,'get'.$name) || method_exists($this,'set'.$name);
}
public function canGetProperty($name)
{
        return method_exists($this,'get'.$name);
}
public function canSetProperty($name)
{
        return method_exists($this,'set'.$name);
}
public function hasEvent($name)
{
        return !strncasecmp($name,'on',2) && method_exists($this,$name);
}
public function hasEventHandler($name)
{
        $name=strtolower($name);
        return isset($this->_e[$name]) && $this->_e[$name]->getCount()>0;
}
public function getEventHandlers($name)
{
        if($this->hasEvent($name))
        {
                $name=strtolower($name);
                if(!isset($this->_e[$name]))
                        $this->_e[$name]=new CList;
                return $this->_e[$name];
        }
        else
                throw new CException(Yii::t('yii','Event "{class}.{event}" is not defined.',
                        array('{class}'=>get_class($this), '{event}'=>$name)));
}
public function attachEventHandler($name,$handler)
{
        $this->getEventHandlers($name)->add($handler);
}
public function detachEventHandler($name,$handler)
{
        if($this->hasEventHandler($name))
                return $this->getEventHandlers($name)->remove($handler)!==false;
        else
                return false;
}
public function raiseEvent($name,$event)
{
        $name=strtolower($name);
        if(isset($this->_e[$name]))
        {
                foreach($this->_e[$name] as $handler)
                {
                        if(is_string($handler))
                                call_user_func($handler,$event);
                        elseif(is_callable($handler,true))
                        {
                                if(is_array($handler))
                                {
                                        // an array: 0 - object, 1 - method name
                                        list($object,$method)=$handler;
                                        if(is_string($object))      // static method call
                                            call_user_func($handler,$event);
                                        elseif(method_exists($object,$method))
                                            $object->$method($event);
                                        else
                                            throw   new   CException(Yii::t('yii','Event   "{class}.{event}"   is
attached with an invalid handler "{handler}".',
                                                array('{class}'=>get_class($this),           '{event}'=>$name,
'{handler}'=>$handler[1])));
                                    }
                                    else // PHP 5.3: anonymous function
                                        call_user_func($handler,$event);
                                }
                                else
                                    throw new CException(Yii::t('yii','Event "{class}.{event}" is attached with
an invalid handler "{handler}".',
                                        array('{class}'=>get_class($this),                        '{event}'=>$name,
'{handler}'=>gettype($handler))));
                                // stop further handling if param.handled is set true
                                if(($event instanceof CEvent) && $event->handled)
                                    return;
                            }
                        }
                        elseif(YII_DEBUG && !$this->hasEvent($name))
                            throw new CException(Yii::t('yii','Event "{class}.{event}" is not defined.',
                                array('{class}'=>get_class($this), '{event}'=>$name)));
    }
    public function evaluateExpression($_expression_,$_data_=array())
    {
        if(is_string($_expression_))
        {
            extract($_data_);
            return eval('return '.$_expression_.';');
        }
        else
        {
            $_data_[]=$this;
            return call_user_func_array($_expression_, $_data_);
        }
    }
}
class CEvent extends CComponent
```

```
{
        public $sender;
        public $handled=false;
        public $params;
        public function __construct($sender=null,$params=null)
        {
                $this->sender=$sender;
                $this->params=$params;
        }
}
class CEnumerable
{
}
abstract class CModule extends CComponent
{
        public $preload=array();
        public $behaviors=array();
        private $_id;
        private $_parentModule;
        private $_basePath;
        private $_modulePath;
        private $_params;
        private $_modules=array();
        private $_moduleConfig=array();
        private $_components=array();
        private $_componentConfig=array();
        public function __construct($id,$parent,$config=null)
        {
                $this->_id=$id;
                $this->_parentModule=$parent;
                // set basePath at early as possible to avoid trouble
                if(is_string($config))
                        $config=require($config);
                if(isset($config['basePath']))
                {
                        $this->setBasePath($config['basePath']);
                        unset($config['basePath']);
                }
                Yii::setPathOfAlias($id,$this->getBasePath());
                $this->preinit();
                $this->configure($config);
                $this->attachBehaviors($this->behaviors);
                $this->preloadComponents();
                $this->init();
```

```
}
public function __get($name)
{
        if($this->hasComponent($name))
                return $this->getComponent($name);
        else
                return parent::__get($name);
}
public function __isset($name)
{
        if($this->hasComponent($name))
                return $this->getComponent($name)!==null;
        else
                return parent::__isset($name);
}
public function getId()
{
        return $this->_id;
}
public function setId($id)
{
        $this->_id=$id;
}
public function getBasePath()
{
        if($this->_basePath===null)
        {
                $class=new ReflectionClass(get_class($this));
                $this->_basePath=dirname($class->getFileName());
        }
        return $this->_basePath;
}
public function setBasePath($path)
{
        if(($this->_basePath=realpath($path))===false || !is_dir($this->_basePath))
                throw new CException(Yii::t('yii','Base path "{path}" is not a valid directory.',
                        array('{path}'=>$path)));
}
public function getParams()
{
        if($this->_params!==null)
                return $this->_params;
        else
        {
                $this->_params=new CAttributeCollection;
                $this->_params->caseSensitive=true;
                return $this->_params;
            }
        }
        public function setParams($value)
        {
            $params=$this->getParams();
            foreach($value as $k=>$v)
                $params->add($k,$v);
        }
        public function getModulePath()
        {
            if($this->_modulePath!==null)
                return $this->_modulePath;
            else
                return
    $this->_modulePath=$this->getBasePath().DIRECTORY_SEPARATOR.'modules';
        }
        public function setModulePath($value)
        {
            if(($this->_modulePath=realpath($value))===false || !is_dir($this->_modulePath))
                throw new CException(Yii::t('yii','The module path "{path}" is not a valid
    directory.',
                    array('{path}'=>$value)));
        }
        public function setImport($aliases)
        {
            foreach($aliases as $alias)
                Yii::import($alias);
        }
        public function setAliases($mappings)
        {
            foreach($mappings as $name=>$alias)
            {
                if(($path=Yii::getPathOfAlias($alias))!==false)
                    Yii::setPathOfAlias($name,$path);
                else
                    Yii::setPathOfAlias($name,$alias);
            }
        }
        public function getParentModule()
        {
            return $this->_parentModule;
```

```
}
public function getModule($id)
{
        if(isset($this->_modules[$id]) || array_key_exists($id,$this->_modules))
                return $this->_modules[$id];
        elseif(isset($this->_moduleConfig[$id]))
        {
                $config=$this->_moduleConfig[$id];
                if(!isset($config['enabled']) || $config['enabled'])
                {
                        $class=$config['class'];
                        unset($config['class'], $config['enabled']);
                        if($this===Yii::app())
                                $module=Yii::createComponent($class,$id,null,$config);
                        else

$module=Yii::createComponent($class,$this->getId().'/'.$id,$this,$config);
                        return $this->_modules[$id]=$module;
                }
        }
}
public function hasModule($id)
{
        return isset($this->_moduleConfig[$id]) || isset($this->_modules[$id]);
}
public function getModules()
{
        return $this->_moduleConfig;
}
public function setModules($modules)
{
        foreach($modules as $id=>$module)
        {
                if(is_int($id))
                {
                        $id=$module;
                        $module=array();
                }
                if(!isset($module['class']))
                {
                        Yii::setPathOfAlias($id,$this->getModulePath().DIRECTORY_SEPARATOR.$id);
                        $module['class']=$id.'.'.ucfirst($id).'Module';
                }
                if(isset($this->_moduleConfig[$id]))
        $this->_moduleConfig[$id]=CMap::mergeArray($this->_moduleConfig[$id],$module);
                else
                    $this->_moduleConfig[$id]=$module;
            }
        }
        public function hasComponent($id)
        {
            return isset($this->_components[$id]) || isset($this->_componentConfig[$id]);
        }
        public function getComponent($id,$createIfNull=true)
        {
            if(isset($this->_components[$id]))
                return $this->_components[$id];
            elseif(isset($this->_componentConfig[$id]) && $createIfNull)
            {
                $config=$this->_componentConfig[$id];
                if(!isset($config['enabled']) || $config['enabled'])
                {
                    unset($config['enabled']);
                    $component=Yii::createComponent($config);
                    $component->init();
                    return $this->_components[$id]=$component;
                }
            }
        }
        public function setComponent($id,$component,$merge=true)
        {
            if($component===null)
            {
                unset($this->_components[$id]);
                return;
            }
            elseif($component instanceof IApplicationComponent)
            {
                $this->_components[$id]=$component;
                if(!$component->getIsInitialized())
                    $component->init();
                return;
            }
            elseif(isset($this->_components[$id]))
            {
                if(isset($component['class'])                                          &&
    get_class($this->_components[$id])!==$component['class'])
```

```
                {
                    unset($this->_components[$id]);
                    $this->_componentConfig[$id]=$component; //we should ignore merge here
                    return;
                }
                foreach($component as $key=>$value)
                {
                    if($key!=='class')
                        $this->_components[$id]->$key=$value;
                }
            }
            elseif(isset($this->_componentConfig[$id]['class'],$component['class'])
                && $this->_componentConfig[$id]['class']!==$component['class'])
            {
                $this->_componentConfig[$id]=$component; //we should ignore merge here
                return;
            }
            if(isset($this->_componentConfig[$id]) && $merge)

        $this->_componentConfig[$id]=CMap::mergeArray($this->_componentConfig[$id],$compon
ent);
            else
                $this->_componentConfig[$id]=$component;
        }
        public function getComponents($loadedOnly=true)
        {
            if($loadedOnly)
                return $this->_components;
            else
                return array_merge($this->_componentConfig, $this->_components);
        }
        public function setComponents($components,$merge=true)
        {
            foreach($components as $id=>$component)
                $this->setComponent($id,$component,$merge);
        }
        public function configure($config)
        {
            if(is_array($config))
            {
                foreach($config as $key=>$value)
                    $this->$key=$value;
            }
        }
```

```
	protected function preloadComponents()
	{
		foreach($this->preload as $id)
			$this->getComponent($id);
	}
	protected function preinit()
	{
	}
	protected function init()
	{
	}
}
abstract class CApplication extends CModule
{
	public $name='My Application';
	public $charset='UTF-8';
	public $sourceLanguage='en_us';
	private $_id;
	private $_basePath;
	private $_runtimePath;
	private $_extensionPath;
	private $_globalState;
	private $_stateChanged;
	private $_ended=false;
	private $_language;
	private $_homeUrl;
	abstract public function processRequest();
	public function __construct($config=null)
	{
		Yii::setApplication($this);
		// set basePath at early as possible to avoid trouble
		if(is_string($config))
			$config=require($config);
		if(isset($config['basePath']))
		{
			$this->setBasePath($config['basePath']);
			unset($config['basePath']);
		}
		else
			$this->setBasePath('protected');
		Yii::setPathOfAlias('application',$this->getBasePath());
		Yii::setPathOfAlias('webroot',dirname($_SERVER['SCRIPT_FILENAME']));
		Yii::setPathOfAlias('ext',$this->getBasePath().DIRECTORY_SEPARATOR.'extensions');
		$this->preinit();
```

```
        $this->initSystemHandlers();
        $this->registerCoreComponents();
        $this->configure($config);
        $this->attachBehaviors($this->behaviors);
        $this->preloadComponents();
        $this->init();
}
public function run()
{
        if($this->hasEventHandler('onBeginRequest'))
                $this->onBeginRequest(new CEvent($this));
        register_shutdown_function(array($this,'end'),0,false);
        $this->processRequest();
        if($this->hasEventHandler('onEndRequest'))
                $this->onEndRequest(new CEvent($this));
}
public function end($status=0,$exit=true)
{
        if($this->hasEventHandler('onEndRequest'))
                $this->onEndRequest(new CEvent($this));
        if($exit)
                exit($status);
}
public function onBeginRequest($event)
{
        $this->raiseEvent('onBeginRequest',$event);
}
public function onEndRequest($event)
{
        if(!$this->_ended)
        {
                $this->_ended=true;
                $this->raiseEvent('onEndRequest',$event);
        }
}
public function getId()
{
        if($this->_id!==null)
                return $this->_id;
        else
                return $this->_id=sprintf('%x',crc32($this->getBasePath().$this->name));
}
public function setId($id)
{
        $this->_id=$id;
    }
    public function getBasePath()
    {
        return $this->_basePath;
    }
    public function setBasePath($path)
    {
        if(($this->_basePath=realpath($path))===false || !is_dir($this->_basePath))
            throw new CException(Yii::t('yii','Application base path "{path}" is not a valid
directory.',
                array('{path}'=>$path)));
    }
    public function getRuntimePath()
    {
        if($this->_runtimePath!==null)
            return $this->_runtimePath;
        else
        {
            $this->setRuntimePath($this->getBasePath().DIRECTORY_SEPARATOR.'runtime');
            return $this->_runtimePath;
        }
    }
    public function setRuntimePath($path)
    {
        if(($runtimePath=realpath($path))===false          ||          !is_dir($runtimePath)
|| !is_writable($runtimePath))
            throw new CException(Yii::t('yii','Application runtime path "{path}" is not valid.
Please make sure it is a directory writable by the Web server process.',
                array('{path}'=>$path)));
        $this->_runtimePath=$runtimePath;
    }
    public function getExtensionPath()
    {
        return Yii::getPathOfAlias('ext');
    }
    public function setExtensionPath($path)
    {
        if(($extensionPath=realpath($path))===false || !is_dir($extensionPath))
            throw new CException(Yii::t('yii','Extension path "{path}" does not exist.',
                array('{path}'=>$path)));
        Yii::setPathOfAlias('ext',$extensionPath);
    }
    public function getLanguage()
```

```
	{
		return $this->_language===null ? $this->sourceLanguage : $this->_language;
	}
	public function setLanguage($language)
	{
		$this->_language=$language;
	}
	public function getTimeZone()
	{
		return date_default_timezone_get();
	}
	public function setTimeZone($value)
	{
		date_default_timezone_set($value);
	}
	public function findLocalizedFile($srcFile,$srcLanguage=null,$language=null)
	{
		if($srcLanguage===null)
			$srcLanguage=$this->sourceLanguage;
		if($language===null)
			$language=$this->getLanguage();
		if($language===$srcLanguage)
			return $srcFile;

	$desiredFile=dirname($srcFile).DIRECTORY_SEPARATOR.$language.DIRECTORY_SEPARATOR.
basename($srcFile);
		return is_file($desiredFile) ? $desiredFile : $srcFile;
	}
	public function getLocale($localeID=null)
	{
		return CLocale::getInstance($localeID===null?$this->getLanguage():$localeID);
	}
	public function getLocaleDataPath()
	{
		return     CLocale::$dataPath===null     ?     Yii::getPathOfAlias('system.i18n.data')     :
CLocale::$dataPath;
	}
	public function setLocaleDataPath($value)
	{
		CLocale::$dataPath=$value;
	}
	public function getNumberFormatter()
	{
		return $this->getLocale()->getNumberFormatter();
```

```
}
public function getDateFormatter()
{
	return $this->getLocale()->getDateFormatter();
}
public function getDb()
{
	return $this->getComponent('db');
}
public function getErrorHandler()
{
	return $this->getComponent('errorHandler');
}
public function getSecurityManager()
{
	return $this->getComponent('securityManager');
}
public function getStatePersister()
{
	return $this->getComponent('statePersister');
}
public function getCache()
{
	return $this->getComponent('cache');
}
public function getCoreMessages()
{
	return $this->getComponent('coreMessages');
}
public function getMessages()
{
	return $this->getComponent('messages');
}
public function getRequest()
{
	return $this->getComponent('request');
}
public function getUrlManager()
{
	return $this->getComponent('urlManager');
}
public function getController()
{
	return null;
```

```
}
public function createUrl($route,$params=array(),$ampersand='&')
{
	return $this->getUrlManager()->createUrl($route,$params,$ampersand);
}
public function createAbsoluteUrl($route,$params=array(),$schema='',$ampersand='&')
{
	$url=$this->createUrl($route,$params,$ampersand);
	if(strpos($url,'http')===0)
		return $url;
	else
		return $this->getRequest()->getHostInfo($schema).$url;
}
public function getBaseUrl($absolute=false)
{
	return $this->getRequest()->getBaseUrl($absolute);
}
public function getHomeUrl()
{
	if($this->_homeUrl===null)
	{
		if($this->getUrlManager()->showScriptName)
			return $this->getRequest()->getScriptUrl();
		else
			return $this->getRequest()->getBaseUrl().'/';
	}
	else
		return $this->_homeUrl;
}
public function setHomeUrl($value)
{
	$this->_homeUrl=$value;
}
public function getGlobalState($key,$defaultValue=null)
{
	if($this->_globalState===null)
		$this->loadGlobalState();
	if(isset($this->_globalState[$key]))
		return $this->_globalState[$key];
	else
		return $defaultValue;
}
public function setGlobalState($key,$value,$defaultValue=null)
{
        if($this->_globalState===null)
                $this->loadGlobalState();
        $changed=$this->_stateChanged;
        if($value===$defaultValue)
        {
                if(isset($this->_globalState[$key]))
                {
                        unset($this->_globalState[$key]);
                        $this->_stateChanged=true;
                }
        }
        elseif(!isset($this->_globalState[$key]) || $this->_globalState[$key]!==$value)
        {
                $this->_globalState[$key]=$value;
                $this->_stateChanged=true;
        }
        if($this->_stateChanged!==$changed)
                $this->attachEventHandler('onEndRequest',array($this,'saveGlobalState'));
}
public function clearGlobalState($key)
{
        $this->setGlobalState($key,true,true);
}
public function loadGlobalState()
{
        $persister=$this->getStatePersister();
        if(($this->_globalState=$persister->load())===null)
                $this->_globalState=array();
        $this->_stateChanged=false;
        $this->detachEventHandler('onEndRequest',array($this,'saveGlobalState'));
}
public function saveGlobalState()
{
        if($this->_stateChanged)
        {
                $this->_stateChanged=false;
                $this->detachEventHandler('onEndRequest',array($this,'saveGlobalState'));
                $this->getStatePersister()->save($this->_globalState);
        }
}
public function handleException($exception)
{
        // disable error capturing to avoid recursive errors
        restore_error_handler();
```

```
		restore_exception_handler();
		$category='exception.'.get_class($exception);
		if($exception instanceof CHttpException)
			$category.='.'.$exception->statusCode;
		// php <5.2 doesn't support string conversion auto-magically
		$message=$exception->__toString();
		if(isset($_SERVER['REQUEST_URI']))
			$message.="\nREQUEST_URI=".$_SERVER['REQUEST_URI'];
		if(isset($_SERVER['HTTP_REFERER']))
			$message.="\nHTTP_REFERER=".$_SERVER['HTTP_REFERER'];
		$message.="\n---";
		Yii::log($message,CLogger::LEVEL_ERROR,$category);
		try
		{
			$event=new CExceptionEvent($this,$exception);
			$this->onException($event);
			if(!$event->handled)
			{
				// try an error handler
				if(($handler=$this->getErrorHandler())!==null)
					$handler->handle($event);
				else
					$this->displayException($exception);
			}
		}
		catch(Exception $e)
		{
			$this->displayException($e);
		}
		try
		{
			$this->end(1);
		}
		catch(Exception $e)
		{
			// use the most primitive way to log error
			$msg = get_class($e).': '.$e->getMessage().' ('.$e->getFile().':'.$e->getLine().")\n";
			$msg .= $e->getTraceAsString()."\n";
			$msg .= "Previous exception:\n";
			$msg		.=		get_class($exception).':		'.$exception->getMessage().'
('.$exception->getFile().':'.$exception->getLine().")\n";
			$msg .= $exception->getTraceAsString()."\n";
			$msg .= '$_SERVER='.var_export($_SERVER,true);
			error_log($msg);
```

```
        exit(1);
    }
}
public function handleError($code,$message,$file,$line)
{
    if($code & error_reporting())
    {
        // disable error capturing to avoid recursive errors
        restore_error_handler();
        restore_exception_handler();
        $log="$message ($file:$line)\nStack trace:\n";
        $trace=debug_backtrace();
        // skip the first 3 stacks as they do not tell the error position
        if(count($trace)>3)
            $trace=array_slice($trace,3);
        foreach($trace as $i=>$t)
        {
            if(!isset($t['file']))
                $t['file']='unknown';
            if(!isset($t['line']))
                $t['line']=0;
            if(!isset($t['function']))
                $t['function']='unknown';
            $log.="#$i {$t['file']}({$t['line']}): ";
            if(isset($t['object']) && is_object($t['object']))
                $log.=get_class($t['object']).'->';
            $log.="{$t['function']}()\n";
        }
        if(isset($_SERVER['REQUEST_URI']))
            $log.='REQUEST_URI='.$_SERVER['REQUEST_URI'];
        Yii::log($log,CLogger::LEVEL_ERROR,'php');
        try
        {
            Yii::import('CErrorEvent',true);
            $event=new CErrorEvent($this,$code,$message,$file,$line);
            $this->onError($event);
            if(!$event->handled)
            {
                // try an error handler
                if(($handler=$this->getErrorHandler())!==null)
                    $handler->handle($event);
                else
                    $this->displayError($code,$message,$file,$line);
            }
            }
            catch(Exception $e)
            {
                $this->displayException($e);
            }
            try
            {
                $this->end(1);
            }
            catch(Exception $e)
            {
                // use the most primitive way to log error
                $msg            =            get_class($e).':            '.$e->getMessage().'
('.$e->getFile().':'.$e->getLine().")\n";
                $msg .= $e->getTraceAsString()."\n";
                $msg .= "Previous error:\n";
                $msg .= $log."\n";
                $msg .= '$_SERVER='.var_export($_SERVER,true);
                error_log($msg);
                exit(1);
            }
        }
    }
    public function onException($event)
    {
        $this->raiseEvent('onException',$event);
    }
    public function onError($event)
    {
        $this->raiseEvent('onError',$event);
    }
    public function displayError($code,$message,$file,$line)
    {
        if(YII_DEBUG)
        {
            echo "<h1>PHP Error [$code]</h1>\n";
            echo "<p>$message ($file:$line)</p>\n";
            echo '<pre>';
            $trace=debug_backtrace();
            // skip the first 3 stacks as they do not tell the error position
            if(count($trace)>3)
                $trace=array_slice($trace,3);
            foreach($trace as $i=>$t)
            {
```

```
				if(!isset($t['file']))
					$t['file']='unknown';
				if(!isset($t['line']))
					$t['line']=0;
				if(!isset($t['function']))
					$t['function']='unknown';
				echo "#$i {$t['file']}({$t['line']}): ";
				if(isset($t['object']) && is_object($t['object']))
					echo get_class($t['object']).'->';
				echo "{$t['function']}()\n";
			}
			echo '</pre>';
		}
		else
		{
			echo "<h1>PHP Error [$code]</h1>\n";
			echo "<p>$message</p>\n";
		}
	}
	public function displayException($exception)
	{
		if(YII_DEBUG)
		{
			echo '<h1>'.get_class($exception)."</h1>\n";
			echo                                                            '<p>'.$exception->getMessage().'
('.$exception->getFile().':'.$exception->getLine().')</p>';
			echo '<pre>'.$exception->getTraceAsString().'</pre>';
		}
		else
		{
			echo '<h1>'.get_class($exception)."</h1>\n";
			echo '<p>'.$exception->getMessage().'</p>';
		}
	}
	protected function initSystemHandlers()
	{
		if(YII_ENABLE_EXCEPTION_HANDLER)
			set_exception_handler(array($this,'handleException'));
		if(YII_ENABLE_ERROR_HANDLER)
			set_error_handler(array($this,'handleError'),error_reporting());
	}
	protected function registerCoreComponents()
	{
		$components=array(
```

```php
			'coreMessages'=>array(
				'class'=>'CPhpMessageSource',
				'language'=>'en_us',
				'basePath'=>YII_PATH.DIRECTORY_SEPARATOR.'messages',
			),
			'db'=>array(
				'class'=>'CDbConnection',
			),
			'messages'=>array(
				'class'=>'CPhpMessageSource',
			),
			'errorHandler'=>array(
				'class'=>'CErrorHandler',
			),
			'securityManager'=>array(
				'class'=>'CSecurityManager',
			),
			'statePersister'=>array(
				'class'=>'CStatePersister',
			),
			'urlManager'=>array(
				'class'=>'CUrlManager',
			),
			'request'=>array(
				'class'=>'CHttpRequest',
			),
			'format'=>array(
				'class'=>'CFormatter',
			),
		);
		$this->setComponents($components);
	}
}
class CWebApplication extends CApplication
{
	public $defaultController='site';
	public $layout='main';
	public $controllerMap=array();
	public $catchAllRequest;
	public $controllerNamespace;
	private $_controllerPath;
	private $_viewPath;
	private $_systemViewPath;
	private $_layoutPath;
private $_controller;
private $_theme;
public function processRequest()
{
	if(is_array($this->catchAllRequest) && isset($this->catchAllRequest[0]))
	{
		$route=$this->catchAllRequest[0];
		foreach(array_splice($this->catchAllRequest,1) as $name=>$value)
			$_GET[$name]=$value;
	}
	else
		$route=$this->getUrlManager()->parseUrl($this->getRequest());
	$this->runController($route);
}
protected function registerCoreComponents()
{
	parent::registerCoreComponents();
	$components=array(
		'session'=>array(
			'class'=>'CHttpSession',
		),
		'assetManager'=>array(
			'class'=>'CAssetManager',
		),
		'user'=>array(
			'class'=>'CWebUser',
		),
		'themeManager'=>array(
			'class'=>'CThemeManager',
		),
		'authManager'=>array(
			'class'=>'CPhpAuthManager',
		),
		'clientScript'=>array(
			'class'=>'CClientScript',
		),
		'widgetFactory'=>array(
			'class'=>'CWidgetFactory',
		),
	);
	$this->setComponents($components);
}
public function getAuthManager()
{
        return $this->getComponent('authManager');
}
public function getAssetManager()
{
        return $this->getComponent('assetManager');
}
public function getSession()
{
        return $this->getComponent('session');
}
public function getUser()
{
        return $this->getComponent('user');
}
public function getViewRenderer()
{
        return $this->getComponent('viewRenderer');
}
public function getClientScript()
{
        return $this->getComponent('clientScript');
}
public function getWidgetFactory()
{
        return $this->getComponent('widgetFactory');
}
public function getThemeManager()
{
        return $this->getComponent('themeManager');
}
public function getTheme()
{
        if(is_string($this->_theme))
                $this->_theme=$this->getThemeManager()->getTheme($this->_theme);
        return $this->_theme;
}
public function setTheme($value)
{
        $this->_theme=$value;
}
public function runController($route)
{
        if(($ca=$this->createController($route))!==null)
        {
            list($controller,$actionID)=$ca;
            $oldController=$this->_controller;
            $this->_controller=$controller;
            $controller->init();
            $controller->run($actionID);
            $this->_controller=$oldController;
        }
        else
            throw new CHttpException(404,Yii::t('yii','Unable to resolve the request "{route}".',
                array('{route}'=>$route===''?$this->defaultController:$route)));
    }
    public function createController($route,$owner=null)
    {
        if($owner===null)
            $owner=$this;
        if(($route=trim($route,'/'))==='')
            $route=$owner->defaultController;
        $caseSensitive=$this->getUrlManager()->caseSensitive;
        $route.='/';
        while(($pos=strpos($route,'/'))!==false)
        {
            $id=substr($route,0,$pos);
            if(!preg_match('/^\w+$/',$id))
                return null;
            if(!$caseSensitive)
                $id=strtolower($id);
            $route=(string)substr($route,$pos+1);
            if(!isset($basePath))    // first segment
            {
                if(isset($owner->controllerMap[$id]))
                {
                    return array(

    Yii::createComponent($owner->controllerMap[$id],$id,$owner===$this?null:$owner),
                        $this->parseActionParams($route),
                    );
                }
                if(($module=$owner->getModule($id))!==null)
                    return $this->createController($route,$module);
                $basePath=$owner->getControllerPath();
                $controllerID='';
            }
            else
                $controllerID.='/';
            $className=ucfirst($id).'Controller';
            $classFile=$basePath.DIRECTORY_SEPARATOR.$className.'.php';
            if($owner->controllerNamespace!==null)
                $className=$owner->controllerNamespace.'\\'.$className;
            if(is_file($classFile))
            {
                if(!class_exists($className,false))
                    require($classFile);
                if(class_exists($className,false)                                   &&
is_subclass_of($className,'CController'))
                {
                    $id[0]=strtolower($id[0]);
                    return array(
                        new $className($controllerID.$id,$owner===$this?null:$owner),
                        $this->parseActionParams($route),
                    );
                }
                return null;
            }
            $controllerID.=$id;
            $basePath.=DIRECTORY_SEPARATOR.$id;
        }
    }
    protected function parseActionParams($pathInfo)
    {
        if(($pos=strpos($pathInfo,'/'))!==false)
        {
            $manager=$this->getUrlManager();
            $manager->parsePathInfo((string)substr($pathInfo,$pos+1));
            $actionID=substr($pathInfo,0,$pos);
            return $manager->caseSensitive ? $actionID : strtolower($actionID);
        }
        else
            return $pathInfo;
    }
    public function getController()
    {
        return $this->_controller;
    }
    public function setController($value)
    {
        $this->_controller=$value;
    }
    public function getControllerPath()
```

```
	{
		if($this->_controllerPath!==null)
			return $this->_controllerPath;
		else
			return
$this->_controllerPath=$this->getBasePath().DIRECTORY_SEPARATOR.'controllers';
	}
	public function setControllerPath($value)
	{
		if(($this->_controllerPath=realpath($value))===false || !is_dir($this->_controllerPath))
			throw new CException(Yii::t('yii','The controller path "{path}" is not a valid
directory.',
				array('{path}'=>$value)));
	}
	public function getViewPath()
	{
		if($this->_viewPath!==null)
			return $this->_viewPath;
		else
			return $this->_viewPath=$this->getBasePath().DIRECTORY_SEPARATOR.'views';
	}
	public function setViewPath($path)
	{
		if(($this->_viewPath=realpath($path))===false || !is_dir($this->_viewPath))
			throw new CException(Yii::t('yii','The view path "{path}" is not a valid directory.',
				array('{path}'=>$path)));
	}
	public function getSystemViewPath()
	{
		if($this->_systemViewPath!==null)
			return $this->_systemViewPath;
		else
			return
$this->_systemViewPath=$this->getViewPath().DIRECTORY_SEPARATOR.'system';
	}
	public function setSystemViewPath($path)
	{
		if(($this->_systemViewPath=realpath($path))===false
|| !is_dir($this->_systemViewPath))
			throw new CException(Yii::t('yii','The system view path "{path}" is not a valid
directory.',
				array('{path}'=>$path)));
	}
	public function getLayoutPath()
	{
		if($this->_layoutPath!==null)
			return $this->_layoutPath;
		else
			return $this->_layoutPath=$this->getViewPath().DIRECTORY_SEPARATOR.'layouts';
	}
	public function setLayoutPath($path)
	{
		if(($this->_layoutPath=realpath($path))===false || !is_dir($this->_layoutPath))
			throw new CException(Yii::t('yii','The layout path "{path}" is not a valid directory.',
				array('{path}'=>$path)));
	}
	public function beforeControllerAction($controller,$action)
	{
		return true;
	}
	public function afterControllerAction($controller,$action)
	{
	}
	public function findModule($id)
	{
		if(($controller=$this->getController())!==null                                        &&
($module=$controller->getModule())!==null)
		{
			do
			{
				if(($m=$module->getModule($id))!==null)
					return $m;
			} while(($module=$module->getParentModule())!==null);
		}
		if(($m=$this->getModule($id))!==null)
			return $m;
	}
	protected function init()
	{
		parent::init();
		// preload 'request' so that it has chance to respond to onBeginRequest event.
		$this->getRequest();
	}
}
class CMap extends CComponent implements IteratorAggregate,ArrayAccess,Countable
{
	private $_d=array();
	private $_r=false;
```

```
public function __construct($data=null,$readOnly=false)
{
        if($data!==null)
                $this->copyFrom($data);
        $this->setReadOnly($readOnly);
}
public function getReadOnly()
{
        return $this->_r;
}
protected function setReadOnly($value)
{
        $this->_r=$value;
}
public function getIterator()
{
        return new CMapIterator($this->_d);
}
public function count()
{
        return $this->getCount();
}
public function getCount()
{
        return count($this->_d);
}
public function getKeys()
{
        return array_keys($this->_d);
}
public function itemAt($key)
{
        if(isset($this->_d[$key]))
                return $this->_d[$key];
        else
                return null;
}
public function add($key,$value)
{
        if(!$this->_r)
        {
                if($key===null)
                        $this->_d[]=$value;
                else
                $this->_d[$key]=$value;
        }
        else
            throw new CException(Yii::t('yii','The map is read only.'));
}
public function remove($key)
{
        if(!$this->_r)
        {
            if(isset($this->_d[$key]))
            {
                $value=$this->_d[$key];
                unset($this->_d[$key]);
                return $value;
            }
            else
            {
                // it is possible the value is null, which is not detected by isset
                unset($this->_d[$key]);
                return null;
            }
        }
        else
            throw new CException(Yii::t('yii','The map is read only.'));
}
public function clear()
{
        foreach(array_keys($this->_d) as $key)
            $this->remove($key);
}
public function contains($key)
{
        return isset($this->_d[$key]) || array_key_exists($key,$this->_d);
}
public function toArray()
{
        return $this->_d;
}
public function copyFrom($data)
{
        if(is_array($data) || $data instanceof Traversable)
        {
            if($this->getCount()>0)
                $this->clear();
            if($data instanceof CMap)
                $data=$data->_d;
            foreach($data as $key=>$value)
                $this->add($key,$value);
        }
        elseif($data!==null)
            throw new CException(Yii::t('yii','Map data must be an array or an object
implementing Traversable.'));
    }
    public function mergeWith($data,$recursive=true)
    {
        if(is_array($data) || $data instanceof Traversable)
        {
            if($data instanceof CMap)
                $data=$data->_d;
            if($recursive)
            {
                if($data instanceof Traversable)
                {
                    $d=array();
                    foreach($data as $key=>$value)
                        $d[$key]=$value;
                    $this->_d=self::mergeArray($this->_d,$d);
                }
                else
                    $this->_d=self::mergeArray($this->_d,$data);
            }
            else
            {
                foreach($data as $key=>$value)
                    $this->add($key,$value);
            }
        }
        elseif($data!==null)
            throw new CException(Yii::t('yii','Map data must be an array or an object
implementing Traversable.'));
    }
    public static function mergeArray($a,$b)
    {
        $args=func_get_args();
        $res=array_shift($args);
        while(!empty($args))
        {
            $next=array_shift($args);
```

```
			foreach($next as $k => $v)
			{
				if(is_integer($k))
					isset($res[$k]) ? $res[]=$v : $res[$k]=$v;
				elseif(is_array($v) && isset($res[$k]) && is_array($res[$k]))
					$res[$k]=self::mergeArray($res[$k],$v);
				else
					$res[$k]=$v;
			}
		}
		return $res;
	}
	public function offsetExists($offset)
	{
		return $this->contains($offset);
	}
	public function offsetGet($offset)
	{
		return $this->itemAt($offset);
	}
	public function offsetSet($offset,$item)
	{
		$this->add($offset,$item);
	}
	public function offsetUnset($offset)
	{
		$this->remove($offset);
	}
}
class CLogger extends CComponent
{
	const LEVEL_TRACE='trace';
	const LEVEL_WARNING='warning';
	const LEVEL_ERROR='error';
	const LEVEL_INFO='info';
	const LEVEL_PROFILE='profile';
	public $autoFlush=10000;
	public $autoDump=false;
	private $_logs=array();
	private $_logCount=0;
	private $_levels;
	private $_categories;
	private $_except=array();
	private $_timings;
private $_processing=false;
public function log($message,$level='info',$category='application')
{
        $this->_logs[]=array($message,$level,$category,microtime(true));
        $this->_logCount++;
        if($this->autoFlush>0 && $this->_logCount>=$this->autoFlush && !$this->_processing)
        {
                $this->_processing=true;
                $this->flush($this->autoDump);
                $this->_processing=false;
        }
}
public function getLogs($levels='',$categories=array(), $except=array())
{
        $this->_levels=preg_split('/[\s,]+/',strtolower($levels),-1,PREG_SPLIT_NO_EMPTY);
        if (is_string($categories))

$this->_categories=preg_split('/[\s,]+/',strtolower($categories),-1,PREG_SPLIT_NO_EMPTY);
        else
                $this->_categories=array_filter(array_map('strtolower',$categories));
        if (is_string($except))

$this->_except=preg_split('/[\s,]+/',strtolower($except),-1,PREG_SPLIT_NO_EMPTY);
        else
                $this->_except=array_filter(array_map('strtolower',$except));
        $ret=$this->_logs;
        if(!empty($levels))
                $ret=array_values(array_filter($ret,array($this,'filterByLevel')));
        if(!empty($this->_categories) || !empty($this->_except))
                $ret=array_values(array_filter($ret,array($this,'filterByCategory')));
        return $ret;
}
private function filterByCategory($value)
{
        return $this->filterAllCategories($value, 2);
}
private function filterTimingByCategory($value)
{
        return $this->filterAllCategories($value, 1);
}
private function filterAllCategories($value, $index)
{
        $cat=strtolower($value[$index]);
        $ret=empty($this->_categories);
        foreach($this->_categories as $category)
        {
            if($cat===$category || (($c=rtrim($category,'.*'))!==$category &&
strpos($cat,$c)===0))
                $ret=true;
        }
        if($ret)
        {
            foreach($this->_except as $category)
            {
                if($cat===$category || (($c=rtrim($category,'.*'))!==$category &&
strpos($cat,$c)===0))
                    $ret=false;
            }
        }
        return $ret;
    }
    private function filterByLevel($value)
    {
        return in_array(strtolower($value[1]),$this->_levels);
    }
    public function getExecutionTime()
    {
        return microtime(true)-YII_BEGIN_TIME;
    }
    public function getMemoryUsage()
    {
        if(function_exists('memory_get_usage'))
            return memory_get_usage();
        else
        {
            $output=array();
            if(strncmp(PHP_OS,'WIN',3)===0)
            {
                exec('tasklist /FI "PID eq ' . getmypid() . '" /FO LIST',$output);
                return isset($output[5])?preg_replace('/[\D]/','',$output[5])*1024 : 0;
            }
            else
            {
                $pid=getmypid();
                exec("ps -eo%mem,rss,pid | grep $pid", $output);
                $output=explode("    ",$output[0]);
                return isset($output[1]) ? $output[1]*1024 : 0;
            }
```

```
	}
}
public function getProfilingResults($token=null,$categories=null,$refresh=false)
{
	if($this->_timings===null || $refresh)
		$this->calculateTimings();
	if($token===null && $categories===null)
		return $this->_timings;
	$timings = $this->_timings;
	if($categories!==null) {

$this->_categories=preg_split('/[\s,]+/',strtolower($categories),-1,PREG_SPLIT_NO_EMPTY);
		$timings=array_filter($timings,array($this,'filterTimingByCategory'));
	}
	$results=array();
	foreach($timings as $timing)
	{
		if($token===null || $timing[0]===$token)
			$results[]=$timing[2];
	}
	return $results;
}
private function calculateTimings()
{
	$this->_timings=array();
	$stack=array();
	foreach($this->_logs as $log)
	{
		if($log[1]!==CLogger::LEVEL_PROFILE)
			continue;
		list($message,$level,$category,$timestamp)=$log;
		if(!strncasecmp($message,'begin:',6))
		{
			$log[0]=substr($message,6);
			$stack[]=$log;
		}
		elseif(!strncasecmp($message,'end:',4))
		{
			$token=substr($message,4);
			if(($last=array_pop($stack))!==null && $last[0]===$token)
			{
				$delta=$log[3]-$last[3];
				$this->_timings[]=array($message,$category,$delta);
			}
```

```
                    else
                        throw new CException(Yii::t('yii','CProfileLogRoute found a mismatching
code block "{token}". Make sure the calls to Yii::beginProfile() and Yii::endProfile() be properly
nested.',
                            array('{token}'=>$token)));
                }
            }
            $now=microtime(true);
            while(($last=array_pop($stack))!==null)
            {
                $delta=$now-$last[3];
                $this->_timings[]=array($last[0],$last[2],$delta);
            }
        }
        public function flush($dumpLogs=false)
        {
            $this->onFlush(new CEvent($this, array('dumpLogs'=>$dumpLogs)));
            $this->_logs=array();
            $this->_logCount=0;
        }
        public function onFlush($event)
        {
            $this->raiseEvent('onFlush', $event);
        }
}
abstract class CApplicationComponent extends CComponent implements IApplicationComponent
{
        public $behaviors=array();
        private $_initialized=false;
        public function init()
        {
            $this->attachBehaviors($this->behaviors);
            $this->_initialized=true;
        }
        public function getIsInitialized()
        {
            return $this->_initialized;
        }
}
class CHttpRequest extends CApplicationComponent
{
        public $enableCookieValidation=false;
        public $enableCsrfValidation=false;
        public $csrfTokenName='YII_CSRF_TOKEN';
        public $csrfCookie;
        private $_requestUri;
        private $_pathInfo;
        private $_scriptFile;
        private $_scriptUrl;
        private $_hostInfo;
        private $_baseUrl;
        private $_cookies;
        private $_preferredLanguages;
        private $_csrfToken;
        private $_restParams;
        public function init()
        {
                parent::init();
                $this->normalizeRequest();
        }
        protected function normalizeRequest()
        {
                // normalize request
                if(function_exists('get_magic_quotes_gpc') && get_magic_quotes_gpc())
                {
                        if(isset($_GET))
                                $_GET=$this->stripSlashes($_GET);
                        if(isset($_POST))
                                $_POST=$this->stripSlashes($_POST);
                        if(isset($_REQUEST))
                                $_REQUEST=$this->stripSlashes($_REQUEST);
                        if(isset($_COOKIE))
                                $_COOKIE=$this->stripSlashes($_COOKIE);
                }
                if($this->enableCsrfValidation)
                        Yii::app()->attachEventHandler('onBeginRequest',array($this,'validateCsrfToken'));
        }
        public function stripSlashes(&$data)
        {
                return is_array($data)?array_map(array($this,'stripSlashes'),$data):stripslashes($data);
        }
        public function getParam($name,$defaultValue=null)
        {
                return    isset($_GET[$name])    ?    $_GET[$name]    :    (isset($_POST[$name])    ?
$_POST[$name] : $defaultValue);
        }
        public function getQuery($name,$defaultValue=null)
        {
                return isset($_GET[$name]) ? $_GET[$name] : $defaultValue;
        }
        public function getPost($name,$defaultValue=null)
        {
                return isset($_POST[$name]) ? $_POST[$name] : $defaultValue;
        }
        public function getDelete($name,$defaultValue=null)
        {
                if($this->getIsDeleteViaPostRequest())
                        return $this->getPost($name, $defaultValue);
                if($this->getIsDeleteRequest())
                {
                        $this->getRestParams();
                        return     isset($this->_restParams[$name])     ?     $this->_restParams[$name]     :
$defaultValue;
                }
                else
                        return $defaultValue;
        }
        public function getPut($name,$defaultValue=null)
        {
                if($this->getIsPutViaPostRequest())
                        return $this->getPost($name, $defaultValue);
                if($this->getIsPutRequest())
                {
                        $this->getRestParams();
                        return     isset($this->_restParams[$name])     ?     $this->_restParams[$name]     :
$defaultValue;
                }
                else
                        return $defaultValue;
        }
        public function getRestParams()
        {
                if($this->_restParams===null)
                {
                        $result=array();
                        if(function_exists('mb_parse_str'))
                                mb_parse_str($this->getRawBody(), $result);
                        else
                                parse_str($this->getRawBody(), $result);
                        $this->_restParams=$result;
                }
                return $this->_restParams;
```

```
	}
	public function getRawBody()
	{
		static $rawBody;
		if($rawBody===null)
			$rawBody=file_get_contents('php://input');
		return $rawBody;
	}
	public function getUrl()
	{
		return $this->getRequestUri();
	}
	public function getHostInfo($schema='')
	{
		if($this->_hostInfo===null)
		{
			if($secure=$this->getIsSecureConnection())
				$http='https';
			else
				$http='http';
			if(isset($_SERVER['HTTP_HOST']))
				$this->_hostInfo=$http.'://'.$_SERVER['HTTP_HOST'];
			else
			{
				$this->_hostInfo=$http.'://'.$_SERVER['SERVER_NAME'];
				$port=$secure ? $this->getSecurePort() : $this->getPort();
				if(($port!==80 && !$secure) || ($port!==443 && $secure))
					$this->_hostInfo.=':'.$port;
			}
		}
		if($schema!=='')
		{
			$secure=$this->getIsSecureConnection();
			if($secure && $schema==='https' || !$secure && $schema==='http')
				return $this->_hostInfo;
			$port=$schema==='https' ? $this->getSecurePort() : $this->getPort();
			if($port!==80 && $schema==='http' || $port!==443 && $schema==='https')
				$port=':'.$port;
			else
				$port='';
			$pos=strpos($this->_hostInfo,':');
			return
$schema.substr($this->_hostInfo,$pos,strcspn($this->_hostInfo,':',$pos+1)+1).$port;
		}
		else
			return $this->_hostInfo;
	}
	public function setHostInfo($value)
	{
		$this->_hostInfo=rtrim($value,'/');
	}
	public function getBaseUrl($absolute=false)
	{
		if($this->_baseUrl===null)
			$this->_baseUrl=rtrim(dirname($this->getScriptUrl()),'\\/');
		return $absolute ? $this->getHostInfo() . $this->_baseUrl : $this->_baseUrl;
	}
	public function setBaseUrl($value)
	{
		$this->_baseUrl=$value;
	}
	public function getScriptUrl()
	{
		if($this->_scriptUrl===null)
		{
			$scriptName=basename($_SERVER['SCRIPT_FILENAME']);
			if(basename($_SERVER['SCRIPT_NAME'])===$scriptName)
				$this->_scriptUrl=$_SERVER['SCRIPT_NAME'];
			elseif(basename($_SERVER['PHP_SELF'])===$scriptName)
				$this->_scriptUrl=$_SERVER['PHP_SELF'];
			elseif(isset($_SERVER['ORIG_SCRIPT_NAME'])                    &&
basename($_SERVER['ORIG_SCRIPT_NAME'])===$scriptName)
				$this->_scriptUrl=$_SERVER['ORIG_SCRIPT_NAME'];
			elseif(($pos=strpos($_SERVER['PHP_SELF'],'/'.$scriptName))!==false)
				$this->_scriptUrl=substr($_SERVER['SCRIPT_NAME'],0,$pos).'/'.$scriptName;
			elseif(isset($_SERVER['DOCUMENT_ROOT'])                       &&
strpos($_SERVER['SCRIPT_FILENAME'],$_SERVER['DOCUMENT_ROOT'])===0)

	$this->_scriptUrl=str_replace('\\','/',str_replace($_SERVER['DOCUMENT_ROOT'],'',$_SERVER
['SCRIPT_FILENAME']));
			else
				throw  new  CException(Yii::t('yii','CHttpRequest  is  unable  to  determine  the
entry script URL.'));
		}
		return $this->_scriptUrl;
	}
	public function setScriptUrl($value)
	{
```

```
            $this->_scriptUrl='/'.trim($value,'/');
        }
        public function getPathInfo()
        {
            if($this->_pathInfo===null)
            {
                $pathInfo=$this->getRequestUri();
                if(($pos=strpos($pathInfo,'?'))!==false)
                   $pathInfo=substr($pathInfo,0,$pos);
                $pathInfo=$this->decodePathInfo($pathInfo);
                $scriptUrl=$this->getScriptUrl();
                $baseUrl=$this->getBaseUrl();
                if(strpos($pathInfo,$scriptUrl)===0)
                    $pathInfo=substr($pathInfo,strlen($scriptUrl));
                elseif($baseUrl==='' || strpos($pathInfo,$baseUrl)===0)
                    $pathInfo=substr($pathInfo,strlen($baseUrl));
                elseif(strpos($_SERVER['PHP_SELF'],$scriptUrl)===0)
                    $pathInfo=substr($_SERVER['PHP_SELF'],strlen($scriptUrl));
                else
                    throw new CException(Yii::t('yii','CHttpRequest is unable to determine the
path info of the request.'));
                $this->_pathInfo=trim($pathInfo,'/');
            }
            return $this->_pathInfo;
        }
        protected function decodePathInfo($pathInfo)
        {
            $pathInfo = urldecode($pathInfo);
            // is it UTF-8?
            // http://w3.org/International/questions/qa-forms-utf-8.html
            if(preg_match('%^(?:
               [\x09\x0A\x0D\x20-\x7E]              # ASCII
             | [\xC2-\xDF][\x80-\xBF]               # non-overlong 2-byte
             | \xE0[\xA0-\xBF][\x80-\xBF]           # excluding overlongs
             | [\xE1-\xEC\xEE\xEF][\x80-\xBF]{2}    # straight 3-byte
             | \xED[\x80-\x9F][\x80-\xBF]           # excluding surrogates
             | \xF0[\x90-\xBF][\x80-\xBF]{2}        # planes 1-3
             | [\xF1-\xF3][\x80-\xBF]{3}            # planes 4-15
             | \xF4[\x80-\x8F][\x80-\xBF]{2}        # plane 16
            )*$%xs', $pathInfo))
            {
                return $pathInfo;
            }
            else
```

```
		{
			return utf8_encode($pathInfo);
		}
	}
	public function getRequestUri()
	{
		if($this->_requestUri===null)
		{
			if(isset($_SERVER['HTTP_X_REWRITE_URL'])) // IIS
				$this->_requestUri=$_SERVER['HTTP_X_REWRITE_URL'];
			elseif(isset($_SERVER['REQUEST_URI']))
			{
				$this->_requestUri=$_SERVER['REQUEST_URI'];
				if(!empty($_SERVER['HTTP_HOST']))
				{
					if(strpos($this->_requestUri,$_SERVER['HTTP_HOST'])!==false)

	$this->_requestUri=preg_replace('/^\w+:\/\/[^\/]+/','',$this->_requestUri);
				}
				else

	$this->_requestUri=preg_replace('/^(http|https):\/\/[^\/]+/i','',$this->_requestUri);
			}
			elseif(isset($_SERVER['ORIG_PATH_INFO']))	// IIS 5.0 CGI
			{
				$this->_requestUri=$_SERVER['ORIG_PATH_INFO'];
				if(!empty($_SERVER['QUERY_STRING']))
					$this->_requestUri.='?'.$_SERVER['QUERY_STRING'];
			}
			else
				throw new CException(Yii::t('yii','CHttpRequest is unable to determine the
request URI.'));
		}
		return $this->_requestUri;
	}
	public function getQueryString()
	{
		return isset($_SERVER['QUERY_STRING'])?$_SERVER['QUERY_STRING']:'';
	}
	public function getIsSecureConnection()
	{
		return !empty($_SERVER['HTTPS']) && strcasecmp($_SERVER['HTTPS'],'off');
	}
	public function getRequestType()
	{
		if(isset($_POST['_method']))
			return strtoupper($_POST['_method']);
		return
strtoupper(isset($_SERVER['REQUEST_METHOD'])?$_SERVER['REQUEST_METHOD']:'GET');
	}
	public function getIsPostRequest()
	{
		return isset($_SERVER['REQUEST_METHOD'])
&& !strcasecmp($_SERVER['REQUEST_METHOD'],'POST');
	}
	public function getIsDeleteRequest()
	{
		return (isset($_SERVER['REQUEST_METHOD'])
&& !strcasecmp($_SERVER['REQUEST_METHOD'],'DELETE')) ||
$this->getIsDeleteViaPostRequest();
	}
	protected function getIsDeleteViaPostRequest()
	{
		return isset($_POST['_method']) && !strcasecmp($_POST['_method'],'DELETE');
	}
	public function getIsPutRequest()
	{
		return (isset($_SERVER['REQUEST_METHOD'])
&& !strcasecmp($_SERVER['REQUEST_METHOD'],'PUT')) || $this->getIsPutViaPostRequest();
	}
	protected function getIsPutViaPostRequest()
	{
		return isset($_POST['_method']) && !strcasecmp($_POST['_method'],'PUT');
	}
	public function getIsAjaxRequest()
	{
		return isset($_SERVER['HTTP_X_REQUESTED_WITH']) &&
$_SERVER['HTTP_X_REQUESTED_WITH']==='XMLHttpRequest';
	}
	public function getIsFlashRequest()
	{
		return isset($_SERVER['HTTP_USER_AGENT']) &&
(stripos($_SERVER['HTTP_USER_AGENT'],'Shockwave')!==false ||
stripos($_SERVER['HTTP_USER_AGENT'],'Flash')!==false);
	}
	public function getServerName()
	{
		return $_SERVER['SERVER_NAME'];
	}
	public function getServerPort()
	{
		return $_SERVER['SERVER_PORT'];
	}
	public function getUrlReferrer()
	{
		return isset($_SERVER['HTTP_REFERER'])?$_SERVER['HTTP_REFERER']:null;
	}
	public function getUserAgent()
	{
		return isset($_SERVER['HTTP_USER_AGENT'])?$_SERVER['HTTP_USER_AGENT']:null;
	}
	public function getUserHostAddress()
	{
		return isset($_SERVER['REMOTE_ADDR'])?$_SERVER['REMOTE_ADDR']:'127.0.0.1';
	}
	public function getUserHost()
	{
		return isset($_SERVER['REMOTE_HOST'])?$_SERVER['REMOTE_HOST']:null;
	}
	public function getScriptFile()
	{
		if($this->_scriptFile!==null)
			return $this->_scriptFile;
		else
			return $this->_scriptFile=realpath($_SERVER['SCRIPT_FILENAME']);
	}
	public function getBrowser($userAgent=null)
	{
		return get_browser($userAgent,true);
	}
	public function getAcceptTypes()
	{
		return isset($_SERVER['HTTP_ACCEPT'])?$_SERVER['HTTP_ACCEPT']:null;
	}
	private $_port;
	public function getPort()
	{
		if($this->_port===null)
			$this->_port=!$this->getIsSecureConnection()                                   &&
isset($_SERVER['SERVER_PORT']) ? (int)$_SERVER['SERVER_PORT'] : 80;
		return $this->_port;
	}
	public function setPort($value)
	{
		$this->_port=(int)$value;
		$this->_hostInfo=null;
	}
	private $_securePort;
	public function getSecurePort()
	{
		if($this->_securePort===null)
			$this->_securePort=$this->getIsSecureConnection()                              &&
isset($_SERVER['SERVER_PORT']) ? (int)$_SERVER['SERVER_PORT'] : 443;
		return $this->_securePort;
	}
	public function setSecurePort($value)
	{
		$this->_securePort=(int)$value;
		$this->_hostInfo=null;
	}
	public function getCookies()
	{
		if($this->_cookies!==null)
			return $this->_cookies;
		else
			return $this->_cookies=new CCookieCollection($this);
	}
	public function redirect($url,$terminate=true,$statusCode=302)
	{
		if(strpos($url,'/')===0 && strpos($url,'//')!==0)
			$url=$this->getHostInfo().$url;
		header('Location: '.$url, true, $statusCode);
		if($terminate)
			Yii::app()->end();
	}
	public function getPreferredLanguages()
	{
		if($this->_preferredLanguages===null)
		{
			$sortedLanguages=array();
			if(isset($_SERVER['HTTP_ACCEPT_LANGUAGE'])                              &&
$n=preg_match_all('/([\w\-_]+)(?:\s*;\s*q\s*=\s*(\d*\.?\d*))?/',$_SERVER['HTTP_ACCEPT_LANG
UAGE'],$matches))
			{
				$languages=array();
				for($i=0;$i<$n;++$i)
```

```
                {
                    $q=$matches[2][$i];
                    if($q==='')
                        $q=1;
                    if($q)
                        $languages[]=array((float)$q,$matches[1][$i]);
                }
                usort($languages,create_function('$a,$b','if($a[0]==$b[0])  {return  0;}  return
($a[0]<$b[0]) ? 1 : -1;'));
                foreach($languages as $language)
                    $sortedLanguages[]=$language[1];
            }
            $this->_preferredLanguages=$sortedLanguages;
        }
        return $this->_preferredLanguages;
    }
    public function getPreferredLanguage()
    {
        $preferredLanguages=$this->getPreferredLanguages();
        return                              !empty($preferredLanguages)                              ?
CLocale::getCanonicalID($preferredLanguages[0]) : false;
    }
    public function sendFile($fileName,$content,$mimeType=null,$terminate=true)
    {
        if($mimeType===null)
        {
            if(($mimeType=CFileHelper::getMimeTypeByExtension($fileName))===null)
                $mimeType='text/plain';
        }
        header('Pragma: public');
        header('Expires: 0');
        header('Cache-Control: must-revalidate, post-check=0, pre-check=0');
        header("Content-type: $mimeType");
        header('Content-Length:  '.(function_exists('mb_strlen')  ?  mb_strlen($content,'8bit')  :
strlen($content)));
        header("Content-Disposition: attachment; filename=\"$fileName\"");
        header('Content-Transfer-Encoding: binary');
        if($terminate)
        {
            // clean up the application first because the file downloading could take long time
            // which may cause timeout of some resources (such as DB connection)
            ob_start();
            Yii::app()->end(0,false);
            ob_end_clean();
```

```
                echo $content;
                exit(0);
        }
        else
                echo $content;
}
public function xSendFile($filePath, $options=array())
{
        if(!isset($options['forceDownload']) || $options['forceDownload'])
                $disposition='attachment';
        else
                $disposition='inline';
        if(!isset($options['saveName']))
                $options['saveName']=basename($filePath);
        if(!isset($options['mimeType']))
        {

if(($options['mimeType']=CFileHelper::getMimeTypeByExtension($filePath))===null)
                        $options['mimeType']='text/plain';
        }
        if(!isset($options['xHeader']))
                $options['xHeader']='X-Sendfile';
        if($options['mimeType'] !== null)
                header('Content-type: '.$options['mimeType']);
        header('Content-Disposition: '.$disposition.'; filename="'.$options['saveName'].'"');
        if(isset($options['addHeaders']))
        {
                foreach($options['addHeaders'] as $header=>$value)
                        header($header.': '.$value);
        }
        header(trim($options['xHeader']).': '.$filePath);
        if(!isset($options['terminate']) || $options['terminate'])
                Yii::app()->end();
}
public function getCsrfToken()
{
        if($this->_csrfToken===null)
        {
                $cookie=$this->getCookies()->itemAt($this->csrfTokenName);
                if(!$cookie || ($this->_csrfToken=$cookie->value)==null)
                {
                        $cookie=$this->createCsrfCookie();
                        $this->_csrfToken=$cookie->value;
                        $this->getCookies()->add($cookie->name,$cookie);
```

```
            }
        }
        return $this->_csrfToken;
    }
    protected function createCsrfCookie()
    {
        $cookie=new CHttpCookie($this->csrfTokenName,sha1(uniqid(mt_rand(),true)));
        if(is_array($this->csrfCookie))
        {
            foreach($this->csrfCookie as $name=>$value)
                $cookie->$name=$value;
        }
        return $cookie;
    }
    public function validateCsrfToken($event)
    {
        if ($this->getIsPostRequest() ||
            $this->getIsPutRequest() ||
            $this->getIsDeleteRequest())
        {
            $cookies=$this->getCookies();
            $method=$this->getRequestType();
            switch($method)
            {
                case 'POST':
                    $userToken=$this->getPost($this->csrfTokenName);
                break;
                case 'PUT':
                    $userToken=$this->getPut($this->csrfTokenName);
                break;
                case 'DELETE':
                    $userToken=$this->getDelete($this->csrfTokenName);
            }
            if (!empty($userToken) && $cookies->contains($this->csrfTokenName))
            {
                $cookieToken=$cookies->itemAt($this->csrfTokenName)->value;
                $valid=$cookieToken===$userToken;
            }
            else
                $valid = false;
            if (!$valid)
                throw new CHttpException(400,Yii::t('yii','The CSRF token could not be
verified.'));
        }
        }
}
class CCookieCollection extends CMap
{
        private $_request;
        private $_initialized=false;
        public function __construct(CHttpRequest $request)
        {
                $this->_request=$request;
                $this->copyfrom($this->getCookies());
                $this->_initialized=true;
        }
        public function getRequest()
        {
                return $this->_request;
        }
        protected function getCookies()
        {
                $cookies=array();
                if($this->_request->enableCookieValidation)
                {
                        $sm=Yii::app()->getSecurityManager();
                        foreach($_COOKIE as $name=>$value)
                        {
                                if(is_string($value) && ($value=$sm->validateData($value))!==false)
                                        $cookies[$name]=new CHttpCookie($name,@unserialize($value));
                        }
                }
                else
                {
                        foreach($_COOKIE as $name=>$value)
                                $cookies[$name]=new CHttpCookie($name,$value);
                }
                return $cookies;
        }
        public function add($name,$cookie)
        {
                if($cookie instanceof CHttpCookie)
                {
                        $this->remove($name);
                        parent::add($name,$cookie);
                        if($this->_initialized)
                                $this->addCookie($cookie);
                }
```

```
		else
			throw new CException(Yii::t('yii','CHttpCookieCollection can only hold CHttpCookie
objects.'));
	}
	public function remove($name,$options=array())
	{
		if(($cookie=parent::remove($name))!==null)
		{
			if($this->_initialized)
			{
				$cookie->configure($options);
				$this->removeCookie($cookie);
			}
		}
		return $cookie;
	}
	protected function addCookie($cookie)
	{
		$value=$cookie->value;
		if($this->_request->enableCookieValidation)
			$value=Yii::app()->getSecurityManager()->hashData(serialize($value));
		if(version_compare(PHP_VERSION,'5.2.0','>='))

	setcookie($cookie->name,$value,$cookie->expire,$cookie->path,$cookie->domain,$cookie-
>secure,$cookie->httpOnly);
		else

	setcookie($cookie->name,$value,$cookie->expire,$cookie->path,$cookie->domain,$cookie-
>secure);
	}
	protected function removeCookie($cookie)
	{
		if(version_compare(PHP_VERSION,'5.2.0','>='))

	setcookie($cookie->name,'',0,$cookie->path,$cookie->domain,$cookie->secure,$cookie->htt
pOnly);
		else
			setcookie($cookie->name,'',0,$cookie->path,$cookie->domain,$cookie->secure);
	}
}
class CUrlManager extends CApplicationComponent
{
	const CACHE_KEY='Yii.CUrlManager.rules';
	const GET_FORMAT='get';
```

```
    const PATH_FORMAT='path';
    public $rules=array();
    public $urlSuffix='';
    public $showScriptName=true;
    public $appendParams=true;
    public $routeVar='r';
    public $caseSensitive=true;
    public $matchValue=false;
    public $cacheID='cache';
    public $useStrictParsing=false;
    public $urlRuleClass='CUrlRule';
    private $_urlFormat=self::GET_FORMAT;
    private $_rules=array();
    private $_baseUrl;
    public function init()
    {
        parent::init();
        $this->processRules();
    }
    protected function processRules()
    {
        if(empty($this->rules) || $this->getUrlFormat()===self::GET_FORMAT)
            return;
        if($this->cacheID!==false                                              &&
($cache=Yii::app()->getComponent($this->cacheID))!==null)
        {
            $hash=md5(serialize($this->rules));
            if(($data=$cache->get(self::CACHE_KEY))!==false    &&    isset($data[1])    &&
$data[1]===$hash)
            {
                $this->_rules=$data[0];
                return;
            }
        }
        foreach($this->rules as $pattern=>$route)
            $this->_rules[]=$this->createUrlRule($route,$pattern);
        if(isset($cache))
            $cache->set(self::CACHE_KEY,array($this->_rules,$hash));
    }
    public function addRules($rules,$append=true)
    {
        if ($append)
        {
            foreach($rules as $pattern=>$route)
```

```
                    $this->_rules[]=$this->createUrlRule($route,$pattern);
            }
            else
            {
                $rules=array_reverse($rules);
                foreach($rules as $pattern=>$route)
                    array_unshift($this->_rules, $this->createUrlRule($route,$pattern));
            }
        }
        protected function createUrlRule($route,$pattern)
        {
            if(is_array($route) && isset($route['class']))
                return $route;
            else
                return new $this->urlRuleClass($route,$pattern);
        }
        public function createUrl($route,$params=array(),$ampersand='&')
        {
            unset($params[$this->routeVar]);
            foreach($params as $i=>$param)
                if($param===null)
                    $params[$i]='';
            if(isset($params['#']))
            {
                $anchor='#'.$params['#'];
                unset($params['#']);
            }
            else
                $anchor='';
            $route=trim($route,'/');
            foreach($this->_rules as $i=>$rule)
            {
                if(is_array($rule))
                    $this->_rules[$i]=$rule=Yii::createComponent($rule);
                if(($url=$rule->createUrl($this,$route,$params,$ampersand))!==false)
                {
                    if($rule->hasHostInfo)
                        return $url==='' ? '/'.$anchor : $url.$anchor;
                    else
                        return $this->getBaseUrl().'/'.$url.$anchor;
                }
            }
            return $this->createUrlDefault($route,$params,$ampersand).$anchor;
        }
```

```
protected function createUrlDefault($route,$params,$ampersand)
{
        if($this->getUrlFormat()===self::PATH_FORMAT)
        {
                $url=rtrim($this->getBaseUrl().'/'.$route,'/');
                if($this->appendParams)
                {
                        $url=rtrim($url.'/'.$this->createPathInfo($params,'/','/'),'/');
                        return $route==='' ? $url : $url.$this->urlSuffix;
                }
                else
                {
                        if($route!=='')
                                $url.=$this->urlSuffix;
                        $query=$this->createPathInfo($params,'=',$ampersand);
                        return $query==='' ? $url : $url.'?'.$query;
                }
        }
        else
        {
                $url=$this->getBaseUrl();
                if(!$this->showScriptName)
                        $url.='/';
                if($route!=='')
                {
                        $url.='?'.$this->routeVar.'='.$route;
                        if(($query=$this->createPathInfo($params,'=',$ampersand))!=='')
                                $url.=$ampersand.$query;
                }
                elseif(($query=$this->createPathInfo($params,'=',$ampersand))!=='')
                        $url.='?'.$query;
                return $url;
        }
}
public function parseUrl($request)
{
        if($this->getUrlFormat()===self::PATH_FORMAT)
        {
                $rawPathInfo=$request->getPathInfo();
                $pathInfo=$this->removeUrlSuffix($rawPathInfo,$this->urlSuffix);
                foreach($this->_rules as $i=>$rule)
                {
                        if(is_array($rule))
                                $this->_rules[$i]=$rule=Yii::createComponent($rule);
```

```
				if(($r=$rule->parseUrl($this,$request,$pathInfo,$rawPathInfo))!==false)
					return isset($_GET[$this->routeVar]) ? $_GET[$this->routeVar] : $r;
			}
			if($this->useStrictParsing)
				throw new CHttpException(404,Yii::t('yii','Unable to resolve the request
"{route}".',
					array('{route}'=>$pathInfo)));
			else
				return $pathInfo;
		}
		elseif(isset($_GET[$this->routeVar]))
			return $_GET[$this->routeVar];
		elseif(isset($_POST[$this->routeVar]))
			return $_POST[$this->routeVar];
		else
			return '';
	}
	public function parsePathInfo($pathInfo)
	{
		if($pathInfo==='')
			return;
		$segs=explode('/',$pathInfo.'/');
		$n=count($segs);
		for($i=0;$i<$n-1;$i+=2)
		{
			$key=$segs[$i];
			if($key==='') continue;
			$value=$segs[$i+1];
			if(($pos=strpos($key,'['))!==false                                &&
($m=preg_match_all('/\[(.*?)\]/',$key,$matches))>0)
			{
				$name=substr($key,0,$pos);
				for($j=$m-1;$j>=0;--$j)
				{
					if($matches[1][$j]==='')
						$value=array($value);
					else
						$value=array($matches[1][$j]=>$value);
				}
				if(isset($_GET[$name]) && is_array($_GET[$name]))
					$value=CMap::mergeArray($_GET[$name],$value);
				$_REQUEST[$name]=$_GET[$name]=$value;
			}
			else
```

```
            $_REQUEST[$key]=$_GET[$key]=$value;
        }
}
public function createPathInfo($params,$equal,$ampersand, $key=null)
{
        $pairs = array();
        foreach($params as $k => $v)
        {
                if ($key!==null)
                        $k = $key.'['.$k.']';
                if (is_array($v))
                        $pairs[]=$this->createPathInfo($v,$equal,$ampersand, $k);
                else
                        $pairs[]=urlencode($k).$equal.urlencode($v);
        }
        return implode($ampersand,$pairs);
}
public function removeUrlSuffix($pathInfo,$urlSuffix)
{
        if($urlSuffix!=='' && substr($pathInfo,-strlen($urlSuffix))===$urlSuffix)
                return substr($pathInfo,0,-strlen($urlSuffix));
        else
                return $pathInfo;
}
public function getBaseUrl()
{
        if($this->_baseUrl!==null)
                return $this->_baseUrl;
        else
        {
                if($this->showScriptName)
                        $this->_baseUrl=Yii::app()->getRequest()->getScriptUrl();
                else
                        $this->_baseUrl=Yii::app()->getRequest()->getBaseUrl();
                return $this->_baseUrl;
        }
}
public function setBaseUrl($value)
{
        $this->_baseUrl=$value;
}
public function getUrlFormat()
{
        return $this->_urlFormat;
```

```
	}
	public function setUrlFormat($value)
	{
		if($value===self::PATH_FORMAT || $value===self::GET_FORMAT)
			$this->_urlFormat=$value;
		else
			throw new CException(Yii::t('yii','CUrlManager.UrlFormat must be either "path" or
"get".'));
	}
}
abstract class CBaseUrlRule extends CComponent
{
	public $hasHostInfo=false;
	abstract public function createUrl($manager,$route,$params,$ampersand);
	abstract public function parseUrl($manager,$request,$pathInfo,$rawPathInfo);
}
class CUrlRule extends CBaseUrlRule
{
	public $urlSuffix;
	public $caseSensitive;
	public $defaultParams=array();
	public $matchValue;
	public $verb;
	public $parsingOnly=false;
	public $route;
	public $references=array();
	public $routePattern;
	public $pattern;
	public $template;
	public $params=array();
	public $append;
	public $hasHostInfo;
	public function __construct($route,$pattern)
	{
		if(is_array($route))
		{
			foreach(array('urlSuffix', 'caseSensitive', 'defaultParams', 'matchValue', 'verb',
'parsingOnly') as $name)
			{
				if(isset($route[$name]))
					$this->$name=$route[$name];
			}
			if(isset($route['pattern']))
				$pattern=$route['pattern'];
```

```
                $route=$route[0];
            }
            $this->route=trim($route,'/');
            $tr2['/']=$tr['/']='\\/';
            if(strpos($route,'<')!==false && preg_match_all('/<(\w+)>/',$route,$matches2))
            {
                foreach($matches2[1] as $name)
                    $this->references[$name]="<$name>";
            }
            $this->hasHostInfo=!strncasecmp($pattern,'http://',7)
|| !strncasecmp($pattern,'https://',8);
            if($this->verb!==null)

        $this->verb=preg_split('/[\s,]+/',strtoupper($this->verb),-1,PREG_SPLIT_NO_EMPTY);
            if(preg_match_all('/<(\w+):?(.*?)?>/',$pattern,$matches))
            {
                $tokens=array_combine($matches[1],$matches[2]);
                foreach($tokens as $name=>$value)
                {
                    if($value==='')
                        $value='[^\/]+';
                    $tr["<$name>"]="(?P<$name>$value)";
                    if(isset($this->references[$name]))
                        $tr2["<$name>"]=$tr["<$name>"];
                    else
                        $this->params[$name]=$value;
                }
            }
            $p=rtrim($pattern,'*');
            $this->append=$p!==$pattern;
            $p=trim($p,'/');
            $this->template=preg_replace('/<(\w+):?.*?>/','<$1>',$p);
            $this->pattern='/^'.strtr($this->template,$tr).'\/';
            if($this->append)
                $this->pattern.='/u';
            else
                $this->pattern.='$/u';
            if($this->references!==array())
                $this->routePattern='/^'.strtr($this->route,$tr2).'$/u';
            if(YII_DEBUG && @preg_match($this->pattern,'test')===false)
                throw new CException(Yii::t('yii','The URL pattern "{pattern}" for route "{route}" is
not a valid regular expression.',
                    array('{route}'=>$route,'{pattern}'=>$pattern)));
        }
```

```
    public function createUrl($manager,$route,$params,$ampersand)
    {
        if($this->parsingOnly)
            return false;
        if($manager->caseSensitive && $this->caseSensitive===null || $this->caseSensitive)
            $case='';
        else
            $case='i';
        $tr=array();
        if($route!==$this->route)
        {
            if($this->routePattern!==null                                          &&
preg_match($this->routePattern.$case,$route,$matches))
            {
                foreach($this->references as $key=>$name)
                    $tr[$name]=$matches[$key];
            }
            else
                return false;
        }
        foreach($this->defaultParams as $key=>$value)
        {
            if(isset($params[$key]))
            {
                if($params[$key]==$value)
                    unset($params[$key]);
                else
                    return false;
            }
        }
        foreach($this->params as $key=>$value)
            if(!isset($params[$key]))
                return false;
        if($manager->matchValue && $this->matchValue===null || $this->matchValue)
        {
            foreach($this->params as $key=>$value)
            {
                if(!preg_match('/\A'.$value.'\z/u'.$case,$params[$key]))
                    return false;
            }
        }
        foreach($this->params as $key=>$value)
        {
            $tr["<$key>"]=urlencode($params[$key]);
				unset($params[$key]);
			}
			$suffix=$this->urlSuffix===null ? $manager->urlSuffix : $this->urlSuffix;
			$url=strtr($this->template,$tr);
			if($this->hasHostInfo)
			{
				$hostInfo=Yii::app()->getRequest()->getHostInfo();
				if(stripos($url,$hostInfo)===0)
					$url=substr($url,strlen($hostInfo));
			}
			if(empty($params))
				return $url!=='' ? $url.$suffix : $url;
			if($this->append)
				$url.='/'.$manager->createPathInfo($params,'/','/').$suffix;
			else
			{
				if($url!=='')
					$url.=$suffix;
				$url.='?'.$manager->createPathInfo($params,'=',$ampersand);
			}
			return $url;
		}
		public function parseUrl($manager,$request,$pathInfo,$rawPathInfo)
		{
			if($this->verb!==null && !in_array($request->getRequestType(), $this->verb, true))
				return false;
			if($manager->caseSensitive && $this->caseSensitive===null || $this->caseSensitive)
				$case='';
			else
				$case='i';
			if($this->urlSuffix!==null)
				$pathInfo=$manager->removeUrlSuffix($rawPathInfo,$this->urlSuffix);
			// URL suffix required, but not found in the requested URL
			if($manager->useStrictParsing && $pathInfo===$rawPathInfo)
			{
				$urlSuffix=$this->urlSuffix===null ? $manager->urlSuffix : $this->urlSuffix;
				if($urlSuffix!='' && $urlSuffix!=='/')
					return false;
			}
			if($this->hasHostInfo)
				$pathInfo=strtolower($request->getHostInfo()).rtrim('/'.$pathInfo,'/');
			$pathInfo.='/';
			if(preg_match($this->pattern.$case,$pathInfo,$matches))
			{
```

```
                foreach($this->defaultParams as $name=>$value)
                {
                    if(!isset($_GET[$name]))
                        $_REQUEST[$name]=$_GET[$name]=$value;
                }
                $tr=array();
                foreach($matches as $key=>$value)
                {
                    if(isset($this->references[$key]))
                        $tr[$this->references[$key]]=$value;
                    elseif(isset($this->params[$key]))
                        $_REQUEST[$key]=$_GET[$key]=$value;
                }
                if($pathInfo!==$matches[0]) // there're additional GET params
                    $manager->parsePathInfo(ltrim(substr($pathInfo,strlen($matches[0])),'/'));
                if($this->routePattern!==null)
                    return strtr($this->route,$tr);
                else
                    return $this->route;
            }
            else
                return false;
        }
}
abstract class CBaseController extends CComponent
{
        private $_widgetStack=array();
        abstract public function getViewFile($viewName);
        public function renderFile($viewFile,$data=null,$return=false)
        {
            $widgetCount=count($this->_widgetStack);
            if(($renderer=Yii::app()->getViewRenderer())!==null                                  &&
$renderer->fileExtension==='.'.CFileHelper::getExtension($viewFile))
                $content=$renderer->renderFile($this,$viewFile,$data,$return);
            else
                $content=$this->renderInternal($viewFile,$data,$return);
            if(count($this->_widgetStack)===$widgetCount)
                return $content;
            else
            {
                $widget=end($this->_widgetStack);
                throw  new  CException(Yii::t('yii','{controller}  contains  improperly  nested  widget
tags in its view "{view}". A {widget} widget does not have an endWidget() call.',
                    array('{controller}'=>get_class($this),                              '{view}'=>$viewFile,
'{widget}'=>get_class($widget))));
            }
        }
        public function renderInternal($_viewFile_,$_data_=null,$_return_=false)
        {
            // we use special variable names here to avoid conflict when extracting data
            if(is_array($_data_))
                extract($_data_,EXTR_PREFIX_SAME,'data');
            else
                $data=$_data_;
            if($_return_)
            {
                ob_start();
                ob_implicit_flush(false);
                require($_viewFile_);
                return ob_get_clean();
            }
            else
                require($_viewFile_);
        }
        public function createWidget($className,$properties=array())
        {
            $widget=Yii::app()->getWidgetFactory()->createWidget($this,$className,$properties);
            $widget->init();
            return $widget;
        }
        public function widget($className,$properties=array(),$captureOutput=false)
        {
            if($captureOutput)
            {
                ob_start();
                ob_implicit_flush(false);
                $widget=$this->createWidget($className,$properties);
                $widget->run();
                return ob_get_clean();
            }
            else
            {
                $widget=$this->createWidget($className,$properties);
                $widget->run();
                return $widget;
            }
        }
        public function beginWidget($className,$properties=array())
```

```
	{
		$widget=$this->createWidget($className,$properties);
		$this->_widgetStack[]=$widget;
		return $widget;
	}
	public function endWidget($id='')
	{
		if(($widget=array_pop($this->_widgetStack))!==null)
		{
			$widget->run();
			return $widget;
		}
		else
			throw new CException(Yii::t('yii','{controller} has an extra endWidget({id}) call in its
view.',
				array('{controller}'=>get_class($this),'{id}'=>$id)));
	}
	public function beginClip($id,$properties=array())
	{
		$properties['id']=$id;
		$this->beginWidget('CClipWidget',$properties);
	}
	public function endClip()
	{
		$this->endWidget('CClipWidget');
	}
	public function beginCache($id,$properties=array())
	{
		$properties['id']=$id;
		$cache=$this->beginWidget('COutputCache',$properties);
		if($cache->getIsContentCached())
		{
			$this->endCache();
			return false;
		}
		else
			return true;
	}
	public function endCache()
	{
		$this->endWidget('COutputCache');
	}
	public function beginContent($view=null,$data=array())
	{
		$this->beginWidget('CContentDecorator',array('view'=>$view, 'data'=>$data));
	}
	public function endContent()
	{
		$this->endWidget('CContentDecorator');
	}
}
class CController extends CBaseController
{
	const STATE_INPUT_NAME='YII_PAGE_STATE';
	public $layout;
	public $defaultAction='index';
	private $_id;
	private $_action;
	private $_pageTitle;
	private $_cachingStack;
	private $_clips;
	private $_dynamicOutput;
	private $_pageStates;
	private $_module;
	public function __construct($id,$module=null)
	{
		$this->_id=$id;
		$this->_module=$module;
		$this->attachBehaviors($this->behaviors());
	}
	public function init()
	{
	}
	public function filters()
	{
		return array();
	}
	public function actions()
	{
		return array();
	}
	public function behaviors()
	{
		return array();
	}
	public function accessRules()
	{
		return array();
```

```
}
public function run($actionID)
{
        if(($action=$this->createAction($actionID))!==null)
        {
                if(($parent=$this->getModule())===null)
                        $parent=Yii::app();
                if($parent->beforeControllerAction($this,$action))
                {
                        $this->runActionWithFilters($action,$this->filters());
                        $parent->afterControllerAction($this,$action);
                }
        }
        else
                $this->missingAction($actionID);
}
public function runActionWithFilters($action,$filters)
{
        if(empty($filters))
                $this->runAction($action);
        else
        {
                $priorAction=$this->_action;
                $this->_action=$action;
                CFilterChain::create($this,$action,$filters)->run();
                $this->_action=$priorAction;
        }
}
public function runAction($action)
{
        $priorAction=$this->_action;
        $this->_action=$action;
        if($this->beforeAction($action))
        {
                if($action->runWithParams($this->getActionParams())===false)
                        $this->invalidActionParams($action);
                else
                        $this->afterAction($action);
        }
        $this->_action=$priorAction;
}
public function getActionParams()
{
        return $_GET;
```

```
	}
	public function invalidActionParams($action)
	{
		throw new CHttpException(400,Yii::t('yii','Your request is invalid.'));
	}
	public function processOutput($output)
	{
		Yii::app()->getClientScript()->render($output);
		// if using page caching, we should delay dynamic output replacement
		if($this->_dynamicOutput!==null && $this->isCachingStackEmpty())
		{
			$output=$this->processDynamicOutput($output);
			$this->_dynamicOutput=null;
		}
		if($this->_pageStates===null)
			$this->_pageStates=$this->loadPageStates();
		if(!empty($this->_pageStates))
			$this->savePageStates($this->_pageStates,$output);
		return $output;
	}
	public function processDynamicOutput($output)
	{
		if($this->_dynamicOutput)
		{

	$output=preg_replace_callback('/<###dynamic-(\d+)###>/',array($this,'replaceDynamicOut
put'),$output);
		}
		return $output;
	}
	protected function replaceDynamicOutput($matches)
	{
		$content=$matches[0];
		if(isset($this->_dynamicOutput[$matches[1]]))
		{
			$content=$this->_dynamicOutput[$matches[1]];
			$this->_dynamicOutput[$matches[1]]=null;
		}
		return $content;
	}
	public function createAction($actionID)
	{
		if($actionID==='')
			$actionID=$this->defaultAction;
		if(method_exists($this,'action'.$actionID) && strcasecmp($actionID,'s')) // we have
actions method
			return new CInlineAction($this,$actionID);
		else
		{
			$action=$this->createActionFromMap($this->actions(),$actionID,$actionID);
			if($action!==null && !method_exists($action,'run'))
				throw new CException(Yii::t('yii', 'Action class {class} must implement the
"run" method.', array('{class}'=>get_class($action))));
			return $action;
		}
	}
	protected                                                              function
createActionFromMap($actionMap,$actionID,$requestActionID,$config=array())
	{
		if(($pos=strpos($actionID,'.'))===false && isset($actionMap[$actionID]))
		{
			$baseConfig=is_array($actionMap[$actionID])   ?   $actionMap[$actionID]   :
array('class'=>$actionMap[$actionID]);
			return
Yii::createComponent(empty($config)?$baseConfig:array_merge($baseConfig,$config),$this,$req
uestActionID);
		}
		elseif($pos===false)
			return null;
		// the action is defined in a provider
		$prefix=substr($actionID,0,$pos+1);
		if(!isset($actionMap[$prefix]))
			return null;
		$actionID=(string)substr($actionID,$pos+1);
		$provider=$actionMap[$prefix];
		if(is_string($provider))
			$providerType=$provider;
		elseif(is_array($provider) && isset($provider['class']))
		{
			$providerType=$provider['class'];
			if(isset($provider[$actionID]))
			{
				if(is_string($provider[$actionID]))
					$config=array_merge(array('class'=>$provider[$actionID]),$config);
				else
					$config=array_merge($provider[$actionID],$config);
			}
		}
		else
			throw new CException(Yii::t('yii','Object configuration must be an array containing
a "class" element.'));
		$class=Yii::import($providerType,true);
		$map=call_user_func(array($class,'actions'));
		return $this->createActionFromMap($map,$actionID,$requestActionID,$config);
	}
	public function missingAction($actionID)
	{
		throw new CHttpException(404,Yii::t('yii','The system is unable to find the requested
action "{action}".',
			array('{action}'=>$actionID==''?$this->defaultAction:$actionID)));
	}
	public function getAction()
	{
		return $this->_action;
	}
	public function setAction($value)
	{
		$this->_action=$value;
	}
	public function getId()
	{
		return $this->_id;
	}
	public function getUniqueId()
	{
		return $this->_module ? $this->_module->getId().'/'.$this->_id : $this->_id;
	}
	public function getRoute()
	{
		if(($action=$this->getAction())!==null)
			return $this->getUniqueId().'/'.$action->getId();
		else
			return $this->getUniqueId();
	}
	public function getModule()
	{
		return $this->_module;
	}
	public function getViewPath()
	{
		if(($module=$this->getModule())===null)
			$module=Yii::app();
```

```
        return $module->getViewPath().DIRECTORY_SEPARATOR.$this->getId();
    }
    public function getViewFile($viewName)
    {
        if(($theme=Yii::app()->getTheme())!==null                                  &&
($viewFile=$theme->getViewFile($this,$viewName))!==false)
            return $viewFile;
        $moduleViewPath=$basePath=Yii::app()->getViewPath();
        if(($module=$this->getModule())!==null)
            $moduleViewPath=$module->getViewPath();
        return
$this->resolveViewFile($viewName,$this->getViewPath(),$basePath,$moduleViewPath);
    }
    public function getLayoutFile($layoutName)
    {
        if($layoutName===false)
            return false;
        if(($theme=Yii::app()->getTheme())!==null                                  &&
($layoutFile=$theme->getLayoutFile($this,$layoutName))!==false)
            return $layoutFile;
        if(empty($layoutName))
        {
            $module=$this->getModule();
            while($module!==null)
            {
                if($module->layout===false)
                    return false;
                if(!empty($module->layout))
                    break;
                $module=$module->getParentModule();
            }
            if($module===null)
                $module=Yii::app();
            $layoutName=$module->layout;
        }
        elseif(($module=$this->getModule())===null)
            $module=Yii::app();
        return
$this->resolveViewFile($layoutName,$module->getLayoutPath(),Yii::app()->getViewPath(),$modu
le->getViewPath());
    }
    public function resolveViewFile($viewName,$viewPath,$basePath,$moduleViewPath=null)
    {
        if(empty($viewName))
```

```
            return false;
        if($moduleViewPath===null)
            $moduleViewPath=$basePath;
        if(($renderer=Yii::app()->getViewRenderer())!==null)
            $extension=$renderer->fileExtension;
        else
            $extension='.php';
        if($viewName[0]==='/')
        {
            if(strncmp($viewName,'//',2)===0)
                $viewFile=$basePath.$viewName;
            else
                $viewFile=$moduleViewPath.$viewName;
        }
        elseif(strpos($viewName,'.'))
            $viewFile=Yii::getPathOfAlias($viewName);
        else
            $viewFile=$viewPath.DIRECTORY_SEPARATOR.$viewName;
        if(is_file($viewFile.$extension))
            return Yii::app()->findLocalizedFile($viewFile.$extension);
        elseif($extension!=='.php' && is_file($viewFile.'.php'))
            return Yii::app()->findLocalizedFile($viewFile.'.php');
        else
            return false;
    }
    public function getClips()
    {
        if($this->_clips!==null)
            return $this->_clips;
        else
            return $this->_clips=new CMap;
    }
    public function forward($route,$exit=true)
    {
        if(strpos($route,'/')===false)
            $this->run($route);
        else
        {
            if($route[0]!=='/' && ($module=$this->getModule())!==null)
                $route=$module->getId().'/'.$route;
            Yii::app()->runController($route);
        }
        if($exit)
            Yii::app()->end();
```

```
}
public function render($view,$data=null,$return=false)
{
        if($this->beforeRender($view))
        {
                $output=$this->renderPartial($view,$data,true);
                if(($layoutFile=$this->getLayoutFile($this->layout))!==false)
                        $output=$this->renderFile($layoutFile,array('content'=>$output),true);
                $this->afterRender($view,$output);
                $output=$this->processOutput($output);
                if($return)
                        return $output;
                else
                        echo $output;
        }
}
protected function beforeRender($view)
{
        return true;
}
protected function afterRender($view, &$output)
{
}
public function renderText($text,$return=false)
{
        if(($layoutFile=$this->getLayoutFile($this->layout))!==false)
                $text=$this->renderFile($layoutFile,array('content'=>$text),true);
        $text=$this->processOutput($text);
        if($return)
                return $text;
        else
                echo $text;
}
public function renderPartial($view,$data=null,$return=false,$processOutput=false)
{
        if(($viewFile=$this->getViewFile($view))!==false)
        {
                $output=$this->renderFile($viewFile,$data,true);
                if($processOutput)
                        $output=$this->processOutput($output);
                if($return)
                        return $output;
                else
                        echo $output;
```

```
		}
		else
			throw new CException(Yii::t('yii','{controller} cannot find the requested view "{view}".',
				array('{controller}'=>get_class($this), '{view}'=>$view)));
	}
	public function renderClip($name,$params=array(),$return=false)
	{
		$text=isset($this->clips[$name]) ? strtr($this->clips[$name], $params) : '';
		if($return)
			return $text;
		else
			echo $text;
	}
	public function renderDynamic($callback)
	{
		$n=count($this->_dynamicOutput);
		echo "<###dynamic-$n###>";
		$params=func_get_args();
		array_shift($params);
		$this->renderDynamicInternal($callback,$params);
	}
	public function renderDynamicInternal($callback,$params)
	{
		$this->recordCachingAction('','renderDynamicInternal',array($callback,$params));
		if(is_string($callback) && method_exists($this,$callback))
			$callback=array($this,$callback);
		$this->_dynamicOutput[]=call_user_func_array($callback,$params);
	}
	public function createUrl($route,$params=array(),$ampersand='&')
	{
		if($route==='')
			$route=$this->getId().'/'.$this->getAction()->getId();
		elseif(strpos($route,'/')===false)
			$route=$this->getId().'/'.$route;
		if($route[0]!=='/' && ($module=$this->getModule())!==null)
			$route=$module->getId().'/'.$route;
		return Yii::app()->createUrl(trim($route,'/'),$params,$ampersand);
	}
	public function createAbsoluteUrl($route,$params=array(),$schema='',$ampersand='&')
	{
		$url=$this->createUrl($route,$params,$ampersand);
		if(strpos($url,'http')===0)
			return $url;
		else
			return Yii::app()->getRequest()->getHostInfo($schema).$url;
	}
	public function getPageTitle()
	{
		if($this->_pageTitle!==null)
			return $this->_pageTitle;
		else
		{
			$name=ucfirst(basename($this->getId()));
			if($this->getAction()!==null                                                    &&
strcasecmp($this->getAction()->getId(),$this->defaultAction))
				return                  $this->_pageTitle=Yii::app()->name.'                  -
'.ucfirst($this->getAction()->getId()).' '.$name;
			else
				return $this->_pageTitle=Yii::app()->name.' - '.$name;
		}
	}
	public function setPageTitle($value)
	{
		$this->_pageTitle=$value;
	}
	public function redirect($url,$terminate=true,$statusCode=302)
	{
		if(is_array($url))
		{
			$route=isset($url[0]) ? $url[0] : '';
			$url=$this->createUrl($route,array_splice($url,1));
		}
		Yii::app()->getRequest()->redirect($url,$terminate,$statusCode);
	}
	public function refresh($terminate=true,$anchor='')
	{
		$this->redirect(Yii::app()->getRequest()->getUrl().$anchor,$terminate);
	}
	public function recordCachingAction($context,$method,$params)
	{
		if($this->_cachingStack) // record only when there is an active output cache
		{
			foreach($this->_cachingStack as $cache)
				$cache->recordAction($context,$method,$params);
		}
	}
	public function getCachingStack($createIfNull=true)
```

```
{
        if(!$this->_cachingStack)
                $this->_cachingStack=new CStack;
        return $this->_cachingStack;
}
public function isCachingStackEmpty()
{
        return $this->_cachingStack===null || !$this->_cachingStack->getCount();
}
protected function beforeAction($action)
{
        return true;
}
protected function afterAction($action)
{
}
public function filterPostOnly($filterChain)
{
        if(Yii::app()->getRequest()->getIsPostRequest())
                $filterChain->run();
        else
                throw new CHttpException(400,Yii::t('yii','Your request is invalid.'));
}
public function filterAjaxOnly($filterChain)
{
        if(Yii::app()->getRequest()->getIsAjaxRequest())
                $filterChain->run();
        else
                throw new CHttpException(400,Yii::t('yii','Your request is invalid.'));
}
public function filterAccessControl($filterChain)
{
        $filter=new CAccessControlFilter;
        $filter->setRules($this->accessRules());
        $filter->filter($filterChain);
}
public function getPageState($name,$defaultValue=null)
{
        if($this->_pageStates===null)
                $this->_pageStates=$this->loadPageStates();
        return isset($this->_pageStates[$name])?$this->_pageStates[$name]:$defaultValue;
}
public function setPageState($name,$value,$defaultValue=null)
{
		if($this->_pageStates===null)
			$this->_pageStates=$this->loadPageStates();
		if($value===$defaultValue)
			unset($this->_pageStates[$name]);
		else
			$this->_pageStates[$name]=$value;
		$params=func_get_args();
		$this->recordCachingAction('','setPageState',$params);
	}
	public function clearPageStates()
	{
		$this->_pageStates=array();
	}
	protected function loadPageStates()
	{
		if(!empty($_POST[self::STATE_INPUT_NAME]))
		{
			if(($data=base64_decode($_POST[self::STATE_INPUT_NAME]))!==false)
			{
				if(extension_loaded('zlib'))
					$data=@gzuncompress($data);
				if(($data=Yii::app()->getSecurityManager()->validateData($data))!==false)
					return unserialize($data);
			}
		}
		return array();
	}
	protected function savePageStates($states,&$output)
	{
		$data=Yii::app()->getSecurityManager()->hashData(serialize($states));
		if(extension_loaded('zlib'))
			$data=gzcompress($data);
		$value=base64_encode($data);
		$output=str_replace(CHtml::pageStateField(''),CHtml::pageStateField($value),$output);
	}
}
abstract class CAction extends CComponent implements IAction
{
	private $_id;
	private $_controller;
	public function __construct($controller,$id)
	{
		$this->_controller=$controller;
		$this->_id=$id;
	}
	public function getController()
	{
		return $this->_controller;
	}
	public function getId()
	{
		return $this->_id;
	}
	public function runWithParams($params)
	{
		$method=new ReflectionMethod($this, 'run');
		if($method->getNumberOfParameters()>0)
			return $this->runWithParamsInternal($this, $method, $params);
		else
			return $this->run();
	}
	protected function runWithParamsInternal($object, $method, $params)
	{
		$ps=array();
		foreach($method->getParameters() as $i=>$param)
		{
			$name=$param->getName();
			if(isset($params[$name]))
			{
				if($param->isArray())
					$ps[]=is_array($params[$name])	?	$params[$name]	:
array($params[$name]);
				elseif(!is_array($params[$name]))
					$ps[]=$params[$name];
				else
					return false;
			}
			elseif($param->isDefaultValueAvailable())
				$ps[]=$param->getDefaultValue();
			else
				return false;
		}
		$method->invokeArgs($object,$ps);
		return true;
	}
}
class CInlineAction extends CAction
{
	public function run()
	{
		$method='action'.$this->getId();
		$this->getController()->$method();
	}
	public function runWithParams($params)
	{
		$methodName='action'.$this->getId();
		$controller=$this->getController();
		$method=new ReflectionMethod($controller, $methodName);
		if($method->getNumberOfParameters()>0)
			return $this->runWithParamsInternal($controller, $method, $params);
		else
			return $controller->$methodName();
	}
}
class CWebUser extends CApplicationComponent implements IWebUser
{
	const FLASH_KEY_PREFIX='Yii.CWebUser.flash.';
	const FLASH_COUNTERS='Yii.CWebUser.flashcounters';
	const STATES_VAR='__states';
	const AUTH_TIMEOUT_VAR='__timeout';
	public $allowAutoLogin=false;
	public $guestName='Guest';
	public $loginUrl=array('/site/login');
	public $identityCookie;
	public $authTimeout;
	public $autoRenewCookie=false;
	public $autoUpdateFlash=true;
	public $loginRequiredAjaxResponse;
	private $_keyPrefix;
	private $_access=array();
	public function __get($name)
	{
		if($this->hasState($name))
			return $this->getState($name);
		else
			return parent::__get($name);
	}
	public function __set($name,$value)
	{
		if($this->hasState($name))
			$this->setState($name,$value);
		else
```

```
                parent::__set($name,$value);
        }
        public function __isset($name)
        {
                if($this->hasState($name))
                        return $this->getState($name)!==null;
                else
                        return parent::__isset($name);
        }
        public function __unset($name)
        {
                if($this->hasState($name))
                        $this->setState($name,null);
                else
                        parent::__unset($name);
        }
        public function init()
        {
                parent::init();
                Yii::app()->getSession()->open();
                if($this->getIsGuest() && $this->allowAutoLogin)
                        $this->restoreFromCookie();
                elseif($this->autoRenewCookie && $this->allowAutoLogin)
                        $this->renewCookie();
                if($this->autoUpdateFlash)
                        $this->updateFlash();
                $this->updateAuthStatus();
        }
        public function login($identity,$duration=0)
        {
                $id=$identity->getId();
                $states=$identity->getPersistentStates();
                if($this->beforeLogin($id,$states,false))
                {
                        $this->changeIdentity($id,$identity->getName(),$states);
                        if($duration>0)
                        {
                                if($this->allowAutoLogin)
                                        $this->saveToCookie($duration);
                                else
                                        throw new CException(Yii::t('yii','{class}.allowAutoLogin must be set true
in order to use cookie-based authentication.',
                                                array('{class}'=>get_class($this))));
                        }
```

```
            $this->afterLogin(false);
        }
        return !$this->getIsGuest();
    }
    public function logout($destroySession=true)
    {
        if($this->beforeLogout())
        {
            if($this->allowAutoLogin)
            {
                Yii::app()->getRequest()->getCookies()->remove($this->getStateKeyPrefix());
                if($this->identityCookie!==null)
                {
                    $cookie=$this->createIdentityCookie($this->getStateKeyPrefix());
                    $cookie->value=null;
                    $cookie->expire=0;
                    Yii::app()->getRequest()->getCookies()->add($cookie->name,$cookie);
                }
            }
            if($destroySession)
                Yii::app()->getSession()->destroy();
            else
                $this->clearStates();
            $this->_access=array();
            $this->afterLogout();
        }
    }
    public function getIsGuest()
    {
        return $this->getState('__id')===null;
    }
    public function getId()
    {
        return $this->getState('__id');
    }
    public function setId($value)
    {
        $this->setState('__id',$value);
    }
    public function getName()
    {
        if(($name=$this->getState('__name'))!==null)
            return $name;
        else
```

```
                return $this->guestName;
        }
        public function setName($value)
        {
                $this->setState('__name',$value);
        }
        public function getReturnUrl($defaultUrl=null)
        {
                if($defaultUrl===null)
                {
                        $defaultReturnUrl=Yii::app()->getUrlManager()->showScriptName                    ?
Yii::app()->getRequest()->getScriptUrl() : Yii::app()->getRequest()->getBaseUrl().'/';
                }
                else
                {
                        $defaultReturnUrl=CHtml::normalizeUrl($defaultUrl);
                }
                return $this->getState('__returnUrl',$defaultReturnUrl);
        }
        public function setReturnUrl($value)
        {
                $this->setState('__returnUrl',$value);
        }
        public function loginRequired()
        {
                $app=Yii::app();
                $request=$app->getRequest();
                if(!$request->getIsAjaxRequest())
                        $this->setReturnUrl($request->getUrl());
                elseif(isset($this->loginRequiredAjaxResponse))
                {
                        echo $this->loginRequiredAjaxResponse;
                        Yii::app()->end();
                }
                if(($url=$this->loginUrl)!==null)
                {
                        if(is_array($url))
                        {
                                $route=isset($url[0]) ? $url[0] : $app->defaultController;
                                $url=$app->createUrl($route,array_splice($url,1));
                        }
                        $request->redirect($url);
                }
                else
                throw new CHttpException(403,Yii::t('yii','Login Required'));
        }
        protected function beforeLogin($id,$states,$fromCookie)
        {
                return true;
        }
        protected function afterLogin($fromCookie)
        {
        }
        protected function beforeLogout()
        {
                return true;
        }
        protected function afterLogout()
        {
        }
        protected function restoreFromCookie()
        {
                $app=Yii::app();
                $request=$app->getRequest();
                $cookie=$request->getCookies()->itemAt($this->getStateKeyPrefix());
                if($cookie    &&    !empty($cookie->value)    &&    is_string($cookie->value)    &&
($data=$app->getSecurityManager()->validateData($cookie->value))!==false)
                {
                        $data=@unserialize($data);
                        if(is_array($data) && isset($data[0],$data[1],$data[2],$data[3]))
                        {
                                list($id,$name,$duration,$states)=$data;
                                if($this->beforeLogin($id,$states,true))
                                {
                                        $this->changeIdentity($id,$name,$states);
                                        if($this->autoRenewCookie)
                                        {
                                                $cookie->expire=time()+$duration;
                                                $request->getCookies()->add($cookie->name,$cookie);
                                        }
                                        $this->afterLogin(true);
                                }
                        }
                }
        }
        protected function renewCookie()
        {
                $request=Yii::app()->getRequest();
        $cookies=$request->getCookies();
        $cookie=$cookies->itemAt($this->getStateKeyPrefix());
        if($cookie             &&             !empty($cookie->value)             &&
($data=Yii::app()->getSecurityManager()->validateData($cookie->value))!==false)
        {
            $data=@unserialize($data);
            if(is_array($data) && isset($data[0],$data[1],$data[2],$data[3]))
            {
                $cookie->expire=time()+$data[2];
                $cookies->add($cookie->name,$cookie);
            }
        }
    }
    protected function saveToCookie($duration)
    {
        $app=Yii::app();
        $cookie=$this->createIdentityCookie($this->getStateKeyPrefix());
        $cookie->expire=time()+$duration;
        $data=array(
            $this->getId(),
            $this->getName(),
            $duration,
            $this->saveIdentityStates(),
        );
        $cookie->value=$app->getSecurityManager()->hashData(serialize($data));
        $app->getRequest()->getCookies()->add($cookie->name,$cookie);
    }
    protected function createIdentityCookie($name)
    {
        $cookie=new CHttpCookie($name,'');
        if(is_array($this->identityCookie))
        {
            foreach($this->identityCookie as $name=>$value)
                $cookie->$name=$value;
        }
        return $cookie;
    }
    public function getStateKeyPrefix()
    {
        if($this->_keyPrefix!==null)
            return $this->_keyPrefix;
        else
            return $this->_keyPrefix=md5('Yii.'.get_class($this).'.'.Yii::app()->getId());
    }
```

```
public function setStateKeyPrefix($value)
{
        $this->_keyPrefix=$value;
}
public function getState($key,$defaultValue=null)
{
        $key=$this->getStateKeyPrefix().$key;
        return isset($_SESSION[$key]) ? $_SESSION[$key] : $defaultValue;
}
public function setState($key,$value,$defaultValue=null)
{
        $key=$this->getStateKeyPrefix().$key;
        if($value===$defaultValue)
                unset($_SESSION[$key]);
        else
                $_SESSION[$key]=$value;
}
public function hasState($key)
{
        $key=$this->getStateKeyPrefix().$key;
        return isset($_SESSION[$key]);
}
public function clearStates()
{
        $keys=array_keys($_SESSION);
        $prefix=$this->getStateKeyPrefix();
        $n=strlen($prefix);
        foreach($keys as $key)
        {
                if(!strncmp($key,$prefix,$n))
                        unset($_SESSION[$key]);
        }
}
public function getFlashes($delete=true)
{
        $flashes=array();
        $prefix=$this->getStateKeyPrefix().self::FLASH_KEY_PREFIX;
        $keys=array_keys($_SESSION);
        $n=strlen($prefix);
        foreach($keys as $key)
        {
                if(!strncmp($key,$prefix,$n))
                {
                        $flashes[substr($key,$n)]=$_SESSION[$key];
```

```
                    if($delete)
                        unset($_SESSION[$key]);
                }
            }
            if($delete)
                $this->setState(self::FLASH_COUNTERS,array());
            return $flashes;
        }
        public function getFlash($key,$defaultValue=null,$delete=true)
        {
            $value=$this->getState(self::FLASH_KEY_PREFIX.$key,$defaultValue);
            if($delete)
                $this->setFlash($key,null);
            return $value;
        }
        public function setFlash($key,$value,$defaultValue=null)
        {
            $this->setState(self::FLASH_KEY_PREFIX.$key,$value,$defaultValue);
            $counters=$this->getState(self::FLASH_COUNTERS,array());
            if($value===$defaultValue)
                unset($counters[$key]);
            else
                $counters[$key]=0;
            $this->setState(self::FLASH_COUNTERS,$counters,array());
        }
        public function hasFlash($key)
        {
            return $this->getFlash($key, null, false)!==null;
        }
        protected function changeIdentity($id,$name,$states)
        {
            Yii::app()->getSession()->regenerateID(true);
            $this->setId($id);
            $this->setName($name);
            $this->loadIdentityStates($states);
        }
        protected function saveIdentityStates()
        {
            $states=array();
            foreach($this->getState(self::STATES_VAR,array()) as $name=>$dummy)
                $states[$name]=$this->getState($name);
            return $states;
        }
        protected function loadIdentityStates($states)
```

```
{
        $names=array();
        if(is_array($states))
        {
                foreach($states as $name=>$value)
                {
                        $this->setState($name,$value);
                        $names[$name]=true;
                }
        }
        $this->setState(self::STATES_VAR,$names);
}
protected function updateFlash()
{
        $counters=$this->getState(self::FLASH_COUNTERS);
        if(!is_array($counters))
                return;
        foreach($counters as $key=>$count)
        {
                if($count)
                {
                        unset($counters[$key]);
                        $this->setState(self::FLASH_KEY_PREFIX.$key,null);
                }
                else
                        $counters[$key]++;
        }
        $this->setState(self::FLASH_COUNTERS,$counters,array());
}
protected function updateAuthStatus()
{
        if($this->authTimeout!==null && !$this->getIsGuest())
        {
                $expires=$this->getState(self::AUTH_TIMEOUT_VAR);
                if ($expires!==null && $expires < time())
                        $this->logout(false);
                else
                        $this->setState(self::AUTH_TIMEOUT_VAR,time()+$this->authTimeout);
        }
}
public function checkAccess($operation,$params=array(),$allowCaching=true)
{
        if($allowCaching && $params===array() && isset($this->_access[$operation]))
                return $this->_access[$operation];
        $access=Yii::app()->getAuthManager()->checkAccess($operation,$this->getId(),$params);
                if($allowCaching && $params===array())
                        $this->_access[$operation]=$access;
                return $access;
        }
}
class          CHttpSession          extends          CApplicationComponent          implements
IteratorAggregate,ArrayAccess,Countable
{
        public $autoStart=true;
        public function init()
        {
                parent::init();
                // default session gc probability is 1%
                ini_set('session.gc_probability',1);
                ini_set('session.gc_divisor',100);
                if($this->autoStart)
                        $this->open();
                register_shutdown_function(array($this,'close'));
        }
        public function getUseCustomStorage()
        {
                return false;
        }
        public function open()
        {
                if($this->getUseCustomStorage())

        @session_set_save_handler(array($this,'openSession'),array($this,'closeSession'),array($this
,'readSession'),array($this,'writeSession'),array($this,'destroySession'),array($this,'gcSession'));
                @session_start();
                if(YII_DEBUG && session_id()=='')
                {
                        $message=Yii::t('yii','Failed to start session.');
                        if(function_exists('error_get_last'))
                        {
                                $error=error_get_last();
                                if(isset($error['message']))
                                        $message=$error['message'];
                        }
                        Yii::log($message, CLogger::LEVEL_WARNING, 'system.web.CHttpSession');
                }
        }
```

```
public function close()
{
	if(session_id()!=='')
		@session_write_close();
}
public function destroy()
{
	if(session_id()!=='')
	{
		@session_unset();
		@session_destroy();
	}
}
public function getIsStarted()
{
	return session_id()!=='';
}
public function getSessionID()
{
	return session_id();
}
public function setSessionID($value)
{
	session_id($value);
}
public function regenerateID($deleteOldSession=false)
{
	session_regenerate_id($deleteOldSession);
}
public function getSessionName()
{
	return session_name();
}
public function setSessionName($value)
{
	session_name($value);
}
public function getSavePath()
{
	return session_save_path();
}
public function setSavePath($value)
{
	if(is_dir($value))
```

```
                session_save_path($value);
            else
                throw new CException(Yii::t('yii','CHttpSession.savePath "{path}" is not a valid
directory.',
                    array('{path}'=>$value)));
        }
        public function getCookieParams()
        {
            return session_get_cookie_params();
        }
        public function setCookieParams($value)
        {
            $data=session_get_cookie_params();
            extract($data);
            extract($value);
            if(isset($httponly))
                session_set_cookie_params($lifetime,$path,$domain,$secure,$httponly);
            else
                session_set_cookie_params($lifetime,$path,$domain,$secure);
        }
        public function getCookieMode()
        {
            if(ini_get('session.use_cookies')==='0')
                return 'none';
            elseif(ini_get('session.use_only_cookies')==='0')
                return 'allow';
            else
                return 'only';
        }
        public function setCookieMode($value)
        {
            if($value==='none')
            {
                ini_set('session.use_cookies','0');
                ini_set('session.use_only_cookies','0');
            }
            elseif($value==='allow')
            {
                ini_set('session.use_cookies','1');
                ini_set('session.use_only_cookies','0');
            }
            elseif($value==='only')
            {
                ini_set('session.use_cookies','1');
```

```
				ini_set('session.use_only_cookies','1');
			}
			else
				throw new CException(Yii::t('yii','CHttpSession.cookieMode can only be "none", 
"allow" or "only".'));
	}
	public function getGCProbability()
	{
		return (float)(ini_get('session.gc_probability')/ini_get('session.gc_divisor')*100);
	}
	public function setGCProbability($value)
	{
		if($value>=0 && $value<=100)
		{
			// percent * 21474837 / 2147483647  ≈  percent * 0.01
			ini_set('session.gc_probability',floor($value*21474836.47));
			ini_set('session.gc_divisor',2147483647);
		}
		else
			throw new CException(Yii::t('yii','CHttpSession.gcProbability "{value}" is invalid. It 
must be a float between 0 and 100.',
				array('{value}'=>$value)));
	}
	public function getUseTransparentSessionID()
	{
		return ini_get('session.use_trans_sid')==1;
	}
	public function setUseTransparentSessionID($value)
	{
		ini_set('session.use_trans_sid',$value?'1':'0');
	}
	public function getTimeout()
	{
		return (int)ini_get('session.gc_maxlifetime');
	}
	public function setTimeout($value)
	{
		ini_set('session.gc_maxlifetime',$value);
	}
	public function openSession($savePath,$sessionName)
	{
		return true;
	}
	public function closeSession()
```

```
{
        return true;
}
public function readSession($id)
{
        return '';
}
public function writeSession($id,$data)
{
        return true;
}
public function destroySession($id)
{
        return true;
}
public function gcSession($maxLifetime)
{
        return true;
}
//------ The following methods enable CHttpSession to be CMap-like -----
public function getIterator()
{
        return new CHttpSessionIterator;
}
public function getCount()
{
        return count($_SESSION);
}
public function count()
{
        return $this->getCount();
}
public function getKeys()
{
        return array_keys($_SESSION);
}
public function get($key,$defaultValue=null)
{
        return isset($_SESSION[$key]) ? $_SESSION[$key] : $defaultValue;
}
public function itemAt($key)
{
        return isset($_SESSION[$key]) ? $_SESSION[$key] : null;
}
```

```
public function add($key,$value)
{
    $_SESSION[$key]=$value;
}
public function remove($key)
{
    if(isset($_SESSION[$key]))
    {
        $value=$_SESSION[$key];
        unset($_SESSION[$key]);
        return $value;
    }
    else
        return null;
}
public function clear()
{
    foreach(array_keys($_SESSION) as $key)
        unset($_SESSION[$key]);
}
public function contains($key)
{
    return isset($_SESSION[$key]);
}
public function toArray()
{
    return $_SESSION;
}
public function offsetExists($offset)
{
    return isset($_SESSION[$offset]);
}
public function offsetGet($offset)
{
    return isset($_SESSION[$offset]) ? $_SESSION[$offset] : null;
}
public function offsetSet($offset,$item)
{
    $_SESSION[$offset]=$item;
}
public function offsetUnset($offset)
{
    unset($_SESSION[$offset]);
}
```

```
}
class CHtml
{
        const ID_PREFIX='yt';
        public static $errorSummaryCss='errorSummary';
        public static $errorMessageCss='errorMessage';
        public static $errorCss='error';
        public static $errorContainerTag='div';
        public static $requiredCss='required';
        public static $beforeRequiredLabel='';
        public static $afterRequiredLabel=' <span class="required">*</span>';
        public static $count=0;
        public static $liveEvents=true;
        public static $closeSingleTags=true;
        public static $renderSpecialAttributesValue=true;
        public static function encode($text)
        {
                return htmlspecialchars($text,ENT_QUOTES,Yii::app()->charset);
        }
        public static function decode($text)
        {
                return htmlspecialchars_decode($text,ENT_QUOTES);
        }
        public static function encodeArray($data)
        {
                $d=array();
                foreach($data as $key=>$value)
                {
                        if(is_string($key))
                                $key=htmlspecialchars($key,ENT_QUOTES,Yii::app()->charset);
                        if(is_string($value))
                                $value=htmlspecialchars($value,ENT_QUOTES,Yii::app()->charset);
                        elseif(is_array($value))
                                $value=self::encodeArray($value);
                        $d[$key]=$value;
                }
                return $d;
        }
        public static function tag($tag,$htmlOptions=array(),$content=false,$closeTag=true)
        {
                $html='<' . $tag . self::renderAttributes($htmlOptions);
                if($content===false)
                        return $closeTag && self::$closeSingleTags ? $html.' />' : $html.'>';
                else
```

```
			return $closeTag ? $html.'>'.$content.'</'.$tag.'>' : $html.'>'.$content;
	}
	public static function openTag($tag,$htmlOptions=array())
	{
		return '<' . $tag . self::renderAttributes($htmlOptions) . '>';
	}
	public static function closeTag($tag)
	{
		return '</'.$tag.'>';
	}
	public static function cdata($text)
	{
		return '<![CDATA[' . $text . ']]>';
	}
	public static function metaTag($content,$name=null,$httpEquiv=null,$options=array())
	{
		if($name!==null)
			$options['name']=$name;
		if($httpEquiv!==null)
			$options['http-equiv']=$httpEquiv;
		$options['content']=$content;
		return self::tag('meta',$options);
	}
	public                    static                    function
linkTag($relation=null,$type=null,$href=null,$media=null,$options=array())
	{
		if($relation!==null)
			$options['rel']=$relation;
		if($type!==null)
			$options['type']=$type;
		if($href!==null)
			$options['href']=$href;
		if($media!==null)
			$options['media']=$media;
		return self::tag('link',$options);
	}
	public static function css($text,$media='')
	{
		if($media!=='')
			$media=' media="'.$media.'"';
		return                                                   "<style
type=\"text/css\"{$media}>\n/*<![CDATA[*/\n{$text}\n/*]]>*/\n</style>";
	}
	public static function refresh($seconds, $url='')
```

```
        {
                $content="$seconds";
                if($url!=='')
                        $content.=';'.self::normalizeUrl($url);
                Yii::app()->clientScript->registerMetaTag($content,null,'refresh');
        }
        public static function cssFile($url,$media='')
        {
                return CHtml::linkTag('stylesheet','text/css',$url,$media!=='' ? $media : null);
        }
        public static function script($text)
        {
                return                                                                  "<script
type=\"text/javascript\">\n/*<![CDATA[*/\n{$text}\n/*]]>*/\n</script>";
        }
        public static function scriptFile($url)
        {
                return '<script type="text/javascript" src="'.self::encode($url).'"></script>';
        }
        public static function form($action='',$method='post',$htmlOptions=array())
        {
                return self::beginForm($action,$method,$htmlOptions);
        }
        public static function beginForm($action='',$method='post',$htmlOptions=array())
        {
                $htmlOptions['action']=$url=self::normalizeUrl($action);
                $htmlOptions['method']=$method;
                $form=self::tag('form',$htmlOptions,false,false);
                $hiddens=array();
                if(!strcasecmp($method,'get') && ($pos=strpos($url,'?'))!==false)
                {
                        foreach(explode('&',substr($url,$pos+1)) as $pair)
                        {
                                if(($pos=strpos($pair,'='))!==false)

        $hiddens[]=self::hiddenField(urldecode(substr($pair,0,$pos)),urldecode(substr($pair,$pos+1
)),array('id'=>false));
                                else
                                        $hiddens[]=self::hiddenField(urldecode($pair),'',array('id'=>false));
                        }
                }
                $request=Yii::app()->request;
                if($request->enableCsrfValidation && !strcasecmp($method,'post'))
        $hiddens[]=self::hiddenField($request->csrfTokenName,$request->getCsrfToken(),array('id'=
>false));
            if($hiddens!==array())
                $form.="\n".self::tag('div',array('style'=>'display:none'),implode("\n",$hiddens));
            return $form;
        }
        public static function endForm()
        {
            return '</form>';
        }
        public static function statefulForm($action='',$method='post',$htmlOptions=array())
        {
            return self::form($action,$method,$htmlOptions)."\n".
                self::tag('div',array('style'=>'display:none'),self::pageStateField(''));
        }
        public static function pageStateField($value)
        {
            return    '<input    type="hidden"    name="'.CController::STATE_INPUT_NAME.'"
value="'.$value.'" />';
        }
        public static function link($text,$url='#',$htmlOptions=array())
        {
            if($url!=='')
                $htmlOptions['href']=self::normalizeUrl($url);
            self::clientChange('click',$htmlOptions);
            return self::tag('a',$htmlOptions,$text);
        }
        public static function mailto($text,$email='',$htmlOptions=array())
        {
            if($email==='')
                $email=$text;
            return self::link($text,'mailto:'.$email,$htmlOptions);
        }
        public static function image($src,$alt='',$htmlOptions=array())
        {
            $htmlOptions['src']=$src;
            $htmlOptions['alt']=$alt;
            return self::tag('img',$htmlOptions);
        }
        public static function button($label='button',$htmlOptions=array())
        {
            if(!isset($htmlOptions['name']))
            {
                if(!array_key_exists('name',$htmlOptions))
                    $htmlOptions['name']=self::ID_PREFIX.self::$count++;
            }
            if(!isset($htmlOptions['type']))
                $htmlOptions['type']='button';
            if(!isset($htmlOptions['value']))
                $htmlOptions['value']=$label;
            self::clientChange('click',$htmlOptions);
            return self::tag('input',$htmlOptions);
        }
        public static function htmlButton($label='button',$htmlOptions=array())
        {
            if(!isset($htmlOptions['name']))
                $htmlOptions['name']=self::ID_PREFIX.self::$count++;
            if(!isset($htmlOptions['type']))
                $htmlOptions['type']='button';
            self::clientChange('click',$htmlOptions);
            return self::tag('button',$htmlOptions,$label);
        }
        public static function submitButton($label='submit',$htmlOptions=array())
        {
            $htmlOptions['type']='submit';
            return self::button($label,$htmlOptions);
        }
        public static function resetButton($label='reset',$htmlOptions=array())
        {
            $htmlOptions['type']='reset';
            return self::button($label,$htmlOptions);
        }
        public static function imageButton($src,$htmlOptions=array())
        {
            $htmlOptions['src']=$src;
            $htmlOptions['type']='image';
            return self::button('submit',$htmlOptions);
        }
        public static function linkButton($label='submit',$htmlOptions=array())
        {
            if(!isset($htmlOptions['submit']))
                $htmlOptions['submit']=isset($htmlOptions['href']) ? $htmlOptions['href'] : '';
            return self::link($label,'#',$htmlOptions);
        }
        public static function label($label,$for,$htmlOptions=array())
        {
            if($for===false)
                unset($htmlOptions['for']);
```

```
		else
			$htmlOptions['for']=$for;
		if(isset($htmlOptions['required']))
		{
			if($htmlOptions['required'])
			{
				if(isset($htmlOptions['class']))
					$htmlOptions['class'].=' '.self::$requiredCss;
				else
					$htmlOptions['class']=self::$requiredCss;
				$label=self::$beforeRequiredLabel.$label.self::$afterRequiredLabel;
			}
			unset($htmlOptions['required']);
		}
		return self::tag('label',$htmlOptions,$label);
	}
	public static function textField($name,$value='',$htmlOptions=array())
	{
		self::clientChange('change',$htmlOptions);
		return self::inputField('text',$name,$value,$htmlOptions);
	}
	public static function hiddenField($name,$value='',$htmlOptions=array())
	{
		return self::inputField('hidden',$name,$value,$htmlOptions);
	}
	public static function passwordField($name,$value='',$htmlOptions=array())
	{
		self::clientChange('change',$htmlOptions);
		return self::inputField('password',$name,$value,$htmlOptions);
	}
	public static function fileField($name,$value='',$htmlOptions=array())
	{
		return self::inputField('file',$name,$value,$htmlOptions);
	}
	public static function textArea($name,$value='',$htmlOptions=array())
	{
		$htmlOptions['name']=$name;
		if(!isset($htmlOptions['id']))
			$htmlOptions['id']=self::getIdByName($name);
		elseif($htmlOptions['id']===false)
			unset($htmlOptions['id']);
		self::clientChange('change',$htmlOptions);
		return					self::tag('textarea',$htmlOptions,isset($htmlOptions['encode'])
&& !$htmlOptions['encode'] ? $value : self::encode($value));
```

```
	}
	public static function radioButton($name,$checked=false,$htmlOptions=array())
	{
		if($checked)
			$htmlOptions['checked']='checked';
		else
			unset($htmlOptions['checked']);
		$value=isset($htmlOptions['value']) ? $htmlOptions['value'] : 1;
		self::clientChange('click',$htmlOptions);
		if(array_key_exists('uncheckValue',$htmlOptions))
		{
			$uncheck=$htmlOptions['uncheckValue'];
			unset($htmlOptions['uncheckValue']);
		}
		else
			$uncheck=null;
		if($uncheck!==null)
		{
			// add a hidden field so that if the radio button is not selected, it still submits a
value
			if(isset($htmlOptions['id']) && $htmlOptions['id']!==false)
				$uncheckOptions=array('id'=>self::ID_PREFIX.$htmlOptions['id']);
			else
				$uncheckOptions=array('id'=>false);
			$hidden=self::hiddenField($name,$uncheck,$uncheckOptions);
		}
		else
			$hidden='';
		// add a hidden field so that if the radio button is not selected, it still submits a value
		return $hidden . self::inputField('radio',$name,$value,$htmlOptions);
	}
	public static function checkBox($name,$checked=false,$htmlOptions=array())
	{
		if($checked)
			$htmlOptions['checked']='checked';
		else
			unset($htmlOptions['checked']);
		$value=isset($htmlOptions['value']) ? $htmlOptions['value'] : 1;
		self::clientChange('click',$htmlOptions);
		if(array_key_exists('uncheckValue',$htmlOptions))
		{
			$uncheck=$htmlOptions['uncheckValue'];
			unset($htmlOptions['uncheckValue']);
		}
```

```
			else
				$uncheck=null;
			if($uncheck!==null)
			{
				// add a hidden field so that if the check box is not checked, it still submits a value
				if(isset($htmlOptions['id']) && $htmlOptions['id']!==false)
					$uncheckOptions=array('id'=>self::ID_PREFIX.$htmlOptions['id']);
				else
					$uncheckOptions=array('id'=>false);
				$hidden=self::hiddenField($name,$uncheck,$uncheckOptions);
			}
			else
				$hidden='';
			// add a hidden field so that if the check box is not checked, it still submits a value
			return $hidden . self::inputField('checkbox',$name,$value,$htmlOptions);
		}
		public static function dropDownList($name,$select,$data,$htmlOptions=array())
		{
			$htmlOptions['name']=$name;
			if(!isset($htmlOptions['id']))
				$htmlOptions['id']=self::getIdByName($name);
			elseif($htmlOptions['id']===false)
				unset($htmlOptions['id']);
			self::clientChange('change',$htmlOptions);
			$options="\n".self::listOptions($select,$data,$htmlOptions);
			$hidden='';
			if(isset($htmlOptions['multiple']))
			{
				if(substr($htmlOptions['name'],-2)!=='[]')
					$htmlOptions['name'].='[]';
				if(isset($htmlOptions['unselectValue']))
				{
					$hiddenOptions=isset($htmlOptions['id'])                              ?
array('id'=>self::ID_PREFIX.$htmlOptions['id']) : array('id'=>false);

		$hidden=self::hiddenField(substr($htmlOptions['name'],0,-2),$htmlOptions['unselectValue'],
$hiddenOptions);
					unset($htmlOptions['unselectValue']);
				}
			}
			// add a hidden field so that if the option is not selected, it still submits a value
			return $hidden . self::tag('select',$htmlOptions,$options);
		}
		public static function listBox($name,$select,$data,$htmlOptions=array())
```

```
	{
		if(!isset($htmlOptions['size']))
			$htmlOptions['size']=4;
		if(isset($htmlOptions['multiple']))
		{
			if(substr($name,-2)!=='[]')
				$name.='[]';
		}
		return self::dropDownList($name,$select,$data,$htmlOptions);
	}
	public static function checkBoxList($name,$select,$data,$htmlOptions=array())
	{
		$template=isset($htmlOptions['template'])?$htmlOptions['template']:'{input} {label}';
		$separator=isset($htmlOptions['separator'])?$htmlOptions['separator']:"<br/>\n";
		$container=isset($htmlOptions['container'])?$htmlOptions['container']:'span';
		unset($htmlOptions['template'],$htmlOptions['separator'],$htmlOptions['container']);
		if(substr($name,-2)!=='[]')
			$name.='[]';
		if(isset($htmlOptions['checkAll']))
		{
			$checkAllLabel=$htmlOptions['checkAll'];
			$checkAllLast=isset($htmlOptions['checkAllLast'])                    &&
$htmlOptions['checkAllLast'];
		}
		unset($htmlOptions['checkAll'],$htmlOptions['checkAllLast']);

	$labelOptions=isset($htmlOptions['labelOptions'])?$htmlOptions['labelOptions']:array();
		unset($htmlOptions['labelOptions']);
		$items=array();
		$baseID=isset($htmlOptions['baseID'])     ?     $htmlOptions['baseID']     :
self::getIdByName($name);
		unset($htmlOptions['baseID']);
		$id=0;
		$checkAll=true;
		foreach($data as $value=>$label)
		{
			$checked=!is_array($select)  &&  !strcmp($value,$select)  ||  is_array($select)  &&
in_array($value,$select);
			$checkAll=$checkAll && $checked;
			$htmlOptions['value']=$value;
			$htmlOptions['id']=$baseID.'_'.$id++;
			$option=self::checkBox($name,$checked,$htmlOptions);
			$label=self::label($label,$htmlOptions['id'],$labelOptions);
			$items[]=strtr($template,array('{input}'=>$option,'{label}'=>$label));
			}
			if(isset($checkAllLabel))
			{
				$htmlOptions['value']=1;
				$htmlOptions['id']=$id=$baseID.'_all';
				$option=self::checkBox($id,$checkAll,$htmlOptions);
				$label=self::label($checkAllLabel,$id,$labelOptions);
				$item=strtr($template,array('{input}'=>$option,'{label}'=>$label));
				if($checkAllLast)
					$items[]=$item;
				else
					array_unshift($items,$item);
				$name=strtr($name,array('['=>'\\[',']'=>'\\]'));
				$js=<<<EOD
jQuery('#$id').click(function() {
	jQuery("input[name='$name']").prop('checked', this.checked);
});
jQuery("input[name='$name']").click(function() {
	jQuery('#$id').prop('checked', !jQuery("input[name='$name']:not(:checked)").length);
});
jQuery('#$id').prop('checked', !jQuery("input[name='$name']:not(:checked)").length);
EOD;
				$cs=Yii::app()->getClientScript();
				$cs->registerCoreScript('jquery');
				$cs->registerScript($id,$js);
			}
			if(empty($container))
				return implode($separator,$items);
			else
				return self::tag($container,array('id'=>$baseID),implode($separator,$items));
		}
		public static function radioButtonList($name,$select,$data,$htmlOptions=array())
		{
			$template=isset($htmlOptions['template'])?$htmlOptions['template']:'{input} {label}';
			$separator=isset($htmlOptions['separator'])?$htmlOptions['separator']:"<br/>\n";
			$container=isset($htmlOptions['container'])?$htmlOptions['container']:'span';
			unset($htmlOptions['template'],$htmlOptions['separator'],$htmlOptions['container']);

		$labelOptions=isset($htmlOptions['labelOptions'])?$htmlOptions['labelOptions']:array();
			unset($htmlOptions['labelOptions']);
			$items=array();
			$baseID=isset($htmlOptions['baseID'])          ?          $htmlOptions['baseID']          :
self::getIdByName($name);
			unset($htmlOptions['baseID']);
```

```
        $id=0;
        foreach($data as $value=>$label)
        {
            $checked=!strcmp($value,$select);
            $htmlOptions['value']=$value;
            $htmlOptions['id']=$baseID.'_'.$id++;
            $option=self::radioButton($name,$checked,$htmlOptions);
            $label=self::label($label,$htmlOptions['id'],$labelOptions);
            $items[]=strtr($template,array('{input}'=>$option,'{label}'=>$label));
        }
        if(empty($container))
            return implode($separator,$items);
        else
            return self::tag($container,array('id'=>$baseID),implode($separator,$items));
    }
    public static function ajaxLink($text,$url,$ajaxOptions=array(),$htmlOptions=array())
    {
        if(!isset($htmlOptions['href']))
            $htmlOptions['href']='#';
        $ajaxOptions['url']=$url;
        $htmlOptions['ajax']=$ajaxOptions;
        self::clientChange('click',$htmlOptions);
        return self::tag('a',$htmlOptions,$text);
    }
    public static function ajaxButton($label,$url,$ajaxOptions=array(),$htmlOptions=array())
    {
        $ajaxOptions['url']=$url;
        $htmlOptions['ajax']=$ajaxOptions;
        return self::button($label,$htmlOptions);
    }
    public                              static                              function
ajaxSubmitButton($label,$url,$ajaxOptions=array(),$htmlOptions=array())
    {
        $ajaxOptions['type']='POST';
        $htmlOptions['type']='submit';
        return self::ajaxButton($label,$url,$ajaxOptions,$htmlOptions);
    }
    public static function ajax($options)
    {
        Yii::app()->getClientScript()->registerCoreScript('jquery');
        if(!isset($options['url']))
            $options['url']=new CJavaScriptExpression('location.href');
        else
            $options['url']=self::normalizeUrl($options['url']);
```

```
		if(!isset($options['cache']))
			$options['cache']=false;
		if(!isset($options['data']) && isset($options['type']))
			$options['data']=new
CJavaScriptExpression('jQuery(this).parents("form").serialize()');
		foreach(array('beforeSend','complete','error','success') as $name)
		{
			if(isset($options[$name])	&&	!($options[$name]	instanceof
CJavaScriptExpression))
				$options[$name]=new CJavaScriptExpression($options[$name]);
		}
		if(isset($options['update']))
		{
			if(!isset($options['success']))
				$options['success']=new
CJavaScriptExpression('function(html){jQuery("'.$options['update'].'").html(html)}');
			unset($options['update']);
		}
		if(isset($options['replace']))
		{
			if(!isset($options['success']))
				$options['success']=new
CJavaScriptExpression('function(html){jQuery("'.$options['replace'].'").replaceWith(html)}');
			unset($options['replace']);
		}
		return 'jQuery.ajax('.CJavaScript::encode($options).');';
	}
	public static function asset($path,$hashByName=false)
	{
		return Yii::app()->getAssetManager()->publish($path,$hashByName);
	}
	public static function normalizeUrl($url)
	{
		if(is_array($url))
		{
			if(isset($url[0]))
			{
				if(($c=Yii::app()->getController())!==null)
					$url=$c->createUrl($url[0],array_splice($url,1));
				else
					$url=Yii::app()->createUrl($url[0],array_splice($url,1));
			}
			else
				$url='';
```

```
        }
        return $url==='' ? Yii::app()->getRequest()->getUrl() : $url;
}
protected static function inputField($type,$name,$value,$htmlOptions)
{
        $htmlOptions['type']=$type;
        $htmlOptions['value']=$value;
        $htmlOptions['name']=$name;
        if(!isset($htmlOptions['id']))
                $htmlOptions['id']=self::getIdByName($name);
        elseif($htmlOptions['id']===false)
                unset($htmlOptions['id']);
        return self::tag('input',$htmlOptions);
}
public static function activeLabel($model,$attribute,$htmlOptions=array())
{
        if(isset($htmlOptions['for']))
        {
                $for=$htmlOptions['for'];
                unset($htmlOptions['for']);
        }
        else
                $for=self::getIdByName(self::resolveName($model,$attribute));
        if(isset($htmlOptions['label']))
        {
                if(($label=$htmlOptions['label'])===false)
                        return '';
                unset($htmlOptions['label']);
        }
        else
                $label=$model->getAttributeLabel($attribute);
        if($model->hasErrors($attribute))
                self::addErrorCss($htmlOptions);
        return self::label($label,$for,$htmlOptions);
}
public static function activeLabelEx($model,$attribute,$htmlOptions=array())
{
        $realAttribute=$attribute;
        self::resolveName($model,$attribute); // strip off square brackets if any
        $htmlOptions['required']=$model->isAttributeRequired($attribute);
        return self::activeLabel($model,$realAttribute,$htmlOptions);
}
public static function activeTextField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        self::clientChange('change',$htmlOptions);
        return self::activeInputField('text',$model,$attribute,$htmlOptions);
}
public static function activeUrlField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        self::clientChange('change',$htmlOptions);
        return self::activeInputField('url',$model,$attribute,$htmlOptions);
}
public static function activeEmailField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        self::clientChange('change',$htmlOptions);
        return self::activeInputField('email',$model,$attribute,$htmlOptions);
}
public static function activeNumberField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        self::clientChange('change',$htmlOptions);
        return self::activeInputField('number',$model,$attribute,$htmlOptions);
}
public static function activeRangeField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        self::clientChange('change',$htmlOptions);
        return self::activeInputField('range',$model,$attribute,$htmlOptions);
}
public static function activeDateField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        self::clientChange('change',$htmlOptions);
        return self::activeInputField('date',$model,$attribute,$htmlOptions);
}
public static function activeHiddenField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        return self::activeInputField('hidden',$model,$attribute,$htmlOptions);
}
public static function activePasswordField($model,$attribute,$htmlOptions=array())
{
        self::resolveNameID($model,$attribute,$htmlOptions);
        self::clientChange('change',$htmlOptions);
        return self::activeInputField('password',$model,$attribute,$htmlOptions);
```

```
	}
	public static function activeTextArea($model,$attribute,$htmlOptions=array())
	{
		self::resolveNameID($model,$attribute,$htmlOptions);
		self::clientChange('change',$htmlOptions);
		if($model->hasErrors($attribute))
			self::addErrorCss($htmlOptions);
		if(isset($htmlOptions['value']))
		{
			$text=$htmlOptions['value'];
			unset($htmlOptions['value']);
		}
		else
			$text=self::resolveValue($model,$attribute);
		return                            self::tag('textarea',$htmlOptions,isset($htmlOptions['encode'])
&& !$htmlOptions['encode'] ? $text : self::encode($text));
	}
	public static function activeFileField($model,$attribute,$htmlOptions=array())
	{
		self::resolveNameID($model,$attribute,$htmlOptions);
		// add a hidden field so that if a model only has a file field, we can
		// still use isset($_POST[$modelClass]) to detect if the input is submitted
		$hiddenOptions=isset($htmlOptions['id'])                                              ?
array('id'=>self::ID_PREFIX.$htmlOptions['id']) : array('id'=>false);
		return self::hiddenField($htmlOptions['name'],'',$hiddenOptions)
			. self::activeInputField('file',$model,$attribute,$htmlOptions);
	}
	public static function activeRadioButton($model,$attribute,$htmlOptions=array())
	{
		self::resolveNameID($model,$attribute,$htmlOptions);
		if(!isset($htmlOptions['value']))
			$htmlOptions['value']=1;
		if(!isset($htmlOptions['checked'])                                                  &&
self::resolveValue($model,$attribute)==$htmlOptions['value'])
			$htmlOptions['checked']='checked';
		self::clientChange('click',$htmlOptions);
		if(array_key_exists('uncheckValue',$htmlOptions))
		{
			$uncheck=$htmlOptions['uncheckValue'];
			unset($htmlOptions['uncheckValue']);
		}
		else
			$uncheck='0';
		$hiddenOptions=isset($htmlOptions['id'])                                              ?
```

```
array('id'=>self::ID_PREFIX.$htmlOptions['id']) : array('id'=>false);
		$hidden=$uncheck!==null                                                          ?
self::hiddenField($htmlOptions['name'],$uncheck,$hiddenOptions) : '';
		// add a hidden field so that if the radio button is not selected, it still submits a value
		return $hidden . self::activeInputField('radio',$model,$attribute,$htmlOptions);
	}
	public static function activeCheckBox($model,$attribute,$htmlOptions=array())
	{
		self::resolveNameID($model,$attribute,$htmlOptions);
		if(!isset($htmlOptions['value']))
			$htmlOptions['value']=1;
		if(!isset($htmlOptions['checked'])                                             &&
self::resolveValue($model,$attribute)==$htmlOptions['value'])
			$htmlOptions['checked']='checked';
		self::clientChange('click',$htmlOptions);
		if(array_key_exists('uncheckValue',$htmlOptions))
		{
			$uncheck=$htmlOptions['uncheckValue'];
			unset($htmlOptions['uncheckValue']);
		}
		else
			$uncheck='0';
		$hiddenOptions=isset($htmlOptions['id'])                                         ?
array('id'=>self::ID_PREFIX.$htmlOptions['id']) : array('id'=>false);
		$hidden=$uncheck!==null                                                          ?
self::hiddenField($htmlOptions['name'],$uncheck,$hiddenOptions) : '';
		return $hidden . self::activeInputField('checkbox',$model,$attribute,$htmlOptions);
	}
	public static function activeDropDownList($model,$attribute,$data,$htmlOptions=array())
	{
		self::resolveNameID($model,$attribute,$htmlOptions);
		$selection=self::resolveValue($model,$attribute);
		$options="\n".self::listOptions($selection,$data,$htmlOptions);
		self::clientChange('change',$htmlOptions);
		if($model->hasErrors($attribute))
			self::addErrorCss($htmlOptions);
		$hidden='';
		if(isset($htmlOptions['multiple']))
		{
			if(substr($htmlOptions['name'],-2)!=='[]')
				$htmlOptions['name'].='[]';
			if(isset($htmlOptions['unselectValue']))
			{
				$hiddenOptions=isset($htmlOptions['id'])                                 ?
array('id'=>self::ID_PREFIX.$htmlOptions['id']) : array('id'=>false);

        $hidden=self::hiddenField(substr($htmlOptions['name'],0,-2),$htmlOptions['unselectValue'],
$hiddenOptions);
                        unset($htmlOptions['unselectValue']);
                    }
                }
                return $hidden . self::tag('select',$htmlOptions,$options);
        }
        public static function activeListBox($model,$attribute,$data,$htmlOptions=array())
        {
                if(!isset($htmlOptions['size']))
                        $htmlOptions['size']=4;
                return self::activeDropDownList($model,$attribute,$data,$htmlOptions);
        }
        public static function activeCheckBoxList($model,$attribute,$data,$htmlOptions=array())
        {
                self::resolveNameID($model,$attribute,$htmlOptions);
                $selection=self::resolveValue($model,$attribute);
                if($model->hasErrors($attribute))
                        self::addErrorCss($htmlOptions);
                $name=$htmlOptions['name'];
                unset($htmlOptions['name']);
                if(array_key_exists('uncheckValue',$htmlOptions))
                {
                        $uncheck=$htmlOptions['uncheckValue'];
                        unset($htmlOptions['uncheckValue']);
                }
                else
                        $uncheck='';
                $hiddenOptions=isset($htmlOptions['id'])                                        ?
array('id'=>self::ID_PREFIX.$htmlOptions['id']) : array('id'=>false);
                $hidden=$uncheck!==null ? self::hiddenField($name,$uncheck,$hiddenOptions) : '';
                return $hidden . self::checkBoxList($name,$selection,$data,$htmlOptions);
        }
        public static function activeRadioButtonList($model,$attribute,$data,$htmlOptions=array())
        {
                self::resolveNameID($model,$attribute,$htmlOptions);
                $selection=self::resolveValue($model,$attribute);
                if($model->hasErrors($attribute))
                        self::addErrorCss($htmlOptions);
                $name=$htmlOptions['name'];
                unset($htmlOptions['name']);
                if(array_key_exists('uncheckValue',$htmlOptions))
        {
            $uncheck=$htmlOptions['uncheckValue'];
            unset($htmlOptions['uncheckValue']);
        }
        else
            $uncheck='';
        $hiddenOptions=isset($htmlOptions['id'])                                    ?
array('id'=>self::ID_PREFIX.$htmlOptions['id']) : array('id'=>false);
        $hidden=$uncheck!==null ? self::hiddenField($name,$uncheck,$hiddenOptions) : '';
        return $hidden . self::radioButtonList($name,$selection,$data,$htmlOptions);
    }
    public                          static                          function
errorSummary($model,$header=null,$footer=null,$htmlOptions=array())
    {
        $content='';
        if(!is_array($model))
            $model=array($model);
        if(isset($htmlOptions['firstError']))
        {
            $firstError=$htmlOptions['firstError'];
            unset($htmlOptions['firstError']);
        }
        else
            $firstError=false;
        foreach($model as $m)
        {
            foreach($m->getErrors() as $errors)
            {
                foreach($errors as $error)
                {
                    if($error!='')
                        $content.="<li>$error</li>\n";
                    if($firstError)
                        break;
                }
            }
        }
        if($content!=='')
        {
            if($header===null)
                $header='<p>'.Yii::t('yii','Please fix the following input errors:').'</p>';
            if(!isset($htmlOptions['class']))
                $htmlOptions['class']=self::$errorSummaryCss;
            return self::tag('div',$htmlOptions,$header."\n<ul>\n$content</ul>".$footer);
        }
        else
                return '';
}
public static function error($model,$attribute,$htmlOptions=array())
{
        self::resolveName($model,$attribute); // turn [a][b]attr into attr
        $error=$model->getError($attribute);
        if($error!='')
        {
                if(!isset($htmlOptions['class']))
                        $htmlOptions['class']=self::$errorMessageCss;
                return self::tag(self::$errorContainerTag,$htmlOptions,$error);
        }
        else
                return '';
}
public static function listData($models,$valueField,$textField,$groupField='')
{
        $listData=array();
        if($groupField==='')
        {
                foreach($models as $model)
                {
                        $value=self::value($model,$valueField);
                        $text=self::value($model,$textField);
                        $listData[$value]=$text;
                }
        }
        else
        {
                foreach($models as $model)
                {
                        $group=self::value($model,$groupField);
                        $value=self::value($model,$valueField);
                        $text=self::value($model,$textField);
                        if($group===null)
                                $listData[$value]=$text;
                        else
                                $listData[$group][$value]=$text;
                }
        }
        return $listData;
}
```

```
public static function value($model,$attribute,$defaultValue=null)
{
    if(is_scalar($attribute) || $attribute===null)
        foreach(explode('.',$attribute) as $name)
        {
            if(is_object($model) && isset($model->$name))
                $model=$model->$name;
            elseif(is_array($model) && isset($model[$name]))
                $model=$model[$name];
            else
                return $defaultValue;
        }
    else
        return call_user_func($attribute,$model);
    return $model;
}
public static function getIdByName($name)
{
    return str_replace(array('[]', '][', '[', ']', ' '), array('', '_', '_', '', '_'), $name);
}
public static function activeId($model,$attribute)
{
    return self::getIdByName(self::activeName($model,$attribute));
}
public static function activeName($model,$attribute)
{
    $a=$attribute; // because the attribute name may be changed by resolveName
    return self::resolveName($model,$a);
}
protected static function activeInputField($type,$model,$attribute,$htmlOptions)
{
    $htmlOptions['type']=$type;
    if($type==='text' || $type==='password')
    {
        if(!isset($htmlOptions['maxlength']))
        {
            foreach($model->getValidators($attribute) as $validator)
            {
                if($validator instanceof CStringValidator && $validator->max!==null)
                {
                    $htmlOptions['maxlength']=$validator->max;
                    break;
                }
            }
```

```
                }
                elseif($htmlOptions['maxlength']===false)
                        unset($htmlOptions['maxlength']);
        }
        if($type==='file')
                unset($htmlOptions['value']);
        elseif(!isset($htmlOptions['value']))
                $htmlOptions['value']=self::resolveValue($model,$attribute);
        if($model->hasErrors($attribute))
                self::addErrorCss($htmlOptions);
        return self::tag('input',$htmlOptions);
}
public static function listOptions($selection,$listData,&$htmlOptions)
{
        $raw=isset($htmlOptions['encode']) && !$htmlOptions['encode'];
        $content='';
        if(isset($htmlOptions['prompt']))
        {
                $content.='<option           value="">'.strtr($htmlOptions['prompt'],array('<'=>'&lt;',
'>'=>'&gt;'))."</option>\n";
                unset($htmlOptions['prompt']);
        }
        if(isset($htmlOptions['empty']))
        {
                if(!is_array($htmlOptions['empty']))
                        $htmlOptions['empty']=array(''=>$htmlOptions['empty']);
                foreach($htmlOptions['empty'] as $value=>$label)
                        $content.='<option
value="'.self::encode($value).'">'.strtr($label,array('<'=>'&lt;', '>'=>'&gt;'))."</option>\n";
                unset($htmlOptions['empty']);
        }
        if(isset($htmlOptions['options']))
        {
                $options=$htmlOptions['options'];
                unset($htmlOptions['options']);
        }
        else
                $options=array();
        $key=isset($htmlOptions['key']) ? $htmlOptions['key'] : 'primaryKey';
        if(is_array($selection))
        {
                foreach($selection as $i=>$item)
                {
                        if(is_object($item))
                        $selection[$i]=$item->$key;
                }
            }
            elseif(is_object($selection))
                $selection=$selection->$key;
            foreach($listData as $key=>$value)
            {
                if(is_array($value))
                {
                    $content.='<optgroup label="'.($raw?$key : self::encode($key))."\">\n";
                    $dummy=array('options'=>$options);
                    if(isset($htmlOptions['encode']))
                        $dummy['encode']=$htmlOptions['encode'];
                    $content.=self::listOptions($selection,$value,$dummy);
                    $content.='</optgroup>'."\n";
                }
                else
                {
                    $attributes=array('value'=>(string)$key, 'encode'=>!$raw);
                    if(!is_array($selection)  &&  !strcmp($key,$selection)  ||  is_array($selection)
&& in_array($key,$selection))
                        $attributes['selected']='selected';
                    if(isset($options[$key]))
                        $attributes=array_merge($attributes,$options[$key]);
                    $content.=self::tag('option',$attributes,$raw?(string)$value                    :
self::encode((string)$value))."\n";
                }
            }
            unset($htmlOptions['key']);
            return $content;
        }
        protected static function clientChange($event,&$htmlOptions)
        {
            if(!isset($htmlOptions['submit'])            &&            !isset($htmlOptions['confirm'])
&& !isset($htmlOptions['ajax']))
                return;
            if(isset($htmlOptions['live']))
            {
                $live=$htmlOptions['live'];
                unset($htmlOptions['live']);
            }
            else
                $live = self::$liveEvents;
            if(isset($htmlOptions['return']) && $htmlOptions['return'])
```

```
                $return='return true';
            else
                $return='return false';
            if(isset($htmlOptions['on'.$event]))
            {
                $handler=trim($htmlOptions['on'.$event],';').';';
                unset($htmlOptions['on'.$event]);
            }
            else
                $handler='';
            if(isset($htmlOptions['id']))
                $id=$htmlOptions['id'];
            else

        $id=$htmlOptions['id']=isset($htmlOptions['name'])?$htmlOptions['name']:self::ID_PREFIX.s
elf::$count++;
            $cs=Yii::app()->getClientScript();
            $cs->registerCoreScript('jquery');
            if(isset($htmlOptions['submit']))
            {
                $cs->registerCoreScript('yii');
                $request=Yii::app()->getRequest();
                if($request->enableCsrfValidation        &&        isset($htmlOptions['csrf'])        &&
$htmlOptions['csrf'])

        $htmlOptions['params'][$request->csrfTokenName]=$request->getCsrfToken();
                if(isset($htmlOptions['params']))
                    $params=CJavaScript::encode($htmlOptions['params']);
                else
                    $params='{}';
                if($htmlOptions['submit']!=='')
                    $url=CJavaScript::quote(self::normalizeUrl($htmlOptions['submit']));
                else
                    $url='';
                $handler.="jQuery.yii.submitForm(this,'$url',$params);{$return};";
            }
            if(isset($htmlOptions['ajax']))
                $handler.=self::ajax($htmlOptions['ajax'])."{$return};";
            if(isset($htmlOptions['confirm']))
            {
                $confirm='confirm(\''.CJavaScript::quote($htmlOptions['confirm']).'\')';
                if($handler!=='')
                    $handler="if($confirm) {".$handler."} else return false;";
                else
```

```
                    $handler="return $confirm;";
            }
            if($live)
                $cs->registerScript('Yii.CHtml.#'                          .                              $id,
"jQuery('body').on('$event','#$id',function(){{$handler}});");
            else
                $cs->registerScript('Yii.CHtml.#'          .          $id,          "jQuery('#$id').on('$event',
function(){{$handler}});");

        unset($htmlOptions['params'],$htmlOptions['submit'],$htmlOptions['ajax'],$htmlOptions['c
onfirm'],$htmlOptions['return'],$htmlOptions['csrf']);
        }
        public static function resolveNameID($model,&$attribute,&$htmlOptions)
        {
            if(!isset($htmlOptions['name']))
                $htmlOptions['name']=self::resolveName($model,$attribute);
            if(!isset($htmlOptions['id']))
                $htmlOptions['id']=self::getIdByName($htmlOptions['name']);
            elseif($htmlOptions['id']===false)
                unset($htmlOptions['id']);
        }
        public static function resolveName($model,&$attribute)
        {
            if(($pos=strpos($attribute,'['))!==false)
            {
                if($pos!==0)    // e.g. name[a][b]
                    return
get_class($model).'['.substr($attribute,0,$pos).']'.substr($attribute,$pos);
                if(($pos=strrpos($attribute,']'))!==false  &&  $pos!==strlen($attribute)-1)    //  e.g.
[a][b]name
                {
                    $sub=substr($attribute,0,$pos+1);
                    $attribute=substr($attribute,$pos+1);
                    return get_class($model).$sub.'['.$attribute.']';
                }
                if(preg_match('/\](\w+\[.*)$/',$attribute,$matches))
                {

        $name=get_class($model).'['.str_replace(']','][',trim(strtr($attribute,array(']['=>']','['=>']')),']')
).']';
                    $attribute=$matches[1];
                    return $name;
                }
            }
        return get_class($model).'['.$attribute.']';
}
public static function resolveValue($model,$attribute)
{
        if(($pos=strpos($attribute,'['))!==false)
        {
                if($pos===0) // [a]name[b][c], should ignore [a]
                {
                        if(preg_match('/\](\w+(\[.+)?)/',$attribute,$matches))
                                $attribute=$matches[1]; // we get: name[b][c]
                        if(($pos=strpos($attribute,'['))===false)
                                return $model->$attribute;
                }
                $name=substr($attribute,0,$pos);
                $value=$model->$name;
                foreach(explode('][',rtrim(substr($attribute,$pos+1),']')) as $id)
                {
                        if((is_array($value) || $value instanceof ArrayAccess) && isset($value[$id]))
                                $value=$value[$id];
                        else
                                return null;
                }
                return $value;
        }
        else
                return $model->$attribute;
}
protected static function addErrorCss(&$htmlOptions)
{
        if(empty(self::$errorCss))
                return;
        if(isset($htmlOptions['class']))
                $htmlOptions['class'].=' '.self::$errorCss;
        else
                $htmlOptions['class']=self::$errorCss;
}
public static function renderAttributes($htmlOptions)
{
        static $specialAttributes=array(
                'async'=>1,
                'autofocus'=>1,
                'autoplay'=>1,
                'checked'=>1,
                'controls'=>1,
                'declare'=>1,
                'default'=>1,
                'defer'=>1,
                'disabled'=>1,
                'formnovalidate'=>1,
                'hidden'=>1,
                'ismap'=>1,
                'loop'=>1,
                'multiple'=>1,
                'muted'=>1,
                'nohref'=>1,
                'noresize'=>1,
                'novalidate'=>1,
                'open'=>1,
                'readonly'=>1,
                'required'=>1,
                'reversed'=>1,
                'scoped'=>1,
                'seamless'=>1,
                'selected'=>1,
                'typemustmatch'=>1,
        );
        if($htmlOptions===array())
                return '';
        $html='';
        if(isset($htmlOptions['encode']))
        {
                $raw=!$htmlOptions['encode'];
                unset($htmlOptions['encode']);
        }
        else
                $raw=false;
        foreach($htmlOptions as $name=>$value)
        {
                if(isset($specialAttributes[$name]))
                {
                        if($value)
                        {
                                $html .= ' ' . $name;
                                if(self::$renderSpecialAttributesValue)
                                    $html .= '="' . $name . '"';
                        }
                }
                elseif($value!==null)
```

```
                $html .= ' ' . $name . '="' . ($raw ? $value : self::encode($value)) . '"';
            }
            return $html;
        }
}
class CWidgetFactory extends CApplicationComponent implements IWidgetFactory
{
        public $enableSkin=false;
        public $widgets=array();
        public $skinnableWidgets;
        public $skinPath;
        private $_skins=array();    // class name, skin name, property name => value
        public function init()
        {
            parent::init();
            if($this->enableSkin && $this->skinPath===null)
                $this->skinPath=Yii::app()->getViewPath().DIRECTORY_SEPARATOR.'skins';
        }
        public function createWidget($owner,$className,$properties=array())
        {
            $className=Yii::import($className,true);
            $widget=new $className($owner);
            if(isset($this->widgets[$className]))
                $properties=$properties===array() ? $this->widgets[$className] :
CMap::mergeArray($this->widgets[$className],$properties);
            if($this->enableSkin)
            {
                if($this->skinnableWidgets===null ||
in_array($className,$this->skinnableWidgets))
                {
                    $skinName=isset($properties['skin']) ? $properties['skin'] : 'default';
                    if($skinName!==false &&
($skin=$this->getSkin($className,$skinName))!==array())
                        $properties=$properties===array() ? $skin :
CMap::mergeArray($skin,$properties);
                }
            }
            foreach($properties as $name=>$value)
                $widget->$name=$value;
            return $widget;
        }
        protected function getSkin($className,$skinName)
        {
            if(!isset($this->_skins[$className][$skinName]))
```

```
                {
                        $skinFile=$this->skinPath.DIRECTORY_SEPARATOR.$className.'.php';
                        if(is_file($skinFile))
                                $this->_skins[$className]=require($skinFile);
                        else
                                $this->_skins[$className]=array();
                        if(($theme=Yii::app()->getTheme())!==null)
                        {
                                $skinFile=$theme->getSkinPath().DIRECTORY_SEPARATOR.$className.'.php';
                                if(is_file($skinFile))
                                {
                                        $skins=require($skinFile);
                                        foreach($skins as $name=>$skin)
                                                $this->_skins[$className][$name]=$skin;
                                }
                        }
                        if(!isset($this->_skins[$className][$skinName]))
                                $this->_skins[$className][$skinName]=array();
                }
                return $this->_skins[$className][$skinName];
        }
}
class CWidget extends CBaseController
{
        public $actionPrefix;
        public $skin='default';
        private static $_viewPaths;
        private static $_counter=0;
        private $_id;
        private $_owner;
        public static function actions()
        {
                return array();
        }
        public function __construct($owner=null)
        {
                $this->_owner=$owner===null?Yii::app()->getController():$owner;
        }
        public function getOwner()
        {
                return $this->_owner;
        }
        public function getId($autoGenerate=true)
        {
```

```
		if($this->_id!==null)
			return $this->_id;
		elseif($autoGenerate)
			return $this->_id='yw'.self::$_counter++;
	}
	public function setId($value)
	{
		$this->_id=$value;
	}
	public function getController()
	{
		if($this->_owner instanceof CController)
			return $this->_owner;
		else
			return Yii::app()->getController();
	}
	public function init()
	{
	}
	public function run()
	{
	}
	public function getViewPath($checkTheme=false)
	{
		$className=get_class($this);
		if(isset(self::$_viewPaths[$className]))
			return self::$_viewPaths[$className];
		else
		{
			if($checkTheme && ($theme=Yii::app()->getTheme())!==null)
			{
				$path=$theme->getViewPath().DIRECTORY_SEPARATOR;
				if(strpos($className,'\\')!==false) // namespaced class
					$path.=str_replace('\\','_',ltrim($className,'\\'));
				else
					$path.=$className;
				if(is_dir($path))
					return self::$_viewPaths[$className]=$path;
			}
			$class=new ReflectionClass($className);
			return
self::$_viewPaths[$className]=dirname($class->getFileName()).DIRECTORY_SEPARATOR.'views';
		}
	}
	public function getViewFile($viewName)
	{
		if(($renderer=Yii::app()->getViewRenderer())!==null)
			$extension=$renderer->fileExtension;
		else
			$extension='.php';
		if(strpos($viewName,'.')) // a path alias
			$viewFile=Yii::getPathOfAlias($viewName);
		else
		{
			$viewFile=$this->getViewPath(true).DIRECTORY_SEPARATOR.$viewName;
			if(is_file($viewFile.$extension))
				return Yii::app()->findLocalizedFile($viewFile.$extension);
			elseif($extension!=='.php' && is_file($viewFile.'.php'))
				return Yii::app()->findLocalizedFile($viewFile.'.php');
			$viewFile=$this->getViewPath(false).DIRECTORY_SEPARATOR.$viewName;
		}
		if(is_file($viewFile.$extension))
			return Yii::app()->findLocalizedFile($viewFile.$extension);
		elseif($extension!=='.php' && is_file($viewFile.'.php'))
			return Yii::app()->findLocalizedFile($viewFile.'.php');
		else
			return false;
	}
	public function render($view,$data=null,$return=false)
	{
		if(($viewFile=$this->getViewFile($view))!==false)
			return $this->renderFile($viewFile,$data,$return);
		else
			throw new CException(Yii::t('yii','{widget} cannot find the view "{view}".',
				array('{widget}'=>get_class($this), '{view}'=>$view)));
	}
}
class CClientScript extends CApplicationComponent
{
	const POS_HEAD=0;
	const POS_BEGIN=1;
	const POS_END=2;
	const POS_LOAD=3;
	const POS_READY=4;
	public $enableJavaScript=true;
	public $scriptMap=array();
	public $packages=array();
	public $corePackages;
public $scripts=array();
protected $cssFiles=array();
protected $scriptFiles=array();
protected $metaTags=array();
protected $linkTags=array();
protected $css=array();
protected $hasScripts=false;
protected $coreScripts=array();
public $coreScriptPosition=self::POS_HEAD;
public $defaultScriptFilePosition=self::POS_HEAD;
public $defaultScriptPosition=self::POS_READY;
private $_baseUrl;
public function reset()
{
	$this->hasScripts=false;
	$this->coreScripts=array();
	$this->cssFiles=array();
	$this->css=array();
	$this->scriptFiles=array();
	$this->scripts=array();
	$this->metaTags=array();
	$this->linkTags=array();
	$this->recordCachingAction('clientScript','reset',array());
}
public function render(&$output)
{
	if(!$this->hasScripts)
		return;
	$this->renderCoreScripts();
	if(!empty($this->scriptMap))
		$this->remapScripts();
	$this->unifyScripts();
	$this->renderHead($output);
	if($this->enableJavaScript)
	{
		$this->renderBodyBegin($output);
		$this->renderBodyEnd($output);
	}
}
protected function unifyScripts()
{
	if(!$this->enableJavaScript)
		return;
	$map=array();
```

```
		if(isset($this->scriptFiles[self::POS_HEAD]))
			$map=$this->scriptFiles[self::POS_HEAD];
		if(isset($this->scriptFiles[self::POS_BEGIN]))
		{
			foreach($this->scriptFiles[self::POS_BEGIN] as $key=>$scriptFile)
			{
				if(isset($map[$scriptFile]))
					unset($this->scriptFiles[self::POS_BEGIN][$key]);
				else
					$map[$scriptFile]=true;
			}
		}
		if(isset($this->scriptFiles[self::POS_END]))
		{
			foreach($this->scriptFiles[self::POS_END] as $key=>$scriptFile)
			{
				if(isset($map[$scriptFile]))
					unset($this->scriptFiles[self::POS_END][$key]);
			}
		}
	}
	protected function remapScripts()
	{
		$cssFiles=array();
		foreach($this->cssFiles as $url=>$media)
		{
			$name=basename($url);
			if(isset($this->scriptMap[$name]))
			{
				if($this->scriptMap[$name]!==false)
					$cssFiles[$this->scriptMap[$name]]=$media;
			}
			elseif(isset($this->scriptMap['*.css']))
			{
				if($this->scriptMap['*.css']!==false)
					$cssFiles[$this->scriptMap['*.css']]=$media;
			}
			else
				$cssFiles[$url]=$media;
		}
		$this->cssFiles=$cssFiles;
		$jsFiles=array();
		foreach($this->scriptFiles as $position=>$scripts)
		{
			$jsFiles[$position]=array();
			foreach($scripts as $key=>$script)
			{
				$name=basename($script);
				if(isset($this->scriptMap[$name]))
				{
					if($this->scriptMap[$name]!==false)

$jsFiles[$position][$this->scriptMap[$name]]=$this->scriptMap[$name];
				}
				elseif(isset($this->scriptMap['*.js']))
				{
					if($this->scriptMap['*.js']!==false)
						$jsFiles[$position][$this->scriptMap['*.js']]=$this->scriptMap['*.js'];
				}
				else
					$jsFiles[$position][$key]=$script;
			}
		}
		$this->scriptFiles=$jsFiles;
	}
	public function renderCoreScripts()
	{
		if($this->coreScripts===null)
			return;
		$cssFiles=array();
		$jsFiles=array();
		foreach($this->coreScripts as $name=>$package)
		{
			$baseUrl=$this->getPackageBaseUrl($name);
			if(!empty($package['js']))
			{
				foreach($package['js'] as $js)
					$jsFiles[$baseUrl.'/'.$js]=$baseUrl.'/'.$js;
			}
			if(!empty($package['css']))
			{
				foreach($package['css'] as $css)
					$cssFiles[$baseUrl.'/'.$css]='';
			}
		}
		// merge in place
		if($cssFiles!==array())
		{
			foreach($this->cssFiles as $cssFile=>$media)
				$cssFiles[$cssFile]=$media;
			$this->cssFiles=$cssFiles;
		}
		if($jsFiles!==array())
		{
			if(isset($this->scriptFiles[$this->coreScriptPosition]))
			{
				foreach($this->scriptFiles[$this->coreScriptPosition] as $url)
					$jsFiles[$url]=$url;
			}
			$this->scriptFiles[$this->coreScriptPosition]=$jsFiles;
		}
	}
	public function renderHead(&$output)
	{
		$html='';
		foreach($this->metaTags as $meta)
			$html.=CHtml::metaTag($meta['content'],null,null,$meta)."\n";
		foreach($this->linkTags as $link)
			$html.=CHtml::linkTag(null,null,null,null,$link)."\n";
		foreach($this->cssFiles as $url=>$media)
			$html.=CHtml::cssFile($url,$media)."\n";
		foreach($this->css as $css)
			$html.=CHtml::css($css[0],$css[1])."\n";
		if($this->enableJavaScript)
		{
			if(isset($this->scriptFiles[self::POS_HEAD]))
			{
				foreach($this->scriptFiles[self::POS_HEAD] as $scriptFile)
					$html.=CHtml::scriptFile($scriptFile)."\n";
			}
			if(isset($this->scripts[self::POS_HEAD]))
				$html.=CHtml::script(implode("\n",$this->scripts[self::POS_HEAD]))."\n";
		}
		if($html!=='')
		{
			$count=0;

	$output=preg_replace('/(<title\b[^>]*>|<\\/head\s*>)/is','<###head###>$1',$output,1,$co
unt);
			if($count)
				$output=str_replace('<###head###>',$html,$output);
			else
```

```
                    $output=$html.$output;
            }
        }
        public function renderBodyBegin(&$output)
        {
            $html='';
            if(isset($this->scriptFiles[self::POS_BEGIN]))
            {
                foreach($this->scriptFiles[self::POS_BEGIN] as $scriptFile)
                    $html.=CHtml::scriptFile($scriptFile)."\n";
            }
            if(isset($this->scripts[self::POS_BEGIN]))
                $html.=CHtml::script(implode("\n",$this->scripts[self::POS_BEGIN]))."\n";
            if($html!=='')
            {
                $count=0;

        $output=preg_replace('/(<body\b[^>]*>)/is','$1<###begin###>',$output,1,$count);
                if($count)
                    $output=str_replace('<###begin###>',$html,$output);
                else
                    $output=$html.$output;
            }
        }
        public function renderBodyEnd(&$output)
        {
            if(!isset($this->scriptFiles[self::POS_END]) && !isset($this->scripts[self::POS_END])
                &&                                  !isset($this->scripts[self::POS_READY])
    && !isset($this->scripts[self::POS_LOAD]))
                return;
            $fullPage=0;
            $output=preg_replace('/(<\\/body\s*>)/is','<###end###>$1',$output,1,$fullPage);
            $html='';
            if(isset($this->scriptFiles[self::POS_END]))
            {
                foreach($this->scriptFiles[self::POS_END] as $scriptFile)
                    $html.=CHtml::scriptFile($scriptFile)."\n";
            }
            $scripts=isset($this->scripts[self::POS_END]) ? $this->scripts[self::POS_END] : array();
            if(isset($this->scripts[self::POS_READY]))
            {
                if($fullPage)
                    $scripts[]="jQuery(function($)
    {\n".implode("\n",$this->scripts[self::POS_READY])."\n});";
```

```
                    else
                        $scripts[]=implode("\n",$this->scripts[self::POS_READY]);
                }
                if(isset($this->scripts[self::POS_LOAD]))
                {
                    if($fullPage)
                        $scripts[]="jQuery(window).on('load',function()
{\n".implode("\n",$this->scripts[self::POS_LOAD])."\n});";
                    else
                        $scripts[]=implode("\n",$this->scripts[self::POS_LOAD]);
                }
                if(!empty($scripts))
                    $html.=CHtml::script(implode("\n",$scripts))."\n";
                if($fullPage)
                    $output=str_replace('<###end###>',$html,$output);
                else
                    $output=$output.$html;
            }
            public function getCoreScriptUrl()
            {
                if($this->_baseUrl!==null)
                    return $this->_baseUrl;
                else
                    return
$this->_baseUrl=Yii::app()->getAssetManager()->publish(YII_PATH.'/web/js/source');
            }
            public function setCoreScriptUrl($value)
            {
                $this->_baseUrl=$value;
            }
            public function getPackageBaseUrl($name)
            {
                if(!isset($this->coreScripts[$name]))
                    return false;
                $package=$this->coreScripts[$name];
                if(isset($package['baseUrl']))
                {
                    $baseUrl=$package['baseUrl'];
                    if($baseUrl==='' || $baseUrl[0]!=='/' && strpos($baseUrl,'://')===false)
                        $baseUrl=Yii::app()->getRequest()->getBaseUrl().'/'.$baseUrl;
                    $baseUrl=rtrim($baseUrl,'/');
                }
                elseif(isset($package['basePath']))
```

```
$baseUrl=Yii::app()->getAssetManager()->publish(Yii::getPathOfAlias($package['basePath']));
        else
                $baseUrl=$this->getCoreScriptUrl();
        return $this->coreScripts[$name]['baseUrl']=$baseUrl;
}
public function registerPackage($name)
{
        return $this->registerCoreScript($name);
}
public function registerCoreScript($name)
{
        if(isset($this->coreScripts[$name]))
                return $this;
        if(isset($this->packages[$name]))
                $package=$this->packages[$name];
        else
        {
                if($this->corePackages===null)
                        $this->corePackages=require(YII_PATH.'/web/js/packages.php');
                if(isset($this->corePackages[$name]))
                        $package=$this->corePackages[$name];
        }
        if(isset($package))
        {
                if(!empty($package['depends']))
                {
                        foreach($package['depends'] as $p)
                                $this->registerCoreScript($p);
                }
                $this->coreScripts[$name]=$package;
                $this->hasScripts=true;
                $params=func_get_args();
                $this->recordCachingAction('clientScript','registerCoreScript',$params);
        }
        return $this;
}
public function registerCssFile($url,$media='')
{
        $this->hasScripts=true;
        $this->cssFiles[$url]=$media;
        $params=func_get_args();
        $this->recordCachingAction('clientScript','registerCssFile',$params);
        return $this;
}
```

```
	public function registerCss($id,$css,$media='')
	{
		$this->hasScripts=true;
		$this->css[$id]=array($css,$media);
		$params=func_get_args();
		$this->recordCachingAction('clientScript','registerCss',$params);
		return $this;
	}
	public function registerScriptFile($url,$position=null)
	{
		if($position===null)
			$position=$this->defaultScriptFilePosition;
		$this->hasScripts=true;
		$this->scriptFiles[$position][$url]=$url;
		$params=func_get_args();
		$this->recordCachingAction('clientScript','registerScriptFile',$params);
		return $this;
	}
	public function registerScript($id,$script,$position=null)
	{
		if($position===null)
			$position=$this->defaultScriptPosition;
		$this->hasScripts=true;
		$this->scripts[$position][$id]=$script;
		if($position===self::POS_READY || $position===self::POS_LOAD)
			$this->registerCoreScript('jquery');
		$params=func_get_args();
		$this->recordCachingAction('clientScript','registerScript',$params);
		return $this;
	}
	public                                                                          function
registerMetaTag($content,$name=null,$httpEquiv=null,$options=array(),$id=null)
	{
		$this->hasScripts=true;
		if($name!==null)
			$options['name']=$name;
		if($httpEquiv!==null)
			$options['http-equiv']=$httpEquiv;
		$options['content']=$content;
		$this->metaTags[null===$id?count($this->metaTags):$id]=$options;
		$params=func_get_args();
		$this->recordCachingAction('clientScript','registerMetaTag',$params);
		return $this;
	}
	public                                                                          function
registerLinkTag($relation=null,$type=null,$href=null,$media=null,$options=array())
	{
		$this->hasScripts=true;
		if($relation!==null)
			$options['rel']=$relation;
		if($type!==null)
			$options['type']=$type;
		if($href!==null)
			$options['href']=$href;
		if($media!==null)
			$options['media']=$media;
		$this->linkTags[serialize($options)]=$options;
		$params=func_get_args();
		$this->recordCachingAction('clientScript','registerLinkTag',$params);
		return $this;
	}
	public function isCssFileRegistered($url)
	{
		return isset($this->cssFiles[$url]);
	}
	public function isCssRegistered($id)
	{
		return isset($this->css[$id]);
	}
	public function isScriptFileRegistered($url,$position=self::POS_HEAD)
	{
		return isset($this->scriptFiles[$position][$url]);
	}
	public function isScriptRegistered($id,$position=self::POS_READY)
	{
		return isset($this->scripts[$position][$id]);
	}
	protected function recordCachingAction($context,$method,$params)
	{
		if(($controller=Yii::app()->getController())!==null)
			$controller->recordCachingAction($context,$method,$params);
	}
	public function addPackage($name,$definition)
	{
		$this->packages[$name]=$definition;
		return $this;
	}
}
```

```
class CList extends CComponent implements IteratorAggregate,ArrayAccess,Countable
{
	private $_d=array();
	private $_c=0;
	private $_r=false;
	public function __construct($data=null,$readOnly=false)
	{
		if($data!==null)
			$this->copyFrom($data);
		$this->setReadOnly($readOnly);
	}
	public function getReadOnly()
	{
		return $this->_r;
	}
	protected function setReadOnly($value)
	{
		$this->_r=$value;
	}
	public function getIterator()
	{
		return new CListIterator($this->_d);
	}
	public function count()
	{
		return $this->getCount();
	}
	public function getCount()
	{
		return $this->_c;
	}
	public function itemAt($index)
	{
		if(isset($this->_d[$index]))
			return $this->_d[$index];
		elseif($index>=0 && $index<$this->_c) // in case the value is null
			return $this->_d[$index];
		else
			throw new CException(Yii::t('yii','List index "{index}" is out of bound.',
				array('{index}'=>$index)));
	}
	public function add($item)
	{
		$this->insertAt($this->_c,$item);
```

```
        return $this->_c-1;
}
public function insertAt($index,$item)
{
        if(!$this->_r)
        {
                if($index===$this->_c)
                        $this->_d[$this->_c++]=$item;
                elseif($index>=0 && $index<$this->_c)
                {
                        array_splice($this->_d,$index,0,array($item));
                        $this->_c++;
                }
                else
                        throw new CException(Yii::t('yii','List index "{index}" is out of bound.',
                                array('{index}'=>$index)));
        }
        else
                throw new CException(Yii::t('yii','The list is read only.'));
}
public function remove($item)
{
        if(($index=$this->indexOf($item))>=0)
        {
                $this->removeAt($index);
                return $index;
        }
        else
                return false;
}
public function removeAt($index)
{
        if(!$this->_r)
        {
                if($index>=0 && $index<$this->_c)
                {
                        $this->_c--;
                        if($index===$this->_c)
                                return array_pop($this->_d);
                        else
                        {
                                $item=$this->_d[$index];
                                array_splice($this->_d,$index,1);
                                return $item;
```

```
				}
			}
			else
				throw new CException(Yii::t('yii','List index "{index}" is out of bound.',
					array('{index}'=>$index)));
		}
		else
			throw new CException(Yii::t('yii','The list is read only.'));
	}
	public function clear()
	{
		for($i=$this->_c-1;$i>=0;--$i)
			$this->removeAt($i);
	}
	public function contains($item)
	{
		return $this->indexOf($item)>=0;
	}
	public function indexOf($item)
	{
		if(($index=array_search($item,$this->_d,true))!==false)
			return $index;
		else
			return -1;
	}
	public function toArray()
	{
		return $this->_d;
	}
	public function copyFrom($data)
	{
		if(is_array($data) || ($data instanceof Traversable))
		{
			if($this->_c>0)
				$this->clear();
			if($data instanceof CList)
				$data=$data->_d;
			foreach($data as $item)
				$this->add($item);
		}
		elseif($data!==null)
			throw new CException(Yii::t('yii','List data must be an array or an object
implementing Traversable.'));
	}
	public function mergeWith($data)
	{
		if(is_array($data) || ($data instanceof Traversable))
		{
			if($data instanceof CList)
				$data=$data->_d;
			foreach($data as $item)
				$this->add($item);
		}
		elseif($data!==null)
			throw new CException(Yii::t('yii','List data must be an array or an object 
implementing Traversable.'));
	}
	public function offsetExists($offset)
	{
		return ($offset>=0 && $offset<$this->_c);
	}
	public function offsetGet($offset)
	{
		return $this->itemAt($offset);
	}
	public function offsetSet($offset,$item)
	{
		if($offset===null || $offset===$this->_c)
			$this->insertAt($this->_c,$item);
		else
		{
			$this->removeAt($offset);
			$this->insertAt($offset,$item);
		}
	}
	public function offsetUnset($offset)
	{
		$this->removeAt($offset);
	}
}
class CFilterChain extends CList
{
	public $controller;
	public $action;
	public $filterIndex=0;
	public function __construct($controller,$action)
	{
		$this->controller=$controller;
```

```
        $this->action=$action;
    }
    public static function create($controller,$action,$filters)
    {
        $chain=new CFilterChain($controller,$action);
        $actionID=$action->getId();
        foreach($filters as $filter)
        {
            if(is_string($filter))    // filterName [+|- action1 action2]
            {
                if(($pos=strpos($filter,'+'))!==false || ($pos=strpos($filter,'-'))!==false)
                {
                    $matched=preg_match("/\b{$actionID}\b/i",substr($filter,$pos+1))>0;
                    if(($filter[$pos]==='+')===$matched)
                        $filter=CInlineFilter::create($controller,trim(substr($filter,0,$pos)));
                }
                else
                    $filter=CInlineFilter::create($controller,$filter);
            }
            elseif(is_array($filter))        //        array('path.to.class        [+|-        action1,
action2]','param1'=>'value1',...)
            {
                if(!isset($filter[0]))
                    throw    new    CException(Yii::t('yii','The    first    element    in    a    filter
configuration must be the filter class.'));
                $filterClass=$filter[0];
                unset($filter[0]);
                if(($pos=strpos($filterClass,'+'))!==false                                        ||
($pos=strpos($filterClass,'-'))!==false)
                {

    $matched=preg_match("/\b{$actionID}\b/i",substr($filterClass,$pos+1))>0;
                    if(($filterClass[$pos]==='+')===$matched)
                        $filterClass=trim(substr($filterClass,0,$pos));
                    else
                        continue;
                }
                $filter['class']=$filterClass;
                $filter=Yii::createComponent($filter);
            }
            if(is_object($filter))
            {
                $filter->init();
                $chain->add($filter);
            }
        }
        return $chain;
    }
    public function insertAt($index,$item)
    {
        if($item instanceof IFilter)
            parent::insertAt($index,$item);
        else
            throw new CException(Yii::t('yii','CFilterChain can only take objects implementing
the IFilter interface.'));
    }
    public function run()
    {
        if($this->offsetExists($this->filterIndex))
        {
            $filter=$this->itemAt($this->filterIndex++);
            $filter->filter($this);
        }
        else
            $this->controller->runAction($this->action);
    }
}
class CFilter extends CComponent implements IFilter
{
    public function filter($filterChain)
    {
        if($this->preFilter($filterChain))
        {
            $filterChain->run();
            $this->postFilter($filterChain);
        }
    }
    public function init()
    {
    }
    protected function preFilter($filterChain)
    {
        return true;
    }
    protected function postFilter($filterChain)
    {
    }
}
```

```
class CInlineFilter extends CFilter
{
        public $name;
        public static function create($controller,$filterName)
        {
                if(method_exists($controller,'filter'.$filterName))
                {
                        $filter=new CInlineFilter;
                        $filter->name=$filterName;
                        return $filter;
                }
                else
                        throw new CException(Yii::t('yii','Filter "{filter}" is invalid. Controller "{class}" does
not have the filter method "filter{filter}".',
                                array('{filter}'=>$filterName, '{class}'=>get_class($controller))));
        }
        public function filter($filterChain)
        {
                $method='filter'.$this->name;
                $filterChain->controller->$method($filterChain);
        }
}
class CAccessControlFilter extends CFilter
{
        public $message;
        private $_rules=array();
        public function getRules()
        {
                return $this->_rules;
        }
        public function setRules($rules)
        {
                foreach($rules as $rule)
                {
                        if(is_array($rule) && isset($rule[0]))
                        {
                                $r=new CAccessRule;
                                $r->allow=$rule[0]==='allow';
                                foreach(array_slice($rule,1) as $name=>$value)
                                {
                                        if($name==='expression' || $name==='roles' || $name==='message' ||
$name==='deniedCallback')
                                                $r->$name=$value;
                                        else
                        $r->$name=array_map('strtolower',$value);
                    }
                    $this->_rules[]=$r;
                }
            }
        }
        protected function preFilter($filterChain)
        {
            $app=Yii::app();
            $request=$app->getRequest();
            $user=$app->getUser();
            $verb=$request->getRequestType();
            $ip=$request->getUserHostAddress();
            foreach($this->getRules() as $rule)
            {

        if(($allow=$rule->isUserAllowed($user,$filterChain->controller,$filterChain->action,$ip,$ver
    b))>0) // allowed
                    break;
                elseif($allow<0) // denied
                {
                    if(isset($rule->deniedCallback))
                        call_user_func($rule->deniedCallback, $rule);
                    else
                        $this->accessDenied($user,$this->resolveErrorMessage($rule));
                    return false;
                }
            }
            return true;
        }
        protected function resolveErrorMessage($rule)
        {
            if($rule->message!==null)
                return $rule->message;
            elseif($this->message!==null)
                return $this->message;
            else
                return Yii::t('yii','You are not authorized to perform this action.');
        }
        protected function accessDenied($user,$message)
        {
            if($user->getIsGuest())
                $user->loginRequired();
            else
```

```
                throw new CHttpException(403,$message);
        }
}
class CAccessRule extends CComponent
{
        public $allow;
        public $actions;
        public $controllers;
        public $users;
        public $roles;
        public $ips;
        public $verbs;
        public $expression;
        public $message;
        public $deniedCallback;
        public function isUserAllowed($user,$controller,$action,$ip,$verb)
        {
                if($this->isActionMatched($action)
                        && $this->isUserMatched($user)
                        && $this->isRoleMatched($user)
                        && $this->isIpMatched($ip)
                        && $this->isVerbMatched($verb)
                        && $this->isControllerMatched($controller)
                        && $this->isExpressionMatched($user))
                        return $this->allow ? 1 : -1;
                else
                        return 0;
        }
        protected function isActionMatched($action)
        {
                return empty($this->actions) || in_array(strtolower($action->getId()),$this->actions);
        }
        protected function isControllerMatched($controller)
        {
                return                  empty($this->controllers)                  ||
in_array(strtolower($controller->getId()),$this->controllers);
        }
        protected function isUserMatched($user)
        {
                if(empty($this->users))
                        return true;
                foreach($this->users as $u)
                {
                        if($u==='*')
                return true;
            elseif($u==='?' && $user->getIsGuest())
                return true;
            elseif($u==='@' && !$user->getIsGuest())
                return true;
            elseif(!strcasecmp($u,$user->getName()))
                return true;
        }
        return false;
    }
    protected function isRoleMatched($user)
    {
        if(empty($this->roles))
            return true;
        foreach($this->roles as $key=>$role)
        {
            if(is_numeric($key))
            {
                if($user->checkAccess($role))
                    return true;
            }
            else
            {
                if($user->checkAccess($key,$role))
                    return true;
            }
        }
        return false;
    }
    protected function isIpMatched($ip)
    {
        if(empty($this->ips))
            return true;
        foreach($this->ips as $rule)
        {
            if($rule==='*'        ||        $rule===$ip        ||        (($pos=strpos($rule,'*'))!==false
&& !strncmp($ip,$rule,$pos)))
                return true;
        }
        return false;
    }
    protected function isVerbMatched($verb)
    {
        return empty($this->verbs) || in_array(strtolower($verb),$this->verbs);
```

```
	}
	protected function isExpressionMatched($user)
	{
		if($this->expression===null)
			return true;
		else
			return $this->evaluateExpression($this->expression, array('user'=>$user));
	}
}
abstract class CModel extends CComponent implements IteratorAggregate, ArrayAccess
{
	private $_errors=array();// attribute name => array of errors
	private $_validators;		// validators
	private $_scenario='';	// scenario
	abstract public function attributeNames();
	public function rules()
	{
		return array();
	}
	public function behaviors()
	{
		return array();
	}
	public function attributeLabels()
	{
		return array();
	}
	public function validate($attributes=null, $clearErrors=true)
	{
		if($clearErrors)
			$this->clearErrors();
		if($this->beforeValidate())
		{
			foreach($this->getValidators() as $validator)
				$validator->validate($this,$attributes);
			$this->afterValidate();
			return !$this->hasErrors();
		}
		else
			return false;
	}
	protected function afterConstruct()
	{
		if($this->hasEventHandler('onAfterConstruct'))
		$this->onAfterConstruct(new CEvent($this));
}
protected function beforeValidate()
{
	$event=new CModelEvent($this);
	$this->onBeforeValidate($event);
	return $event->isValid;
}
protected function afterValidate()
{
	$this->onAfterValidate(new CEvent($this));
}
public function onAfterConstruct($event)
{
	$this->raiseEvent('onAfterConstruct',$event);
}
public function onBeforeValidate($event)
{
	$this->raiseEvent('onBeforeValidate',$event);
}
public function onAfterValidate($event)
{
	$this->raiseEvent('onAfterValidate',$event);
}
public function getValidatorList()
{
	if($this->_validators===null)
		$this->_validators=$this->createValidators();
	return $this->_validators;
}
public function getValidators($attribute=null)
{
	if($this->_validators===null)
		$this->_validators=$this->createValidators();
	$validators=array();
	$scenario=$this->getScenario();
	foreach($this->_validators as $validator)
	{
		if($validator->applyTo($scenario))
		{
			if($attribute===null || in_array($attribute,$validator->attributes,true))
				$validators[]=$validator;
		}
	}
```

```
		return $validators;
	}
	public function createValidators()
	{
		$validators=new CList;
		foreach($this->rules() as $rule)
		{
			if(isset($rule[0],$rule[1]))    // attributes, validator name

	$validators->add(CValidator::createValidator($rule[1],$this,$rule[0],array_slice($rule,2)));
			else
				throw new CException(Yii::t('yii','{class} has an invalid validation rule. The rule
must specify attributes to be validated and the validator name.',
					array('{class}'=>get_class($this))));
		}
		return $validators;
	}
	public function isAttributeRequired($attribute)
	{
		foreach($this->getValidators($attribute) as $validator)
		{
			if($validator instanceof CRequiredValidator)
				return true;
		}
		return false;
	}
	public function isAttributeSafe($attribute)
	{
		$attributes=$this->getSafeAttributeNames();
		return in_array($attribute,$attributes);
	}
	public function getAttributeLabel($attribute)
	{
		$labels=$this->attributeLabels();
		if(isset($labels[$attribute]))
			return $labels[$attribute];
		else
			return $this->generateAttributeLabel($attribute);
	}
	public function hasErrors($attribute=null)
	{
		if($attribute===null)
			return $this->_errors!==array();
		else
```

```
                return isset($this->_errors[$attribute]);
        }
        public function getErrors($attribute=null)
        {
                if($attribute===null)
                        return $this->_errors;
                else
                        return isset($this->_errors[$attribute]) ? $this->_errors[$attribute] : array();
        }
        public function getError($attribute)
        {
                return isset($this->_errors[$attribute]) ? reset($this->_errors[$attribute]) : null;
        }
        public function addError($attribute,$error)
        {
                $this->_errors[$attribute][]=$error;
        }
        public function addErrors($errors)
        {
                foreach($errors as $attribute=>$error)
                {
                        if(is_array($error))
                        {
                                foreach($error as $e)
                                        $this->addError($attribute, $e);
                        }
                        else
                                $this->addError($attribute, $error);
                }
        }
        public function clearErrors($attribute=null)
        {
                if($attribute===null)
                        $this->_errors=array();
                else
                        unset($this->_errors[$attribute]);
        }
        public function generateAttributeLabel($name)
        {
                return                                                  ucwords(trim(strtolower(str_replace(array('-','_','.'),'
',preg_replace('/(?<![A-Z])[A-Z]/', ' \0', $name)))));
        }
        public function getAttributes($names=null)
        {
```

```
        $values=array();
        foreach($this->attributeNames() as $name)
            $values[$name]=$this->$name;
        if(is_array($names))
        {
            $values2=array();
            foreach($names as $name)
                $values2[$name]=isset($values[$name]) ? $values[$name] : null;
            return $values2;
        }
        else
            return $values;
    }
    public function setAttributes($values,$safeOnly=true)
    {
        if(!is_array($values))
            return;
        $attributes=array_flip($safeOnly          ?          $this->getSafeAttributeNames()          :
$this->attributeNames());
        foreach($values as $name=>$value)
        {
            if(isset($attributes[$name]))
                $this->$name=$value;
            elseif($safeOnly)
                $this->onUnsafeAttribute($name,$value);
        }
    }
    public function unsetAttributes($names=null)
    {
        if($names===null)
            $names=$this->attributeNames();
        foreach($names as $name)
            $this->$name=null;
    }
    public function onUnsafeAttribute($name,$value)
    {
        if(YII_DEBUG)
            Yii::log(Yii::t('yii','Failed      to      set      unsafe      attribute      "{attribute}"      of
"{class}".',array('{attribute}'=>$name, '{class}'=>get_class($this))),CLogger::LEVEL_WARNING);
    }
    public function getScenario()
    {
        return $this->_scenario;
    }
```

```
public function setScenario($value)
{
        $this->_scenario=$value;
}
public function getSafeAttributeNames()
{
        $attributes=array();
        $unsafe=array();
        foreach($this->getValidators() as $validator)
        {
                if(!$validator->safe)
                {
                        foreach($validator->attributes as $name)
                                $unsafe[]=$name;
                }
                else
                {
                        foreach($validator->attributes as $name)
                                $attributes[$name]=true;
                }
        }
        foreach($unsafe as $name)
                unset($attributes[$name]);
        return array_keys($attributes);
}
public function getIterator()
{
        $attributes=$this->getAttributes();
        return new CMapIterator($attributes);
}
public function offsetExists($offset)
{
        return property_exists($this,$offset);
}
public function offsetGet($offset)
{
        return $this->$offset;
}
public function offsetSet($offset,$item)
{
        $this->$offset=$item;
}
public function offsetUnset($offset)
{
        unset($this->$offset);
    }
}
abstract class CActiveRecord extends CModel
{
    const BELONGS_TO='CBelongsToRelation';
    const HAS_ONE='CHasOneRelation';
    const HAS_MANY='CHasManyRelation';
    const MANY_MANY='CManyManyRelation';
    const STAT='CStatRelation';
    public static $db;
    private static $_models=array();            // class name => model
    private $_md;                                   // meta data
    private $_new=false;                            // whether this instance is new or not
    private $_attributes=array();               // attribute name => attribute value
    private $_related=array();                      // attribute name => related objects
    private $_c;                                    // query criteria (used by finder only)
    private $_pk;                                   // old primary key value
    private $_alias='t';                        // the table alias being used for query
    public function __construct($scenario='insert')
    {
        if($scenario===null) // internally used by populateRecord() and model()
            return;
        $this->setScenario($scenario);
        $this->setIsNewRecord(true);
        $this->_attributes=$this->getMetaData()->attributeDefaults;
        $this->init();
        $this->attachBehaviors($this->behaviors());
        $this->afterConstruct();
    }
    public function init()
    {
    }
    public function cache($duration, $dependency=null, $queryCount=1)
    {
        $this->getDbConnection()->cache($duration, $dependency, $queryCount);
        return $this;
    }
    public function __sleep()
    {
        $this->_md=null;
        return array_keys((array)$this);
    }
    public function __get($name)
```

```
{
        if(isset($this->_attributes[$name]))
                return $this->_attributes[$name];
        elseif(isset($this->getMetaData()->columns[$name]))
                return null;
        elseif(isset($this->_related[$name]))
                return $this->_related[$name];
        elseif(isset($this->getMetaData()->relations[$name]))
                return $this->getRelated($name);
        else
                return parent::__get($name);
}
public function __set($name,$value)
{
        if($this->setAttribute($name,$value)===false)
        {
                if(isset($this->getMetaData()->relations[$name]))
                        $this->_related[$name]=$value;
                else
                        parent::__set($name,$value);
        }
}
public function __isset($name)
{
        if(isset($this->_attributes[$name]))
                return true;
        elseif(isset($this->getMetaData()->columns[$name]))
                return false;
        elseif(isset($this->_related[$name]))
                return true;
        elseif(isset($this->getMetaData()->relations[$name]))
                return $this->getRelated($name)!==null;
        else
                return parent::__isset($name);
}
public function __unset($name)
{
        if(isset($this->getMetaData()->columns[$name]))
                unset($this->_attributes[$name]);
        elseif(isset($this->getMetaData()->relations[$name]))
                unset($this->_related[$name]);
        else
                parent::__unset($name);
}
```

```
	public function __call($name,$parameters)
	{
		if(isset($this->getMetaData()->relations[$name]))
		{
			if(empty($parameters))
				return $this->getRelated($name,false);
			else
				return $this->getRelated($name,false,$parameters[0]);
		}
		$scopes=$this->scopes();
		if(isset($scopes[$name]))
		{
			$this->getDbCriteria()->mergeWith($scopes[$name]);
			return $this;
		}
		return parent::__call($name,$parameters);
	}
	public function getRelated($name,$refresh=false,$params=array())
	{
		if(!$refresh     &&     $params===array()     &&     (isset($this->_related[$name])     ||
array_key_exists($name,$this->_related)))
			return $this->_related[$name];
		$md=$this->getMetaData();
		if(!isset($md->relations[$name]))
			throw new CDbException(Yii::t('yii','{class} does not have relation "{name}".',
				array('{class}'=>get_class($this), '{name}'=>$name)));
		$relation=$md->relations[$name];
		if($this->getIsNewRecord() && !$refresh && ($relation instanceof CHasOneRelation ||
$relation instanceof CHasManyRelation))
			return $relation instanceof CHasOneRelation ? null : array();
		if($params!==array()) // dynamic query
		{
			$exists=isset($this->_related[$name]) || array_key_exists($name,$this->_related);
			if($exists)
				$save=$this->_related[$name];
			if($params instanceof CDbCriteria)
				$params = $params->toArray();
			$r=array($name=>$params);
		}
		else
			$r=$name;
		unset($this->_related[$name]);
		$finder=new CActiveFinder($this,$r);
		$finder->lazyFind($this);
```

```
        if(!isset($this->_related[$name]))
        {
                if($relation instanceof CHasManyRelation)
                        $this->_related[$name]=array();
                elseif($relation instanceof CStatRelation)
                        $this->_related[$name]=$relation->defaultValue;
                else
                        $this->_related[$name]=null;
        }
        if($params!==array())
        {
                $results=$this->_related[$name];
                if($exists)
                        $this->_related[$name]=$save;
                else
                        unset($this->_related[$name]);
                return $results;
        }
        else
                return $this->_related[$name];
}
public function hasRelated($name)
{
        return isset($this->_related[$name]) || array_key_exists($name,$this->_related);
}
public function getDbCriteria($createIfNull=true)
{
        if($this->_c===null)
        {
                if(($c=$this->defaultScope())!==array() || $createIfNull)
                        $this->_c=new CDbCriteria($c);
        }
        return $this->_c;
}
public function setDbCriteria($criteria)
{
        $this->_c=$criteria;
}
public function defaultScope()
{
        return array();
}
public function resetScope($resetDefault=true)
{
	if($resetDefault)
		$this->_c=new CDbCriteria();
	else
		$this->_c=null;
	return $this;
}
public static function model($className=__CLASS__)
{
	if(isset(self::$_models[$className]))
		return self::$_models[$className];
	else
	{
		$model=self::$_models[$className]=new $className(null);
		$model->_md=new CActiveRecordMetaData($model);
		$model->attachBehaviors($model->behaviors());
		return $model;
	}
}
public function getMetaData()
{
	if($this->_md!==null)
		return $this->_md;
	else
		return $this->_md=self::model(get_class($this))->_md;
}
public function refreshMetaData()
{
	$finder=self::model(get_class($this));
	$finder->_md=new CActiveRecordMetaData($finder);
	if($this!==$finder)
		$this->_md=$finder->_md;
}
public function tableName()
{
	return get_class($this);
}
public function primaryKey()
{
}
public function relations()
{
	return array();
}
public function scopes()
```

```
	{
		return array();
	}
	public function attributeNames()
	{
		return array_keys($this->getMetaData()->columns);
	}
	public function getAttributeLabel($attribute)
	{
		$labels=$this->attributeLabels();
		if(isset($labels[$attribute]))
			return $labels[$attribute];
		elseif(strpos($attribute,'.')!==false)
		{
			$segs=explode('.',$attribute);
			$name=array_pop($segs);
			$model=$this;
			foreach($segs as $seg)
			{
				$relations=$model->getMetaData()->relations;
				if(isset($relations[$seg]))
					$model=CActiveRecord::model($relations[$seg]->className);
				else
					break;
			}
			return $model->getAttributeLabel($name);
		}
		else
			return $this->generateAttributeLabel($attribute);
	}
	public function getDbConnection()
	{
		if(self::$db!==null)
			return self::$db;
		else
		{
			self::$db=Yii::app()->getDb();
			if(self::$db instanceof CDbConnection)
				return self::$db;
			else
				throw new CDbException(Yii::t('yii','Active Record requires a "db"
CDbConnection application component.'));
		}
	}
	public function getActiveRelation($name)
	{
		return			isset($this->getMetaData()->relations[$name])			?
$this->getMetaData()->relations[$name] : null;
	}
	public function getTableSchema()
	{
		return $this->getMetaData()->tableSchema;
	}
	public function getCommandBuilder()
	{
		return $this->getDbConnection()->getSchema()->getCommandBuilder();
	}
	public function hasAttribute($name)
	{
		return isset($this->getMetaData()->columns[$name]);
	}
	public function getAttribute($name)
	{
		if(property_exists($this,$name))
			return $this->$name;
		elseif(isset($this->_attributes[$name]))
			return $this->_attributes[$name];
	}
	public function setAttribute($name,$value)
	{
		if(property_exists($this,$name))
			$this->$name=$value;
		elseif(isset($this->getMetaData()->columns[$name]))
			$this->_attributes[$name]=$value;
		else
			return false;
		return true;
	}
	public function addRelatedRecord($name,$record,$index)
	{
		if($index!==false)
		{
			if(!isset($this->_related[$name]))
				$this->_related[$name]=array();
			if($record instanceof CActiveRecord)
			{
				if($index===true)
					$this->_related[$name][]=$record;
```

```
                else
                    $this->_related[$name][$index]=$record;
            }
        }
        elseif(!isset($this->_related[$name]))
            $this->_related[$name]=$record;
    }
    public function getAttributes($names=true)
    {
        $attributes=$this->_attributes;
        foreach($this->getMetaData()->columns as $name=>$column)
        {
            if(property_exists($this,$name))
                $attributes[$name]=$this->$name;
            elseif($names===true && !isset($attributes[$name]))
                $attributes[$name]=null;
        }
        if(is_array($names))
        {
            $attrs=array();
            foreach($names as $name)
            {
                if(property_exists($this,$name))
                    $attrs[$name]=$this->$name;
                else
                    $attrs[$name]=isset($attributes[$name])?$attributes[$name]:null;
            }
            return $attrs;
        }
        else
            return $attributes;
    }
    public function save($runValidation=true,$attributes=null)
    {
        if(!$runValidation || $this->validate($attributes))
            return        $this->getIsNewRecord()        ?        $this->insert($attributes)        :
$this->update($attributes);
        else
            return false;
    }
    public function getIsNewRecord()
    {
        return $this->_new;
    }
```

```
public function setIsNewRecord($value)
{
        $this->_new=$value;
}
public function onBeforeSave($event)
{
        $this->raiseEvent('onBeforeSave',$event);
}
public function onAfterSave($event)
{
        $this->raiseEvent('onAfterSave',$event);
}
public function onBeforeDelete($event)
{
        $this->raiseEvent('onBeforeDelete',$event);
}
public function onAfterDelete($event)
{
        $this->raiseEvent('onAfterDelete',$event);
}
public function onBeforeFind($event)
{
        $this->raiseEvent('onBeforeFind',$event);
}
public function onAfterFind($event)
{
        $this->raiseEvent('onAfterFind',$event);
}
protected function beforeSave()
{
        if($this->hasEventHandler('onBeforeSave'))
        {
                $event=new CModelEvent($this);
                $this->onBeforeSave($event);
                return $event->isValid;
        }
        else
                return true;
}
protected function afterSave()
{
        if($this->hasEventHandler('onAfterSave'))
                $this->onAfterSave(new CEvent($this));
}
```

```
	protected function beforeDelete()
	{
		if($this->hasEventHandler('onBeforeDelete'))
		{
			$event=new CModelEvent($this);
			$this->onBeforeDelete($event);
			return $event->isValid;
		}
		else
			return true;
	}
	protected function afterDelete()
	{
		if($this->hasEventHandler('onAfterDelete'))
			$this->onAfterDelete(new CEvent($this));
	}
	protected function beforeFind()
	{
		if($this->hasEventHandler('onBeforeFind'))
		{
			$event=new CModelEvent($this);
			$this->onBeforeFind($event);
		}
	}
	protected function afterFind()
	{
		if($this->hasEventHandler('onAfterFind'))
			$this->onAfterFind(new CEvent($this));
	}
	public function beforeFindInternal()
	{
		$this->beforeFind();
	}
	public function afterFindInternal()
	{
		$this->afterFind();
	}
	public function insert($attributes=null)
	{
		if(!$this->getIsNewRecord())
			throw new CDbException(Yii::t('yii','The active record cannot be inserted to
database because it is not new.'));
		if($this->beforeSave())
		{
```

```
                $builder=$this->getCommandBuilder();
                $table=$this->getMetaData()->tableSchema;

        $command=$builder->createInsertCommand($table,$this->getAttributes($attributes));
                if($command->execute())
                {
                    $primaryKey=$table->primaryKey;
                    if($table->sequenceName!==null)
                    {
                        if(is_string($primaryKey) && $this->$primaryKey===null)
                            $this->$primaryKey=$builder->getLastInsertID($table);
                        elseif(is_array($primaryKey))
                        {
                            foreach($primaryKey as $pk)
                            {
                                if($this->$pk===null)
                                {
                                    $this->$pk=$builder->getLastInsertID($table);
                                    break;
                                }
                            }
                        }
                    }
                    $this->_pk=$this->getPrimaryKey();
                    $this->afterSave();
                    $this->setIsNewRecord(false);
                    $this->setScenario('update');
                    return true;
                }
            }
            return false;
        }
        public function update($attributes=null)
        {
            if($this->getIsNewRecord())
                throw new CDbException(Yii::t('yii','The active record cannot be updated because
    it is new.'));
            if($this->beforeSave())
            {
                if($this->_pk===null)
                    $this->_pk=$this->getPrimaryKey();
                $this->updateByPk($this->getOldPrimaryKey(),$this->getAttributes($attributes));
                $this->_pk=$this->getPrimaryKey();
                $this->afterSave();
				return true;
			}
			else
				return false;
		}
		public function saveAttributes($attributes)
		{
			if(!$this->getIsNewRecord())
			{
				$values=array();
				foreach($attributes as $name=>$value)
				{
					if(is_integer($name))
						$values[$value]=$this->$value;
					else
						$values[$name]=$this->$name=$value;
				}
				if($this->_pk===null)
					$this->_pk=$this->getPrimaryKey();
				if($this->updateByPk($this->getOldPrimaryKey(),$values)>0)
				{
					$this->_pk=$this->getPrimaryKey();
					return true;
				}
				else
					return false;
			}
			else
				throw new CDbException(Yii::t('yii','The active record cannot be updated because
	it is new.'));
		}
		public function saveCounters($counters)
		{
			$builder=$this->getCommandBuilder();
			$table=$this->getTableSchema();
			$criteria=$builder->createPkCriteria($table,$this->getOldPrimaryKey());

		$command=$builder->createUpdateCounterCommand($this->getTableSchema(),$counters,
	$criteria);
			if($command->execute())
			{
				foreach($counters as $name=>$value)
					$this->$name=$this->$name+$value;
				return true;
		}
		else
			return false;
	}
	public function delete()
	{
		if(!$this->getIsNewRecord())
		{
			if($this->beforeDelete())
			{
				$result=$this->deleteByPk($this->getPrimaryKey())>0;
				$this->afterDelete();
				return $result;
			}
			else
				return false;
		}
		else
			throw new CDbException(Yii::t('yii','The active record cannot be deleted because it
is new.'));
	}
	public function refresh()
	{
		if(($record=$this->findByPk($this->getPrimaryKey()))!==null)
		{
			$this->_attributes=array();
			$this->_related=array();
			foreach($this->getMetaData()->columns as $name=>$column)
			{
				if(property_exists($this,$name))
					$this->$name=$record->$name;
				else
					$this->_attributes[$name]=$record->$name;
			}
			return true;
		}
		else
			return false;
	}
	public function equals($record)
	{
		return			$this->tableName()===$record->tableName()			&&
$this->getPrimaryKey()===$record->getPrimaryKey();
	}
```

```
public function getPrimaryKey()
{
        $table=$this->getMetaData()->tableSchema;
        if(is_string($table->primaryKey))
                return $this->{$table->primaryKey};
        elseif(is_array($table->primaryKey))
        {
                $values=array();
                foreach($table->primaryKey as $name)
                        $values[$name]=$this->$name;
                return $values;
        }
        else
                return null;
}
public function setPrimaryKey($value)
{
        $this->_pk=$this->getPrimaryKey();
        $table=$this->getMetaData()->tableSchema;
        if(is_string($table->primaryKey))
                $this->{$table->primaryKey}=$value;
        elseif(is_array($table->primaryKey))
        {
                foreach($table->primaryKey as $name)
                        $this->$name=$value[$name];
        }
}
public function getOldPrimaryKey()
{
        return $this->_pk;
}
public function setOldPrimaryKey($value)
{
        $this->_pk=$value;
}
protected function query($criteria,$all=false)
{
        $this->beforeFind();
        $this->applyScopes($criteria);
        if(empty($criteria->with))
        {
                if(!$all)
                        $criteria->limit=1;
```

```
		$command=$this->getCommandBuilder()->createFindCommand($this->getTableSchema(),$
criteria);
			return $all ? $this->populateRecords($command->queryAll(), true,
$criteria->index) : $this->populateRecord($command->queryRow());
		}
		else
		{
			$finder=new CActiveFinder($this,$criteria->with);
			return $finder->query($criteria,$all);
		}
	}
	public function applyScopes(&$criteria)
	{
		if(!empty($criteria->scopes))
		{
			$scs=$this->scopes();
			$c=$this->getDbCriteria();
			foreach((array)$criteria->scopes as $k=>$v)
			{
				if(is_integer($k))
				{
					if(is_string($v))
					{
						if(isset($scs[$v]))
						{
							$c->mergeWith($scs[$v],true);
							continue;
						}
						$scope=$v;
						$params=array();
					}
					elseif(is_array($v))
					{
						$scope=key($v);
						$params=current($v);
					}
				}
				elseif(is_string($k))
				{
					$scope=$k;
					$params=$v;
				}
				call_user_func_array(array($this,$scope),(array)$params);
			}
```

```
        }
        if(isset($c) || ($c=$this->getDbCriteria(false))!==null)
        {
            $c->mergeWith($criteria);
            $criteria=$c;
            $this->resetScope(false);
        }
    }
    public function getTableAlias($quote=false, $checkScopes=true)
    {
        if($checkScopes        &&        ($criteria=$this->getDbCriteria(false))!==null        &&
$criteria->alias!='')
            $alias=$criteria->alias;
        else
            $alias=$this->_alias;
        return  $quote  ?  $this->getDbConnection()->getSchema()->quoteTableName($alias)  :
$alias;
    }
    public function setTableAlias($alias)
    {
        $this->_alias=$alias;
    }
    public function find($condition='',$params=array())
    {
        $criteria=$this->getCommandBuilder()->createCriteria($condition,$params);
        return $this->query($criteria);
    }
    public function findAll($condition='',$params=array())
    {
        $criteria=$this->getCommandBuilder()->createCriteria($condition,$params);
        return $this->query($criteria,true);
    }
    public function findByPk($pk,$condition='',$params=array())
    {
        $prefix=$this->getTableAlias(true).'.';

    $criteria=$this->getCommandBuilder()->createPkCriteria($this->getTableSchema(),$pk,$con
dition,$params,$prefix);
        return $this->query($criteria);
    }
    public function findAllByPk($pk,$condition='',$params=array())
    {
        $prefix=$this->getTableAlias(true).'.';
```

```
	$criteria=$this->getCommandBuilder()->createPkCriteria($this->getTableSchema(),$pk,$con
dition,$params,$prefix);
		return $this->query($criteria,true);
	}
	public function findByAttributes($attributes,$condition='',$params=array())
	{
		$prefix=$this->getTableAlias(true).'.';

	$criteria=$this->getCommandBuilder()->createColumnCriteria($this->getTableSchema(),$att
ributes,$condition,$params,$prefix);
		return $this->query($criteria);
	}
	public function findAllByAttributes($attributes,$condition='',$params=array())
	{
		$prefix=$this->getTableAlias(true).'.';

	$criteria=$this->getCommandBuilder()->createColumnCriteria($this->getTableSchema(),$att
ributes,$condition,$params,$prefix);
		return $this->query($criteria,true);
	}
	public function findBySql($sql,$params=array())
	{
		$this->beforeFind();
		if(($criteria=$this->getDbCriteria(false))!==null && !empty($criteria->with))
		{
			$this->resetScope(false);
			$finder=new CActiveFinder($this,$criteria->with);
			return $finder->findBySql($sql,$params);
		}
		else
		{
			$command=$this->getCommandBuilder()->createSqlCommand($sql,$params);
			return $this->populateRecord($command->queryRow());
		}
	}
	public function findAllBySql($sql,$params=array())
	{
		$this->beforeFind();
		if(($criteria=$this->getDbCriteria(false))!==null && !empty($criteria->with))
		{
			$this->resetScope(false);
			$finder=new CActiveFinder($this,$criteria->with);
			return $finder->findAllBySql($sql,$params);
		}
		else
		{
			$command=$this->getCommandBuilder()->createSqlCommand($sql,$params);
			return $this->populateRecords($command->queryAll());
		}
	}
	public function count($condition='',$params=array())
	{
		$builder=$this->getCommandBuilder();
		$criteria=$builder->createCriteria($condition,$params);
		$this->applyScopes($criteria);
		if(empty($criteria->with))
			return
$builder->createCountCommand($this->getTableSchema(),$criteria)->queryScalar();
		else
		{
			$finder=new CActiveFinder($this,$criteria->with);
			return $finder->count($criteria);
		}
	}
	public function countByAttributes($attributes,$condition='',$params=array())
	{
		$prefix=$this->getTableAlias(true).'.';
		$builder=$this->getCommandBuilder();

	$criteria=$builder->createColumnCriteria($this->getTableSchema(),$attributes,$condition,$
params,$prefix);
		$this->applyScopes($criteria);
		if(empty($criteria->with))
			return
$builder->createCountCommand($this->getTableSchema(),$criteria)->queryScalar();
		else
		{
			$finder=new CActiveFinder($this,$criteria->with);
			return $finder->count($criteria);
		}
	}
	public function countBySql($sql,$params=array())
	{
		return
$this->getCommandBuilder()->createSqlCommand($sql,$params)->queryScalar();
	}
	public function exists($condition='',$params=array())
	{
        $builder=$this->getCommandBuilder();
        $criteria=$builder->createCriteria($condition,$params);
        $table=$this->getTableSchema();
        $criteria->select='1';
        $criteria->limit=1;
        $this->applyScopes($criteria);
        if(empty($criteria->with))
            return $builder->createFindCommand($table,$criteria)->queryRow()!==false;
        else
        {
            $criteria->select='*';
            $finder=new CActiveFinder($this,$criteria->with);
            return $finder->count($criteria)>0;
        }
    }
    public function with()
    {
        if(func_num_args()>0)
        {
            $with=func_get_args();
            if(is_array($with[0]))    // the parameter is given as an array
                $with=$with[0];
            if(!empty($with))
                $this->getDbCriteria()->mergeWith(array('with'=>$with));
        }
        return $this;
    }
    public function together()
    {
        $this->getDbCriteria()->together=true;
        return $this;
    }
    public function updateByPk($pk,$attributes,$condition='',$params=array())
    {
        $builder=$this->getCommandBuilder();
        $table=$this->getTableSchema();
        $criteria=$builder->createPkCriteria($table,$pk,$condition,$params);
        $command=$builder->createUpdateCommand($table,$attributes,$criteria);
        return $command->execute();
    }
    public function updateAll($attributes,$condition='',$params=array())
    {
        $builder=$this->getCommandBuilder();
        $criteria=$builder->createCriteria($condition,$params);
        $command=$builder->createUpdateCommand($this->getTableSchema(),$attributes,$criteri
a);
            return $command->execute();
        }
        public function updateCounters($counters,$condition='',$params=array())
        {
            $builder=$this->getCommandBuilder();
            $criteria=$builder->createCriteria($condition,$params);

        $command=$builder->createUpdateCounterCommand($this->getTableSchema(),$counters,
$criteria);
            return $command->execute();
        }
        public function deleteByPk($pk,$condition='',$params=array())
        {
            $builder=$this->getCommandBuilder();

        $criteria=$builder->createPkCriteria($this->getTableSchema(),$pk,$condition,$params);
            $command=$builder->createDeleteCommand($this->getTableSchema(),$criteria);
            return $command->execute();
        }
        public function deleteAll($condition='',$params=array())
        {
            $builder=$this->getCommandBuilder();
            $criteria=$builder->createCriteria($condition,$params);
            $command=$builder->createDeleteCommand($this->getTableSchema(),$criteria);
            return $command->execute();
        }
        public function deleteAllByAttributes($attributes,$condition='',$params=array())
        {
            $builder=$this->getCommandBuilder();
            $table=$this->getTableSchema();
            $criteria=$builder->createColumnCriteria($table,$attributes,$condition,$params);
            $command=$builder->createDeleteCommand($table,$criteria);
            return $command->execute();
        }
        public function populateRecord($attributes,$callAfterFind=true)
        {
            if($attributes!==false)
            {
                $record=$this->instantiate($attributes);
                $record->setScenario('update');
                $record->init();
                $md=$record->getMetaData();
                foreach($attributes as $name=>$value)
                {
                    if(property_exists($record,$name))
                        $record->$name=$value;
                    elseif(isset($md->columns[$name]))
                        $record->_attributes[$name]=$value;
                }
                $record->_pk=$record->getPrimaryKey();
                $record->attachBehaviors($record->behaviors());
                if($callAfterFind)
                    $record->afterFind();
                return $record;
            }
            else
                return null;
        }
        public function populateRecords($data,$callAfterFind=true,$index=null)
        {
            $records=array();
            foreach($data as $attributes)
            {
                if(($record=$this->populateRecord($attributes,$callAfterFind))!==null)
                {
                    if($index===null)
                        $records[]=$record;
                    else
                        $records[$record->$index]=$record;
                }
            }
            return $records;
        }
        protected function instantiate($attributes)
        {
            $class=get_class($this);
            $model=new $class(null);
            return $model;
        }
        public function offsetExists($offset)
        {
            return $this->__isset($offset);
        }
    }
    class CBaseActiveRelation extends CComponent
```

```
{
        public $name;
        public $className;
        public $foreignKey;
        public $select='*';
        public $condition='';
        public $params=array();
        public $group='';
        public $join='';
        public $having='';
        public $order='';
        public function __construct($name,$className,$foreignKey,$options=array())
        {
                $this->name=$name;
                $this->className=$className;
                $this->foreignKey=$foreignKey;
                foreach($options as $name=>$value)
                        $this->$name=$value;
        }
        public function mergeWith($criteria,$fromScope=false)
        {
                if($criteria instanceof CDbCriteria)
                        $criteria=$criteria->toArray();
                if(isset($criteria['select']) && $this->select!==$criteria['select'])
                {
                        if($this->select==='*')
                                $this->select=$criteria['select'];
                        elseif($criteria['select']!=='*')
                        {

        $select1=is_string($this->select)?preg_split('/\s*,\s*/',trim($this->select),-1,PREG_SPLIT_NO
_EMPTY):$this->select;

        $select2=is_string($criteria['select'])?preg_split('/\s*,\s*/',trim($criteria['select']),-1,PREG_S
PLIT_NO_EMPTY):$criteria['select'];
                                $this->select=array_merge($select1,array_diff($select2,$select1));
                        }
                }
                if(isset($criteria['condition']) && $this->condition!==$criteria['condition'])
                {
                        if($this->condition==='')
                                $this->condition=$criteria['condition'];
                        elseif($criteria['condition']!=='')
                                $this->condition="({$this->condition}) AND ({$criteria['condition']})";
```

```
			}
			if(isset($criteria['params']) && $this->params!==$criteria['params'])
				$this->params=array_merge($this->params,$criteria['params']);
			if(isset($criteria['order']) && $this->order!==$criteria['order'])
			{
				if($this->order==='')
					$this->order=$criteria['order'];
				elseif($criteria['order']!=='')
					$this->order=$criteria['order'].', '.$this->order;
			}
			if(isset($criteria['group']) && $this->group!==$criteria['group'])
			{
				if($this->group==='')
					$this->group=$criteria['group'];
				elseif($criteria['group']!=='')
					$this->group.=', '.$criteria['group'];
			}
			if(isset($criteria['join']) && $this->join!==$criteria['join'])
			{
				if($this->join==='')
					$this->join=$criteria['join'];
				elseif($criteria['join']!=='')
					$this->join.=' '.$criteria['join'];
			}
			if(isset($criteria['having']) && $this->having!==$criteria['having'])
			{
				if($this->having==='')
					$this->having=$criteria['having'];
				elseif($criteria['having']!=='')
					$this->having="({$this->having}) AND ({$criteria['having']})";
			}
		}
	}
	class CStatRelation extends CBaseActiveRelation
	{
		public $select='COUNT(*)';
		public $defaultValue=0;
		public function mergeWith($criteria,$fromScope=false)
		{
			if($criteria instanceof CDbCriteria)
				$criteria=$criteria->toArray();
			parent::mergeWith($criteria,$fromScope);
			if(isset($criteria['defaultValue']))
				$this->defaultValue=$criteria['defaultValue'];
	}
}
class CActiveRelation extends CBaseActiveRelation
{
	public $joinType='LEFT OUTER JOIN';
	public $on='';
	public $alias;
	public $with=array();
	public $together;
	  public $scopes;
	public function mergeWith($criteria,$fromScope=false)
	{
		if($criteria instanceof CDbCriteria)
			$criteria=$criteria->toArray();
		if($fromScope)
		{
			if(isset($criteria['condition']) && $this->on!==$criteria['condition'])
			{
				if($this->on==='')
					$this->on=$criteria['condition'];
				elseif($criteria['condition']!=='')
					$this->on="({$this->on}) AND ({$criteria['condition']})";
			}
			unset($criteria['condition']);
		}
		parent::mergeWith($criteria);
		if(isset($criteria['joinType']))
			$this->joinType=$criteria['joinType'];
		if(isset($criteria['on']) && $this->on!==$criteria['on'])
		{
			if($this->on==='')
				$this->on=$criteria['on'];
			elseif($criteria['on']!=='')
				$this->on="({$this->on}) AND ({$criteria['on']})";
		}
		if(isset($criteria['with']))
			$this->with=$criteria['with'];
		if(isset($criteria['alias']))
			$this->alias=$criteria['alias'];
		if(isset($criteria['together']))
			$this->together=$criteria['together'];
	}
}
class CBelongsToRelation extends CActiveRelation
```

```
{
}
class CHasOneRelation extends CActiveRelation
{
        public $through;
}
class CHasManyRelation extends CActiveRelation
{
        public $limit=-1;
        public $offset=-1;
        public $index;
        public $through;
        public function mergeWith($criteria,$fromScope=false)
        {
                if($criteria instanceof CDbCriteria)
                        $criteria=$criteria->toArray();
                parent::mergeWith($criteria,$fromScope);
                if(isset($criteria['limit']) && $criteria['limit']>0)
                        $this->limit=$criteria['limit'];
                if(isset($criteria['offset']) && $criteria['offset']>=0)
                        $this->offset=$criteria['offset'];
                if(isset($criteria['index']))
                        $this->index=$criteria['index'];
        }
}
class CManyManyRelation extends CHasManyRelation
{
        private $_junctionTableName=null;
        private $_junctionForeignKeys=null;
        public function getJunctionTableName()
        {
                if ($this->_junctionTableName===null)
                        $this->initJunctionData();
                return $this->_junctionTableName;
        }
        public function getJunctionForeignKeys()
        {
                if ($this->_junctionForeignKeys===null)
                        $this->initJunctionData();
                return $this->_junctionForeignKeys;
        }
        private function initJunctionData()
        {
                if(!preg_match('/^\s*(.*?)\((.*)\)\s*$/',$this->foreignKey,$matches))
                throw new CDbException(Yii::t('yii','The relation "{relation}" in active record class
"{class}" is specified with an invalid foreign key. The format of the foreign key must be
"joinTable(fk1,fk2,...)".',
                    array('{class}'=>$this->className,'{relation}'=>$this->name)));
            $this->_junctionTableName=$matches[1];

        $this->_junctionForeignKeys=preg_split('/\s*,\s*/',$matches[2],-1,PREG_SPLIT_NO_EMPTY);
        }
}
class CActiveRecordMetaData
{
        public $tableSchema;
        public $columns;
        public $relations=array();
        public $attributeDefaults=array();
        private $_model;
        public function __construct($model)
        {
            $this->_model=$model;
            $tableName=$model->tableName();
            if(($table=$model->getDbConnection()->getSchema()->getTable($tableName))===null)
                throw new CDbException(Yii::t('yii','The table "{table}" for active record class
"{class}" cannot be found in the database.',
                    array('{class}'=>get_class($model),'{table}'=>$tableName)));
            if($table->primaryKey===null)
            {
                $table->primaryKey=$model->primaryKey();
                if(is_string($table->primaryKey) && isset($table->columns[$table->primaryKey]))
                    $table->columns[$table->primaryKey]->isPrimaryKey=true;
                elseif(is_array($table->primaryKey))
                {
                    foreach($table->primaryKey as $name)
                    {
                        if(isset($table->columns[$name]))
                            $table->columns[$name]->isPrimaryKey=true;
                    }
                }
            }
            $this->tableSchema=$table;
            $this->columns=$table->columns;
            foreach($table->columns as $name=>$column)
            {
                if(!$column->isPrimaryKey && $column->defaultValue!==null)
                    $this->attributeDefaults[$name]=$column->defaultValue;
```

```
		}
		foreach($model->relations() as $name=>$config)
		{
			$this->addRelation($name,$config);
		}
	}
	public function addRelation($name,$config)
	{
		if(isset($config[0],$config[1],$config[2]))	// relation class, AR class, FK
			$this->relations[$name]=new
$config[0]($name,$config[1],$config[2],array_slice($config,3));
		else
			throw new CDbException(Yii::t('yii','Active record "{class}" has an invalid
configuration for relation "{relation}". It must specify the relation type, the related active record
class and the foreign key.', array('{class}'=>get_class($this->_model),'{relation}'=>$name)));
	}
	public function hasRelation($name)
	{
		return isset($this->relations[$name]);
	}
	public function removeRelation($name)
	{
		unset($this->relations[$name]);
	}
}
class CDbConnection extends CApplicationComponent
{
	public $connectionString;
	public $username='';
	public $password='';
	public $schemaCachingDuration=0;
	public $schemaCachingExclude=array();
	public $schemaCacheID='cache';
	public $queryCachingDuration=0;
	public $queryCachingDependency;
	public $queryCachingCount=0;
	public $queryCacheID='cache';
	public $autoConnect=true;
	public $charset;
	public $emulatePrepare;
	public $enableParamLogging=false;
	public $enableProfiling=false;
	public $tablePrefix;
	public $initSQLs;
public $driverMap=array(
        'pgsql'=>'CPgsqlSchema',        // PostgreSQL
        'mysqli'=>'CMysqlSchema',      // MySQL
        'mysql'=>'CMysqlSchema',        // MySQL
        'sqlite'=>'CSqliteSchema',    // sqlite 3
        'sqlite2'=>'CSqliteSchema', // sqlite 2
        'mssql'=>'CMssqlSchema',        // Mssql driver on windows hosts
        'dblib'=>'CMssqlSchema',        // dblib drivers on linux (and maybe others os) hosts
        'sqlsrv'=>'CMssqlSchema',      // Mssql
        'oci'=>'COciSchema',                // Oracle driver
);
public $pdoClass = 'PDO';
private $_attributes=array();
private $_active=false;
private $_pdo;
private $_transaction;
private $_schema;
public function __construct($dsn='',$username='',$password='')
{
        $this->connectionString=$dsn;
        $this->username=$username;
        $this->password=$password;
}
public function __sleep()
{
        $this->close();
        return array_keys(get_object_vars($this));
}
public static function getAvailableDrivers()
{
        return PDO::getAvailableDrivers();
}
public function init()
{
        parent::init();
        if($this->autoConnect)
                $this->setActive(true);
}
public function getActive()
{
        return $this->_active;
}
public function setActive($value)
{
		if($value!=$this->_active)
		{
			if($value)
				$this->open();
			else
				$this->close();
		}
	}
	public function cache($duration, $dependency=null, $queryCount=1)
	{
		$this->queryCachingDuration=$duration;
		$this->queryCachingDependency=$dependency;
		$this->queryCachingCount=$queryCount;
		return $this;
	}
	protected function open()
	{
		if($this->_pdo===null)
		{
			if(empty($this->connectionString))
				throw new CDbException('CDbConnection.connectionString cannot be
empty.');
			try
			{
				$this->_pdo=$this->createPdoInstance();
				$this->initConnection($this->_pdo);
				$this->_active=true;
			}
			catch(PDOException $e)
			{
				if(YII_DEBUG)
				{
					throw new CDbException('CDbConnection failed to open the DB
connection: '.
						$e->getMessage(),(int)$e->getCode(),$e->errorInfo);
				}
				else
				{

	Yii::log($e->getMessage(),CLogger::LEVEL_ERROR,'exception.CDbException');
					throw new CDbException('CDbConnection failed to open the DB
connection.',(int)$e->getCode(),$e->errorInfo);
				}
			}
		}
	}
	protected function close()
	{
		$this->_pdo=null;
		$this->_active=false;
		$this->_schema=null;
	}
	protected function createPdoInstance()
	{
		$pdoClass=$this->pdoClass;
		if(($pos=strpos($this->connectionString,':'))!==false)
		{
			$driver=strtolower(substr($this->connectionString,0,$pos));
			if($driver==='mssql' || $driver==='dblib')
				$pdoClass='CMssqlPdoAdapter';
			elseif($driver==='sqlsrv')
				$pdoClass='CMssqlSqlsrvPdoAdapter';
		}
		return new $pdoClass($this->connectionString,$this->username,
							$this->password,$this->_attributes);
	}
	protected function initConnection($pdo)
	{
		$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
		if($this->emulatePrepare!==null && constant('PDO::ATTR_EMULATE_PREPARES'))
			$pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES,$this->emulatePrepare);
		if($this->charset!==null)
		{
			$driver=strtolower($pdo->getAttribute(PDO::ATTR_DRIVER_NAME));
			if(in_array($driver,array('pgsql','mysql','mysqli')))
				$pdo->exec('SET NAMES '.$pdo->quote($this->charset));
		}
		if($this->initSQLs!==null)
		{
			foreach($this->initSQLs as $sql)
				$pdo->exec($sql);
		}
	}
	public function getPdoInstance()
	{
		return $this->_pdo;
	}
	public function createCommand($query=null)
	{
		$this->setActive(true);
		return new CDbCommand($this,$query);
	}
	public function getCurrentTransaction()
	{
		if($this->_transaction!==null)
		{
			if($this->_transaction->getActive())
				return $this->_transaction;
		}
		return null;
	}
	public function beginTransaction()
	{
		$this->setActive(true);
		$this->_pdo->beginTransaction();
		return $this->_transaction=new CDbTransaction($this);
	}
	public function getSchema()
	{
		if($this->_schema!==null)
			return $this->_schema;
		else
		{
			$driver=$this->getDriverName();
			if(isset($this->driverMap[$driver]))
				return	$this->_schema=Yii::createComponent($this->driverMap[$driver],
$this);
			else
				throw new CDbException(Yii::t('yii','CDbConnection does not support reading
schema for {driver} database.',
					array('{driver}'=>$driver)));
		}
	}
	public function getCommandBuilder()
	{
		return $this->getSchema()->getCommandBuilder();
	}
	public function getLastInsertID($sequenceName='')
	{
		$this->setActive(true);
		return $this->_pdo->lastInsertId($sequenceName);
	}
```

```
public function quoteValue($str)
{
        if(is_int($str) || is_float($str))
                return $str;
        $this->setActive(true);
        if(($value=$this->_pdo->quote($str))!==false)
                return $value;
        else    // the driver doesn't support quote (e.g. oci)
                return "'" . addcslashes(str_replace("'", "''", $str), "\000\n\r\\\032") . "'";
}
public function quoteTableName($name)
{
        return $this->getSchema()->quoteTableName($name);
}
public function quoteColumnName($name)
{
        return $this->getSchema()->quoteColumnName($name);
}
public function getPdoType($type)
{
        static $map=array
        (
                'boolean'=>PDO::PARAM_BOOL,
                'integer'=>PDO::PARAM_INT,
                'string'=>PDO::PARAM_STR,
                'resource'=>PDO::PARAM_LOB,
                'NULL'=>PDO::PARAM_NULL,
        );
        return isset($map[$type]) ? $map[$type] : PDO::PARAM_STR;
}
public function getColumnCase()
{
        return $this->getAttribute(PDO::ATTR_CASE);
}
public function setColumnCase($value)
{
        $this->setAttribute(PDO::ATTR_CASE,$value);
}
public function getNullConversion()
{
        return $this->getAttribute(PDO::ATTR_ORACLE_NULLS);
}
public function setNullConversion($value)
{
        $this->setAttribute(PDO::ATTR_ORACLE_NULLS,$value);
}
public function getAutoCommit()
{
        return $this->getAttribute(PDO::ATTR_AUTOCOMMIT);
}
public function setAutoCommit($value)
{
        $this->setAttribute(PDO::ATTR_AUTOCOMMIT,$value);
}
public function getPersistent()
{
        return $this->getAttribute(PDO::ATTR_PERSISTENT);
}
public function setPersistent($value)
{
        return $this->setAttribute(PDO::ATTR_PERSISTENT,$value);
}
public function getDriverName()
{
        if(($pos=strpos($this->connectionString, ':'))!==false)
                return strtolower(substr($this->connectionString, 0, $pos));
        // return $this->getAttribute(PDO::ATTR_DRIVER_NAME);
}
public function getClientVersion()
{
        return $this->getAttribute(PDO::ATTR_CLIENT_VERSION);
}
public function getConnectionStatus()
{
        return $this->getAttribute(PDO::ATTR_CONNECTION_STATUS);
}
public function getPrefetch()
{
        return $this->getAttribute(PDO::ATTR_PREFETCH);
}
public function getServerInfo()
{
        return $this->getAttribute(PDO::ATTR_SERVER_INFO);
}
public function getServerVersion()
{
        return $this->getAttribute(PDO::ATTR_SERVER_VERSION);
}
```

```
        public function getTimeout()
        {
                return $this->getAttribute(PDO::ATTR_TIMEOUT);
        }
        public function getAttribute($name)
        {
                $this->setActive(true);
                return $this->_pdo->getAttribute($name);
        }
        public function setAttribute($name,$value)
        {
                if($this->_pdo instanceof PDO)
                        $this->_pdo->setAttribute($name,$value);
                else
                        $this->_attributes[$name]=$value;
        }
        public function getAttributes()
        {
                return $this->_attributes;
        }
        public function setAttributes($values)
        {
                foreach($values as $name=>$value)
                        $this->_attributes[$name]=$value;
        }
        public function getStats()
        {
                $logger=Yii::getLogger();
                $timings=$logger->getProfilingResults(null,'system.db.CDbCommand.query');
                $count=count($timings);
                $time=array_sum($timings);
                $timings=$logger->getProfilingResults(null,'system.db.CDbCommand.execute');
                $count+=count($timings);
                $time+=array_sum($timings);
                return array($count,$time);
        }
}
abstract class CDbSchema extends CComponent
{
        public $columnTypes=array();
        private $_tableNames=array();
        private $_tables=array();
        private $_connection;
        private $_builder;
        private $_cacheExclude=array();
        abstract protected function loadTable($name);
        public function __construct($conn)
        {
                $this->_connection=$conn;
                foreach($conn->schemaCachingExclude as $name)
                        $this->_cacheExclude[$name]=true;
        }
        public function getDbConnection()
        {
                return $this->_connection;
        }
        public function getTable($name,$refresh=false)
        {
                if($refresh===false && isset($this->_tables[$name]))
                        return $this->_tables[$name];
                else
                {
                        if($this->_connection->tablePrefix!==null && strpos($name,'{{')!==false)

        $realName=preg_replace('/\{\{(.*?)\}\}/',$this->_connection->tablePrefix.'$1',$name);
                        else
                                $realName=$name;
                        // temporarily disable query caching
                        if($this->_connection->queryCachingDuration>0)
                        {
                                $qcDuration=$this->_connection->queryCachingDuration;
                                $this->_connection->queryCachingDuration=0;
                        }
                        if(!isset($this->_cacheExclude[$name])                                  &&
($duration=$this->_connection->schemaCachingDuration)>0                                         &&
$this->_connection->schemaCacheID!==false                                                       &&
($cache=Yii::app()->getComponent($this->_connection->schemaCacheID))!==null)
                        {

        $key='yii:dbschema'.$this->_connection->connectionString.':'.$this->_connection->usernam
e.':'.$name;
                                $table=$cache->get($key);
                                if($refresh===true || $table===false)
                                {
                                        $table=$this->loadTable($realName);
                                        if($table!==null)
                                                $cache->set($key,$table,$duration);
                                }
                $this->_tables[$name]=$table;
            }
            else
                $this->_tables[$name]=$table=$this->loadTable($realName);
            if(isset($qcDuration))    // re-enable query caching
                $this->_connection->queryCachingDuration=$qcDuration;
            return $table;
        }
    }
    public function getTables($schema='')
    {
        $tables=array();
        foreach($this->getTableNames($schema) as $name)
        {
            if(($table=$this->getTable($name))!==null)
                $tables[$name]=$table;
        }
        return $tables;
    }
    public function getTableNames($schema='')
    {
        if(!isset($this->_tableNames[$schema]))
            $this->_tableNames[$schema]=$this->findTableNames($schema);
        return $this->_tableNames[$schema];
    }
    public function getCommandBuilder()
    {
        if($this->_builder!==null)
            return $this->_builder;
        else
            return $this->_builder=$this->createCommandBuilder();
    }
    public function refresh()
    {
        if(($duration=$this->_connection->schemaCachingDuration)>0                          &&
$this->_connection->schemaCacheID!==false                                                   &&
($cache=Yii::app()->getComponent($this->_connection->schemaCacheID))!==null)
        {
            foreach(array_keys($this->_tables) as $name)
            {
                if(!isset($this->_cacheExclude[$name]))
                {

    $key='yii:dbschema'.$this->_connection->connectionString.':'.$this->_connection->usernam
e.':'.$name;
                    $cache->delete($key);
                }
            }
        }
        $this->_tables=array();
        $this->_tableNames=array();
        $this->_builder=null;
    }
    public function quoteTableName($name)
    {
        if(strpos($name,'.')===false)
            return $this->quoteSimpleTableName($name);
        $parts=explode('.',$name);
        foreach($parts as $i=>$part)
            $parts[$i]=$this->quoteSimpleTableName($part);
        return implode('.',$parts);
    }
    public function quoteSimpleTableName($name)
    {
        return "'".$name."'";
    }
    public function quoteColumnName($name)
    {
        if(($pos=strrpos($name,'.'))!==false)
        {
            $prefix=$this->quoteTableName(substr($name,0,$pos)).'.';
            $name=substr($name,$pos+1);
        }
        else
            $prefix='';
        return $prefix . ($name==='*' ? $name : $this->quoteSimpleColumnName($name));
    }
    public function quoteSimpleColumnName($name)
    {
        return '"'.$name.'"';
    }
    public function compareTableNames($name1,$name2)
    {
        $name1=str_replace(array('"','`',"'"),'',$name1);
        $name2=str_replace(array('"','`',"'"),'',$name2);
        if(($pos=strrpos($name1,'.'))!==false)
            $name1=substr($name1,$pos+1);
        if(($pos=strrpos($name2,'.'))!==false)
```

```
			$name2=substr($name2,$pos+1);
		if($this->_connection->tablePrefix!==null)
		{
			if(strpos($name1,'{')!==false)
				$name1=$this->_connection->tablePrefix.str_replace(array('{','}'),'',$name1);
			if(strpos($name2,'{')!==false)
				$name2=$this->_connection->tablePrefix.str_replace(array('{','}'),'',$name2);
		}
		return $name1===$name2;
	}
	public function resetSequence($table,$value=null)
	{
	}
	public function checkIntegrity($check=true,$schema='')
	{
	}
	protected function createCommandBuilder()
	{
		return new CDbCommandBuilder($this);
	}
	protected function findTableNames($schema='')
	{
		throw new CDbException(Yii::t('yii','{class} does not support fetching all table names.',
			array('{class}'=>get_class($this))));
	}
	public function getColumnType($type)
	{
		if(isset($this->columnTypes[$type]))
			return $this->columnTypes[$type];
		elseif(($pos=strpos($type,' '))!==false)
		{
			$t=substr($type,0,$pos);
			return	(isset($this->columnTypes[$t])	?	$this->columnTypes[$t]	:
$t).substr($type,$pos);
		}
		else
			return $type;
	}
	public function createTable($table, $columns, $options=null)
	{
		$cols=array();
		foreach($columns as $name=>$type)
		{
			if(is_string($name))
```

```
                    $cols[]="\t".$this->quoteColumnName($name).'
'.$this->getColumnType($type);
                else
                    $cols[]="\t".$type;
            }
            $sql="CREATE            TABLE            ".$this->quoteTableName($table)."
(\n".implode(",\n",$cols)."\n)";
            return $options===null ? $sql : $sql.' '.$options;
        }
        public function renameTable($table, $newName)
        {
            return  'RENAME  TABLE  '  .  $this->quoteTableName($table)  .  '  TO  '  .
$this->quoteTableName($newName);
        }
        public function dropTable($table)
        {
            return "DROP TABLE ".$this->quoteTableName($table);
        }
        public function truncateTable($table)
        {
            return "TRUNCATE TABLE ".$this->quoteTableName($table);
        }
        public function addColumn($table, $column, $type)
        {
            return 'ALTER TABLE ' . $this->quoteTableName($table)
                . ' ADD ' . $this->quoteColumnName($column) . ' '
                . $this->getColumnType($type);
        }
        public function dropColumn($table, $column)
        {
            return "ALTER TABLE ".$this->quoteTableName($table)
                ." DROP COLUMN ".$this->quoteColumnName($column);
        }
        public function renameColumn($table, $name, $newName)
        {
            return "ALTER TABLE ".$this->quoteTableName($table)
                . " RENAME COLUMN ".$this->quoteColumnName($name)
                . " TO ".$this->quoteColumnName($newName);
        }
        public function alterColumn($table, $column, $type)
        {
            return 'ALTER TABLE ' . $this->quoteTableName($table) . ' CHANGE '
                . $this->quoteColumnName($column) . ' '
                . $this->quoteColumnName($column) . ' '
                . $this->getColumnType($type);
        }
        public function addForeignKey($name, $table, $columns, $refTable, $refColumns,
$delete=null, $update=null)
        {
                $columns=preg_split('/\s*,\s*/',$columns,-1,PREG_SPLIT_NO_EMPTY);
                foreach($columns as $i=>$col)
                        $columns[$i]=$this->quoteColumnName($col);
                $refColumns=preg_split('/\s*,\s*/',$refColumns,-1,PREG_SPLIT_NO_EMPTY);
                foreach($refColumns as $i=>$col)
                        $refColumns[$i]=$this->quoteColumnName($col);
                $sql='ALTER TABLE '.$this->quoteTableName($table)
                        .' ADD CONSTRAINT '.$this->quoteColumnName($name)
                        .' FOREIGN KEY ('.implode(', ', $columns).')'
                        .' REFERENCES '.$this->quoteTableName($refTable)
                        .' ('.implode(', ', $refColumns).')';
                if($delete!==null)
                        $sql.=' ON DELETE '.$delete;
                if($update!==null)
                        $sql.=' ON UPDATE '.$update;
                return $sql;
        }
        public function dropForeignKey($name, $table)
        {
                return 'ALTER TABLE '.$this->quoteTableName($table)
                        .' DROP CONSTRAINT '.$this->quoteColumnName($name);
        }
        public function createIndex($name, $table, $column, $unique=false)
        {
                $cols=array();
                $columns=preg_split('/\s*,\s*/',$column,-1,PREG_SPLIT_NO_EMPTY);
                foreach($columns as $col)
                {
                        if(strpos($col,'(')!==false)
                                $cols[]=$col;
                        else
                                $cols[]=$this->quoteColumnName($col);
                }
                return ($unique ? 'CREATE UNIQUE INDEX ' : 'CREATE INDEX ')
                        . $this->quoteTableName($name).' ON '
                        . $this->quoteTableName($table).' ('.implode(', ',$cols).')';
        }
        public function dropIndex($name, $table)
        {
```

```
		return 'DROP INDEX '.$this->quoteTableName($name).' ON
'.$this->quoteTableName($table);
	}
	public function addPrimaryKey($name,$table,$columns)
	{
		$columns=preg_split('/\s*,\s*/',$columns,-1,PREG_SPLIT_NO_EMPTY);
		foreach($columns as $i=>$col)
			$columns[$i]=$this->quoteColumnName($col);
		return 'ALTER TABLE ' . $this->quoteTableName($table) . ' ADD CONSTRAINT '
			. $this->quoteColumnName($name) . '    PRIMARY KEY ('
			. implode(', ', $columns). ' )';
	}
	public function dropPrimaryKey($name,$table)
	{
		return 'ALTER TABLE ' . $this->quoteTableName($table) . ' DROP CONSTRAINT '
			. $this->quoteColumnName($name);
	}
}
class CSqliteSchema extends CDbSchema
{
	public $columnTypes=array(
		'pk' => 'integer PRIMARY KEY AUTOINCREMENT NOT NULL',
		'string' => 'varchar(255)',
		'text' => 'text',
		'integer' => 'integer',
		'float' => 'float',
		'decimal' => 'decimal',
		'datetime' => 'datetime',
		'timestamp' => 'timestamp',
		'time' => 'time',
		'date' => 'date',
		'binary' => 'blob',
		'boolean' => 'tinyint(1)',
		'money' => 'decimal(19,4)',
	);
	public function resetSequence($table,$value=null)
	{
		if($table->sequenceName!==null)
		{
			if($value===null)
				$value=$this->getDbConnection()->createCommand("SELECT
MAX(`{$table->primaryKey}`) FROM {$table->rawName}")->queryScalar();
			else
				$value=(int)$value-1;
			try
			{
				// it's possible sqlite_sequence does not exist
				$this->getDbConnection()->createCommand("UPDATE   sqlite_sequence   SET
seq='$value' WHERE name='{$table->name}'")->execute();
			}
			catch(Exception $e)
			{
			}
		}
	}
	public function checkIntegrity($check=true,$schema='')
	{
		// SQLite doesn't enforce integrity
		return;
	}
	protected function findTableNames($schema='')
	{
		$sql="SELECT       DISTINCT       tbl_name       FROM       sqlite_master       WHERE
tbl_name<>'sqlite_sequence'";
		return $this->getDbConnection()->createCommand($sql)->queryColumn();
	}
	protected function createCommandBuilder()
	{
		return new CSqliteCommandBuilder($this);
	}
	protected function loadTable($name)
	{
		$table=new CDbTableSchema;
		$table->name=$name;
		$table->rawName=$this->quoteTableName($name);
		if($this->findColumns($table))
		{
			$this->findConstraints($table);
			return $table;
		}
		else
			return null;
	}
	protected function findColumns($table)
	{
		$sql="PRAGMA table_info({$table->rawName})";
		$columns=$this->getDbConnection()->createCommand($sql)->queryAll();
		if(empty($columns))
```

```
                return false;
            foreach($columns as $column)
            {
                $c=$this->createColumn($column);
                $table->columns[$c->name]=$c;
                if($c->isPrimaryKey)
                {
                    if($table->primaryKey===null)
                        $table->primaryKey=$c->name;
                    elseif(is_string($table->primaryKey))
                        $table->primaryKey=array($table->primaryKey,$c->name);
                    else
                        $table->primaryKey[]=$c->name;
                }
            }
            if(is_string($table->primaryKey)
    && !strncasecmp($table->columns[$table->primaryKey]->dbType,'int',3))
            {
                $table->sequenceName='';
                $table->columns[$table->primaryKey]->autoIncrement=true;
            }
            return true;
        }
        protected function findConstraints($table)
        {
            $foreignKeys=array();
            $sql="PRAGMA foreign_key_list({$table->rawName})";
            $keys=$this->getDbConnection()->createCommand($sql)->queryAll();
            foreach($keys as $key)
            {
                $column=$table->columns[$key['from']];
                $column->isForeignKey=true;
                $foreignKeys[$key['from']]=array($key['table'],$key['to']);
            }
            $table->foreignKeys=$foreignKeys;
        }
        protected function createColumn($column)
        {
            $c=new CSqliteColumnSchema;
            $c->name=$column['name'];
            $c->rawName=$this->quoteColumnName($c->name);
            $c->allowNull=!$column['notnull'];
            $c->isPrimaryKey=$column['pk']!=0;
            $c->isForeignKey=false;
```

```
		$c->comment=null; // SQLite does not support column comments at all
		$c->init(strtolower($column['type']),$column['dflt_value']);
		return $c;
	}
	public function renameTable($table, $newName)
	{
		return 'ALTER TABLE ' . $this->quoteTableName($table) . ' RENAME TO ' .
$this->quoteTableName($newName);
	}
	public function truncateTable($table)
	{
		return "DELETE FROM ".$this->quoteTableName($table);
	}
	public function dropColumn($table, $column)
	{
		throw new CDbException(Yii::t('yii', 'Dropping DB column is not supported by SQLite.'));
	}
	public function renameColumn($table, $name, $newName)
	{
		throw new CDbException(Yii::t('yii', 'Renaming a DB column is not supported by
SQLite.'));
	}
	public function addForeignKey($name, $table, $columns, $refTable, $refColumns,
$delete=null, $update=null)
	{
		throw new CDbException(Yii::t('yii', 'Adding a foreign key constraint to an existing table
is not supported by SQLite.'));
	}
	public function dropForeignKey($name, $table)
	{
		throw new CDbException(Yii::t('yii', 'Dropping a foreign key constraint is not supported
by SQLite.'));
	}
	public function alterColumn($table, $column, $type)
	{
		throw new CDbException(Yii::t('yii', 'Altering a DB column is not supported by SQLite.'));
	}
	public function dropIndex($name, $table)
	{
		return 'DROP INDEX '.$this->quoteTableName($name);
	}
	public function addPrimaryKey($name,$table,$columns)
	{
		throw new CDbException(Yii::t('yii', 'Adding a primary key after table has been created
is not supported by SQLite.'));
	}
	public function dropPrimaryKey($name,$table)
	{
		throw new CDbException(Yii::t('yii', 'Removing a primary key after table has been
created is not supported by SQLite.'));
	}
}
class CDbTableSchema extends CComponent
{
	public $name;
	public $rawName;
	public $primaryKey;
	public $sequenceName;
	public $foreignKeys=array();
	public $columns=array();
	public function getColumn($name)
	{
		return isset($this->columns[$name]) ? $this->columns[$name] : null;
	}
	public function getColumnNames()
	{
		return array_keys($this->columns);
	}
}
class CDbCommand extends CComponent
{
	public $params=array();
	private $_connection;
	private $_text;
	private $_statement;
	private $_paramLog=array();
	private $_query;
	private $_fetchMode = array(PDO::FETCH_ASSOC);
	public function __construct(CDbConnection $connection,$query=null)
	{
		$this->_connection=$connection;
		if(is_array($query))
		{
			foreach($query as $name=>$value)
				$this->$name=$value;
		}
		else
			$this->setText($query);
```

```
}
public function __sleep()
{
        $this->_statement=null;
        return array_keys(get_object_vars($this));
}
public function setFetchMode($mode)
{
        $params=func_get_args();
        $this->_fetchMode = $params;
        return $this;
}
public function reset()
{
        $this->_text=null;
        $this->_query=null;
        $this->_statement=null;
        $this->_paramLog=array();
        $this->params=array();
        return $this;
}
public function getText()
{
        if($this->_text=='' && !empty($this->_query))
                $this->setText($this->buildQuery($this->_query));
        return $this->_text;
}
public function setText($value)
{
        if($this->_connection->tablePrefix!==null && $value!='')

$this->_text=preg_replace('/{{(.*?)}}/',$this->_connection->tablePrefix.'\1',$value);
        else
                $this->_text=$value;
        $this->cancel();
        return $this;
}
public function getConnection()
{
        return $this->_connection;
}
public function getPdoStatement()
{
        return $this->_statement;
```

```
	}
	public function prepare()
	{
		if($this->_statement==null)
		{
			try
			{

	$this->_statement=$this->getConnection()->getPdoInstance()->prepare($this->getText());
				$this->_paramLog=array();
			}
			catch(Exception $e)
			{
				Yii::log('Error in preparing SQL: 
'.$this->getText(),CLogger::LEVEL_ERROR,'system.db.CDbCommand');
				$errorInfo=$e instanceof PDOException ? $e->errorInfo : null;
				throw new CDbException(Yii::t('yii','CDbCommand failed to prepare the SQL 
statement: {error}',
					array('{error}'=>$e->getMessage())),(int)$e->getCode(),$errorInfo);
			}
		}
	}
	public function cancel()
	{
		$this->_statement=null;
	}
	public function bindParam($name, &$value, $dataType=null, $length=null, 
$driverOptions=null)
	{
		$this->prepare();
		if($dataType===null)

	$this->_statement->bindParam($name,$value,$this->_connection->getPdoType(gettype($va
lue)));
		elseif($length===null)
			$this->_statement->bindParam($name,$value,$dataType);
		elseif($driverOptions===null)
			$this->_statement->bindParam($name,$value,$dataType,$length);
		else
			$this->_statement->bindParam($name,$value,$dataType,$length,$driverOptions);
		$this->_paramLog[$name]=&$value;
		return $this;
	}
	public function bindValue($name, $value, $dataType=null)
	{
		$this->prepare();
		if($dataType===null)

	$this->_statement->bindValue($name,$value,$this->_connection->getPdoType(gettype($val
ue)));
		else
			$this->_statement->bindValue($name,$value,$dataType);
		$this->_paramLog[$name]=$value;
		return $this;
	}
	public function bindValues($values)
	{
		$this->prepare();
		foreach($values as $name=>$value)
		{

	$this->_statement->bindValue($name,$value,$this->_connection->getPdoType(gettype($val
ue)));
			$this->_paramLog[$name]=$value;
		}
		return $this;
	}
	public function execute($params=array())
	{
		if($this->_connection->enableParamLogging                                   &&
($pars=array_merge($this->_paramLog,$params))!==array())
		{
			$p=array();
			foreach($pars as $name=>$value)
				$p[$name]=$name.'='.var_export($value,true);
			$par='. Bound with ' .implode(', ',$p);
		}
		else
			$par='';
		try
		{
			if($this->_connection->enableProfiling)

	Yii::beginProfile('system.db.CDbCommand.execute('.$this->getText().$par.')','system.db.CDb
Command.execute');
			$this->prepare();
			if($params===array())
				$this->_statement->execute();
			else
				$this->_statement->execute($params);
			$n=$this->_statement->rowCount();
			if($this->_connection->enableProfiling)

	Yii::endProfile('system.db.CDbCommand.execute('.$this->getText().$par.')','system.db.CDbCo
mmand.execute');
			return $n;
		}
		catch(Exception $e)
		{
			if($this->_connection->enableProfiling)

	Yii::endProfile('system.db.CDbCommand.execute('.$this->getText().$par.')','system.db.CDbCo
mmand.execute');
			$errorInfo=$e instanceof PDOException ? $e->errorInfo : null;
			$message=$e->getMessage();
			Yii::log(Yii::t('yii','CDbCommand::execute() failed: {error}. The SQL statement
executed was: {sql}.',
				array('{error}'=>$message,
'{sql}'=>$this->getText().$par)),CLogger::LEVEL_ERROR,'system.db.CDbCommand');
			if(YII_DEBUG)
				$message.='. The SQL statement executed was: '.$this->getText().$par;
			throw new CDbException(Yii::t('yii','CDbCommand failed to execute the SQL
statement: {error}',
				array('{error}'=>$message)),(int)$e->getCode(),$errorInfo);
		}
	}
	public function query($params=array())
	{
		return $this->queryInternal('',0,$params);
	}
	public function queryAll($fetchAssociative=true,$params=array())
	{
		return $this->queryInternal('fetchAll',$fetchAssociative ? $this->_fetchMode :
PDO::FETCH_NUM, $params);
	}
	public function queryRow($fetchAssociative=true,$params=array())
	{
		return $this->queryInternal('fetch',$fetchAssociative ? $this->_fetchMode :
PDO::FETCH_NUM, $params);
	}
	public function queryScalar($params=array())
	{
		$result=$this->queryInternal('fetchColumn',0,$params);
		if(is_resource($result) && get_resource_type($result)==='stream')
			return stream_get_contents($result);
		else
			return $result;
	}
	public function queryColumn($params=array())
	{
		return $this->queryInternal('fetchAll',array(PDO::FETCH_COLUMN, 0),$params);
	}
	private function queryInternal($method,$mode,$params=array())
	{
		$params=array_merge($this->params,$params);
		if($this->_connection->enableParamLogging                                    &&
($pars=array_merge($this->_paramLog,$params))!==array())
		{
			$p=array();
			foreach($pars as $name=>$value)
				$p[$name]=$name.'='.var_export($value,true);
			$par='. Bound with '.implode(', ',$p);
		}
		else
			$par='';
		if($this->_connection->queryCachingCount>0 && $method!==''
				&& $this->_connection->queryCachingDuration>0
				&& $this->_connection->queryCacheID!==false
				&&
($cache=Yii::app()->getComponent($this->_connection->queryCacheID))!==null)
		{
			$this->_connection->queryCachingCount--;

	$cacheKey='yii:dbquery'.$this->_connection->connectionString.':'.$this->_connection->user
name;

	$cacheKey.=':'.$this->getText().':'.serialize(array_merge($this->_paramLog,$params));
			if(($result=$cache->get($cacheKey))!==false)
			{
				return $result[0];
			}
		}
		try
		{
			if($this->_connection->enableProfiling)
```

```
        Yii::beginProfile('system.db.CDbCommand.query('.$this->getText().$par.')','system.db.CDbCo
mmand.query');
                        $this->prepare();
                        if($params===array())
                                $this->_statement->execute();
                        else
                                $this->_statement->execute($params);
                        if($method==='')
                                $result=new CDbDataReader($this);
                        else
                        {
                                $mode=(array)$mode;
                                call_user_func_array(array($this->_statement, 'setFetchMode'), $mode);
                                $result=$this->_statement->$method();
                                $this->_statement->closeCursor();
                        }
                        if($this->_connection->enableProfiling)

        Yii::endProfile('system.db.CDbCommand.query('.$this->getText().$par.')','system.db.CDbCom
mand.query');
                        if(isset($cache,$cacheKey))
                                $cache->set($cacheKey,                                          array($result),
$this->_connection->queryCachingDuration, $this->_connection->queryCachingDependency);
                        return $result;
                }
                catch(Exception $e)
                {
                        if($this->_connection->enableProfiling)

        Yii::endProfile('system.db.CDbCommand.query('.$this->getText().$par.')','system.db.CDbCom
mand.query');
                        $errorInfo=$e instanceof PDOException ? $e->errorInfo : null;
                        $message=$e->getMessage();
                        Yii::log(Yii::t('yii','CDbCommand::{method}() failed: {error}. The SQL statement
executed was: {sql}.',
                                array('{method}'=>$method,                              '{error}'=>$message,
'{sql}'=>$this->getText().$par)),CLogger::LEVEL_ERROR,'system.db.CDbCommand');
                        if(YII_DEBUG)
                                $message.='. The SQL statement executed was: '.$this->getText().$par;
                        throw new CDbException(Yii::t('yii','CDbCommand failed to execute the SQL
statement: {error}',
                                array('{error}'=>$message)),(int)$e->getCode(),$errorInfo);
                }
        }
```

```
	public function buildQuery($query)
	{
		$sql=!empty($query['distinct']) ? 'SELECT DISTINCT' : 'SELECT';
		$sql.=' '.(!empty($query['select']) ? $query['select'] : '*');
		if(!empty($query['from']))
			$sql.="\nFROM ".$query['from'];
		else
			throw new CDbException(Yii::t('yii','The DB query must contain the "from"
portion.'));
		if(!empty($query['join']))
			$sql.="\n".(is_array($query['join']) ? implode("\n",$query['join']) : $query['join']);
		if(!empty($query['where']))
			$sql.="\nWHERE ".$query['where'];
		if(!empty($query['group']))
			$sql.="\nGROUP BY ".$query['group'];
		if(!empty($query['having']))
			$sql.="\nHAVING ".$query['having'];
		if(!empty($query['union']))
			$sql.="\nUNION (\n".(is_array($query['union']) ? implode("\n) UNION
(\n",$query['union']) : $query['union']) . ')';
		if(!empty($query['order']))
			$sql.="\nORDER BY ".$query['order'];
		$limit=isset($query['limit']) ? (int)$query['limit'] : -1;
		$offset=isset($query['offset']) ? (int)$query['offset'] : -1;
		if($limit>=0 || $offset>0)
			$sql=$this->_connection->getCommandBuilder()->applyLimit($sql,$limit,$offset);
		return $sql;
	}
	public function select($columns='*', $option='')
	{
		if(is_string($columns) && strpos($columns,'(')!==false)
			$this->_query['select']=$columns;
		else
		{
			if(!is_array($columns))
				$columns=preg_split('/\s*,\s*/',trim($columns),-1,PREG_SPLIT_NO_EMPTY);
			foreach($columns as $i=>$column)
			{
				if(is_object($column))
					$columns[$i]=(string)$column;
				elseif(strpos($column,'(')===false)
				{
					if(preg_match('/^(.*?)(?i:\s+as\s+|\s+)(.*)$/',$column,$matches))
```

```
        $columns[$i]=$this->_connection->quoteColumnName($matches[1]).'                    AS
'.$this->_connection->quoteColumnName($matches[2]);
                    else
                        $columns[$i]=$this->_connection->quoteColumnName($column);
                }
            }
            $this->_query['select']=implode(', ',$columns);
        }
        if($option!='')
            $this->_query['select']=$option.' '.$this->_query['select'];
        return $this;
    }
    public function getSelect()
    {
        return isset($this->_query['select']) ? $this->_query['select'] : '';
    }
    public function setSelect($value)
    {
        $this->select($value);
    }
    public function selectDistinct($columns='*')
    {
        $this->_query['distinct']=true;
        return $this->select($columns);
    }
    public function getDistinct()
    {
        return isset($this->_query['distinct']) ? $this->_query['distinct'] : false;
    }
    public function setDistinct($value)
    {
        $this->_query['distinct']=$value;
    }
    public function from($tables)
    {
        if(is_string($tables) && strpos($tables,'(')!==false)
            $this->_query['from']=$tables;
        else
        {
            if(!is_array($tables))
                $tables=preg_split('/\s*,\s*/',trim($tables),-1,PREG_SPLIT_NO_EMPTY);
            foreach($tables as $i=>$table)
            {
                if(strpos($table,'(')===false)
                {
                    if(preg_match('/^(.*?)(?i:\s+as\s+|\s+)(.*)$/',$table,$matches))    // with
alias
                        $tables[$i]=$this->_connection->quoteTableName($matches[1]).'
'.$this->_connection->quoteTableName($matches[2]);
                    else
                        $tables[$i]=$this->_connection->quoteTableName($table);
                }
            }
            $this->_query['from']=implode(', ',$tables);
        }
        return $this;
    }
    public function getFrom()
    {
        return isset($this->_query['from']) ? $this->_query['from'] : '';
    }
    public function setFrom($value)
    {
        $this->from($value);
    }
    public function where($conditions, $params=array())
    {
        $this->_query['where']=$this->processConditions($conditions);
        foreach($params as $name=>$value)
            $this->params[$name]=$value;
        return $this;
    }
    public function andWhere($conditions,$params=array())
    {
        if(isset($this->_query['where']))

    $this->_query['where']=$this->processConditions(array('AND',$this->_query['where'],$condi
tions));
        else
            $this->_query['where']=$this->processConditions($conditions);
        foreach($params as $name=>$value)
            $this->params[$name]=$value;
        return $this;
    }
    public function orWhere($conditions,$params=array())
    {
        if(isset($this->_query['where']))
    $this->_query['where']=$this->processConditions(array('OR',$this->_query['where'],$conditi
ons));
        else
            $this->_query['where']=$this->processConditions($conditions);
        foreach($params as $name=>$value)
            $this->params[$name]=$value;
        return $this;
    }
    public function getWhere()
    {
        return isset($this->_query['where']) ? $this->_query['where'] : '';
    }
    public function setWhere($value)
    {
        $this->where($value);
    }
    public function join($table, $conditions, $params=array())
    {
        return $this->joinInternal('join', $table, $conditions, $params);
    }
    public function getJoin()
    {
        return isset($this->_query['join']) ? $this->_query['join'] : '';
    }
    public function setJoin($value)
    {
        $this->_query['join']=$value;
    }
    public function leftJoin($table, $conditions, $params=array())
    {
        return $this->joinInternal('left join', $table, $conditions, $params);
    }
    public function rightJoin($table, $conditions, $params=array())
    {
        return $this->joinInternal('right join', $table, $conditions, $params);
    }
    public function crossJoin($table)
    {
        return $this->joinInternal('cross join', $table);
    }
    public function naturalJoin($table)
    {
        return $this->joinInternal('natural join', $table);
    }
```

```
public function group($columns)
{
	if(is_string($columns) && strpos($columns,'(')!==false)
		$this->_query['group']=$columns;
	else
	{
		if(!is_array($columns))
			$columns=preg_split('/\s*,\s*/',trim($columns),-1,PREG_SPLIT_NO_EMPTY);
		foreach($columns as $i=>$column)
		{
			if(is_object($column))
				$columns[$i]=(string)$column;
			elseif(strpos($column,'(')===false)
				$columns[$i]=$this->_connection->quoteColumnName($column);
		}
		$this->_query['group']=implode(', ',$columns);
	}
	return $this;
}
public function getGroup()
{
	return isset($this->_query['group']) ? $this->_query['group'] : '';
}
public function setGroup($value)
{
	$this->group($value);
}
public function having($conditions, $params=array())
{
	$this->_query['having']=$this->processConditions($conditions);
	foreach($params as $name=>$value)
		$this->params[$name]=$value;
	return $this;
}
public function getHaving()
{
	return isset($this->_query['having']) ? $this->_query['having'] : '';
}
public function setHaving($value)
{
	$this->having($value);
}
public function order($columns)
{
		if(is_string($columns) && strpos($columns,'(')!==false)
			$this->_query['order']=$columns;
		else
		{
			if(!is_array($columns))
				$columns=preg_split('/\s*,\s*/',trim($columns),-1,PREG_SPLIT_NO_EMPTY);
			foreach($columns as $i=>$column)
			{
				if(is_object($column))
					$columns[$i]=(string)$column;
				elseif(strpos($column,'(')===false)
				{
					if(preg_match('/^(.*?)\s+(asc|desc)$/i',$column,$matches))

	$columns[$i]=$this->_connection->quoteColumnName($matches[1]).'
'.strtoupper($matches[2]);
					else
						$columns[$i]=$this->_connection->quoteColumnName($column);
				}
			}
			$this->_query['order']=implode(', ',$columns);
		}
		return $this;
	}
	public function getOrder()
	{
		return isset($this->_query['order']) ? $this->_query['order'] : '';
	}
	public function setOrder($value)
	{
		$this->order($value);
	}
	public function limit($limit, $offset=null)
	{
		$this->_query['limit']=(int)$limit;
		if($offset!==null)
			$this->offset($offset);
		return $this;
	}
	public function getLimit()
	{
		return isset($this->_query['limit']) ? $this->_query['limit'] : -1;
	}
	public function setLimit($value)
```

```
{
        $this->limit($value);
}
public function offset($offset)
{
        $this->_query['offset']=(int)$offset;
        return $this;
}
public function getOffset()
{
        return isset($this->_query['offset']) ? $this->_query['offset'] : -1;
}
public function setOffset($value)
{
        $this->offset($value);
}
public function union($sql)
{
        if(isset($this->_query['union']) && is_string($this->_query['union']))
                $this->_query['union']=array($this->_query['union']);
        $this->_query['union'][]=$sql;
        return $this;
}
public function getUnion()
{
        return isset($this->_query['union']) ? $this->_query['union'] : '';
}
public function setUnion($value)
{
        $this->_query['union']=$value;
}
public function insert($table, $columns)
{
        $params=array();
        $names=array();
        $placeholders=array();
        foreach($columns as $name=>$value)
        {
                $names[]=$this->_connection->quoteColumnName($name);
                if($value instanceof CDbExpression)
                {
                        $placeholders[] = $value->expression;
                        foreach($value->params as $n => $v)
                                $params[$n] = $v;
                }
                else
                {
                    $placeholders[] = ':' . $name;
                    $params[':' . $name] = $value;
                }
            }
            $sql='INSERT INTO ' . $this->_connection->quoteTableName($table)
                . ' (' . implode(', ',$names) . ') VALUES ('
                . implode(', ', $placeholders) . ')';
            return $this->setText($sql)->execute($params);
        }
        public function update($table, $columns, $conditions='', $params=array())
        {
            $lines=array();
            foreach($columns as $name=>$value)
            {
                if($value instanceof CDbExpression)
                {
                    $lines[]=$this->_connection->quoteColumnName($name)    .    '='    .
$value->expression;
                    foreach($value->params as $n => $v)
                        $params[$n] = $v;
                }
                else
                {
                    $lines[]=$this->_connection->quoteColumnName($name) . '=:' . $name;
                    $params[':' . $name]=$value;
                }
            }
            $sql='UPDATE ' . $this->_connection->quoteTableName($table) . ' SET ' . implode(', ',
$lines);
            if(($where=$this->processConditions($conditions))!='')
                $sql.=' WHERE '.$where;
            return $this->setText($sql)->execute($params);
        }
        public function delete($table, $conditions='', $params=array())
        {
            $sql='DELETE FROM ' . $this->_connection->quoteTableName($table);
            if(($where=$this->processConditions($conditions))!='')
                $sql.=' WHERE '.$where;
            return $this->setText($sql)->execute($params);
        }
        public function createTable($table, $columns, $options=null)
```

```
	{
		return $this->setText($this->getConnection()->getSchema()->createTable($table, 
$columns, $options))->execute();
	}
	public function renameTable($table, $newName)
	{
		return $this->setText($this->getConnection()->getSchema()->renameTable($table, 
$newName))->execute();
	}
	public function dropTable($table)
	{
		return 
$this->setText($this->getConnection()->getSchema()->dropTable($table))->execute();
	}
	public function truncateTable($table)
	{
		$schema=$this->getConnection()->getSchema();
		$n=$this->setText($schema->truncateTable($table))->execute();
		if(strncasecmp($this->getConnection()->getDriverName(),'sqlite',6)===0)
			$schema->resetSequence($schema->getTable($table));
		return $n;
	}
	public function addColumn($table, $column, $type)
	{
		return $this->setText($this->getConnection()->getSchema()->addColumn($table, 
$column, $type))->execute();
	}
	public function dropColumn($table, $column)
	{
		return $this->setText($this->getConnection()->getSchema()->dropColumn($table, 
$column))->execute();
	}
	public function renameColumn($table, $name, $newName)
	{
		return $this->setText($this->getConnection()->getSchema()->renameColumn($table, 
$name, $newName))->execute();
	}
	public function alterColumn($table, $column, $type)
	{
		return $this->setText($this->getConnection()->getSchema()->alterColumn($table, 
$column, $type))->execute();
	}
	public function addForeignKey($name, $table, $columns, $refTable, $refColumns, 
$delete=null, $update=null)
```

```
	{
		return	$this->setText($this->getConnection()->getSchema()->addForeignKey($name,
$table, $columns, $refTable, $refColumns, $delete, $update))->execute();
	}
	public function dropForeignKey($name, $table)
	{
		return	$this->setText($this->getConnection()->getSchema()->dropForeignKey($name,
$table))->execute();
	}
	public function createIndex($name, $table, $column, $unique=false)
	{
		return	$this->setText($this->getConnection()->getSchema()->createIndex($name,
$table, $column, $unique))->execute();
	}
	public function dropIndex($name, $table)
	{
		return	$this->setText($this->getConnection()->getSchema()->dropIndex($name,
$table))->execute();
	}
	private function processConditions($conditions)
	{
		if(!is_array($conditions))
			return $conditions;
		elseif($conditions===array())
			return '';
		$n=count($conditions);
		$operator=strtoupper($conditions[0]);
		if($operator==='OR' || $operator==='AND')
		{
			$parts=array();
			for($i=1;$i<$n;++$i)
			{
				$condition=$this->processConditions($conditions[$i]);
				if($condition!=='')
					$parts[]='('.$condition.')';
			}
			return $parts===array() ? '' : implode(' '.$operator.' ', $parts);
		}
		if(!isset($conditions[1],$conditions[2]))
			return '';
		$column=$conditions[1];
		if(strpos($column,'(')===false)
			$column=$this->_connection->quoteColumnName($column);
		$values=$conditions[2];
		if(!is_array($values))
			$values=array($values);
		if($operator==='IN' || $operator==='NOT IN')
		{
			if($values===array())
				return $operator==='IN' ? '0=1' : '';
			foreach($values as $i=>$value)
			{
				if(is_string($value))
					$values[$i]=$this->_connection->quoteValue($value);
				else
					$values[$i]=(string)$value;
			}
			return $column.' '.$operator.' ('.implode(', ',$values).')';
		}
		if($operator==='LIKE'   ||   $operator==='NOT   LIKE'   ||   $operator==='OR   LIKE'   ||
$operator==='OR NOT LIKE')
		{
			if($values===array())
				return $operator==='LIKE' || $operator==='OR LIKE' ? '0=1' : '';
			if($operator==='LIKE' || $operator==='NOT LIKE')
				$andor=' AND ';
			else
			{
				$andor=' OR ';
				$operator=$operator==='OR LIKE' ? 'LIKE' : 'NOT LIKE';
			}
			$expressions=array();
			foreach($values as $value)
				$expressions[]=$column.'                                   '.$operator.'
'.$this->_connection->quoteValue($value);
			return implode($andor,$expressions);
		}
		throw    new    CDbException(Yii::t('yii',    'Unknown    operator    "{operator}".',
array('{operator}'=>$operator)));
	}
	private function joinInternal($type, $table, $conditions='', $params=array())
	{
		if(strpos($table,'(')===false)
		{
			if(preg_match('/^(.*?)(?i:\s+as\s+|\s+)(.*)$/',$table,$matches))   // with alias
				$table=$this->_connection->quoteTableName($matches[1]).'
'.$this->_connection->quoteTableName($matches[2]);
			else
```

```
                $table=$this->_connection->quoteTableName($table);
            }
            $conditions=$this->processConditions($conditions);
            if($conditions!='')
                $conditions=' ON '.$conditions;
            if(isset($this->_query['join']) && is_string($this->_query['join']))
                $this->_query['join']=array($this->_query['join']);
            $this->_query['join'][]=strtoupper($type) . ' ' . $table . $conditions;
            foreach($params as $name=>$value)
                $this->params[$name]=$value;
            return $this;
        }
        public function addPrimaryKey($name,$table,$columns)
        {
            return
$this->setText($this->getConnection()->getSchema()->addPrimaryKey($name,$table,$columns))-
>execute();
        }
        public function dropPrimaryKey($name,$table)
        {
            return
$this->setText($this->getConnection()->getSchema()->dropPrimaryKey($name,$table))->execute(
);
        }
}
class CDbColumnSchema extends CComponent
{
        public $name;
        public $rawName;
        public $allowNull;
        public $dbType;
        public $type;
        public $defaultValue;
        public $size;
        public $precision;
        public $scale;
        public $isPrimaryKey;
        public $isForeignKey;
        public $autoIncrement=false;
        public $comment='';
        public function init($dbType, $defaultValue)
        {
            $this->dbType=$dbType;
            $this->extractType($dbType);
        $this->extractLimit($dbType);
        if($defaultValue!==null)
                $this->extractDefault($defaultValue);
}
protected function extractType($dbType)
{
        if(stripos($dbType,'int')!==false && stripos($dbType,'unsigned int')===false)
                $this->type='integer';
        elseif(stripos($dbType,'bool')!==false)
                $this->type='boolean';
        elseif(preg_match('/(real|floa|doub)/i',$dbType))
                $this->type='double';
        else
                $this->type='string';
}
protected function extractLimit($dbType)
{
        if(strpos($dbType,'(') && preg_match('/\((.*)\)/',$dbType,$matches))
        {
                $values=explode(',',$matches[1]);
                $this->size=$this->precision=(int)$values[0];
                if(isset($values[1]))
                        $this->scale=(int)$values[1];
        }
}
protected function extractDefault($defaultValue)
{
        $this->defaultValue=$this->typecast($defaultValue);
}
public function typecast($value)
{
        if(gettype($value)===$this->type || $value===null || $value instanceof CDbExpression)
                return $value;
        if($value==='' && $this->allowNull)
                return $this->type==='string' ? '' : null;
        switch($this->type)
        {
                case 'string': return (string)$value;
                case 'integer': return (integer)$value;
                case 'boolean': return (boolean)$value;
                case 'double':
                default: return $value;
        }
}
```

```
}
class CSqliteColumnSchema extends CDbColumnSchema
{
	protected function extractDefault($defaultValue)
	{
		$this->defaultValue=$this->typecast(strcasecmp($defaultValue,'null')  ?  $defaultValue  :
null);
		if($this->type==='string' && $this->defaultValue!==null) // PHP 5.2.6 adds single quotes
while 5.2.0 doesn't
			$this->defaultValue=trim($this->defaultValue,"'\"");
	}
}
abstract class CValidator extends CComponent
{
	public static $builtInValidators=array(
		'required'=>'CRequiredValidator',
		'filter'=>'CFilterValidator',
		'match'=>'CRegularExpressionValidator',
		'email'=>'CEmailValidator',
		'url'=>'CUrlValidator',
		'unique'=>'CUniqueValidator',
		'compare'=>'CCompareValidator',
		'length'=>'CStringValidator',
		'in'=>'CRangeValidator',
		'numerical'=>'CNumberValidator',
		'captcha'=>'CCaptchaValidator',
		'type'=>'CTypeValidator',
		'file'=>'CFileValidator',
		'default'=>'CDefaultValueValidator',
		'exist'=>'CExistValidator',
		'boolean'=>'CBooleanValidator',
		'safe'=>'CSafeValidator',
		'unsafe'=>'CUnsafeValidator',
		'date'=>'CDateValidator',
	);
	public $attributes;
	public $message;
	public $skipOnError=false;
	public $on;
	public $except;
	public $safe=true;
	public $enableClientValidation=true;
	abstract protected function validateAttribute($object,$attribute);
	public static function createValidator($name,$object,$attributes,$params=array())
```

```
{
        if(is_string($attributes))
                $attributes=preg_split('/[\s,]+/',$attributes,-1,PREG_SPLIT_NO_EMPTY);
        if(isset($params['on']))
        {
                if(is_array($params['on']))
                        $on=$params['on'];
                else
                        $on=preg_split('/[\s,]+/',$params['on'],-1,PREG_SPLIT_NO_EMPTY);
        }
        else
                $on=array();
        if(isset($params['except']))
        {
                if(is_array($params['except']))
                        $except=$params['except'];
                else
                        $except=preg_split('/[\s,]+/',$params['except'],-1,PREG_SPLIT_NO_EMPTY);
        }
        else
                $except=array();
        if(method_exists($object,$name))
        {
                $validator=new CInlineValidator;
                $validator->attributes=$attributes;
                $validator->method=$name;
                if(isset($params['clientValidate']))
                {
                        $validator->clientValidate=$params['clientValidate'];
                        unset($params['clientValidate']);
                }
                $validator->params=$params;
                if(isset($params['skipOnError']))
                        $validator->skipOnError=$params['skipOnError'];
        }
        else
        {
                $params['attributes']=$attributes;
                if(isset(self::$builtInValidators[$name]))
                        $className=Yii::import(self::$builtInValidators[$name],true);
                else
                        $className=Yii::import($name,true);
                $validator=new $className;
                foreach($params as $name=>$value)
                $validator->$name=$value;
            }
            $validator->on=empty($on) ? array() : array_combine($on,$on);
            $validator->except=empty($except) ? array() : array_combine($except,$except);
            return $validator;
        }
        public function validate($object,$attributes=null)
        {
            if(is_array($attributes))
                $attributes=array_intersect($this->attributes,$attributes);
            else
                $attributes=$this->attributes;
            foreach($attributes as $attribute)
            {
                if(!$this->skipOnError || !$object->hasErrors($attribute))
                    $this->validateAttribute($object,$attribute);
            }
        }
        public function clientValidateAttribute($object,$attribute)
        {
        }
        public function applyTo($scenario)
        {
            if(isset($this->except[$scenario]))
                return false;
            return empty($this->on) || isset($this->on[$scenario]);
        }
        protected function addError($object,$attribute,$message,$params=array())
        {
            $params['{attribute}']=$object->getAttributeLabel($attribute);
            $object->addError($attribute,strtr($message,$params));
        }
        protected function isEmpty($value,$trim=false)
        {
            return $value===null || $value===array() || $value==='' || $trim && is_scalar($value)
&& trim($value)==='';
        }
}
class CStringValidator extends CValidator
{
        public $max;
        public $min;
        public $is;
        public $tooShort;
```

```
	public $tooLong;
	public $allowEmpty=true;
	public $encoding;
	protected function validateAttribute($object,$attribute)
	{
		$value=$object->$attribute;
		if($this->allowEmpty && $this->isEmpty($value))
			return;
		if(function_exists('mb_strlen') && $this->encoding!==false)
			$length=mb_strlen($value,	$this->encoding	?	$this->encoding	:
Yii::app()->charset);
		else
			$length=strlen($value);
		if($this->min!==null && $length<$this->min)
		{
			$message=$this->tooShort!==null?$this->tooShort:Yii::t('yii','{attribute}	is	too
short (minimum is {min} characters).');
			$this->addError($object,$attribute,$message,array('{min}'=>$this->min));
		}
		if($this->max!==null && $length>$this->max)
		{
			$message=$this->tooLong!==null?$this->tooLong:Yii::t('yii','{attribute}  is  too  long
(maximum is {max} characters).');
			$this->addError($object,$attribute,$message,array('{max}'=>$this->max));
		}
		if($this->is!==null && $length!==$this->is)
		{
			$message=$this->message!==null?$this->message:Yii::t('yii','{attribute}   is   of   the
wrong length (should be {length} characters).');
			$this->addError($object,$attribute,$message,array('{length}'=>$this->is));
		}
	}
	public function clientValidateAttribute($object,$attribute)
	{
		$label=$object->getAttributeLabel($attribute);
		if(($message=$this->message)===null)
			$message=Yii::t('yii','{attribute}   is   of   the   wrong   length   (should   be   {length}
characters).');
		$message=strtr($message, array(
			'{attribute}'=>$label,
			'{length}'=>$this->is,
		));
		if(($tooShort=$this->tooShort)===null)
			$tooShort=Yii::t('yii','{attribute} is too short (minimum is {min} characters).');
		$tooShort=strtr($tooShort, array(
			'{attribute}'=>$label,
			'{min}'=>$this->min,
		));
		if(($tooLong=$this->tooLong)===null)
			$tooLong=Yii::t('yii','{attribute} is too long (maximum is {max} characters).');
		$tooLong=strtr($tooLong, array(
			'{attribute}'=>$label,
			'{max}'=>$this->max,
		));
		$js='';
		if($this->min!==null)
		{
			$js.="
if(value.length<{$this->min}) {
	messages.push(".CJSON::encode($tooShort).");
}
";
		}
		if($this->max!==null)
		{
			$js.="
if(value.length>{$this->max}) {
	messages.push(".CJSON::encode($tooLong).");
}
";
		}
		if($this->is!==null)
		{
			$js.="
if(value.length!={$this->is}) {
	messages.push(".CJSON::encode($message).");
}
";
		}
		if($this->allowEmpty)
		{
			$js="
if(jQuery.trim(value)!='') {
	$js
}
";
		}
		return $js;
```

```
        }
}
class CRequiredValidator extends CValidator
{
        public $requiredValue;
        public $strict=false;
        protected function validateAttribute($object,$attribute)
        {
                $value=$object->$attribute;
                if($this->requiredValue!==null)
                {
                        if(!$this->strict      &&      $value!=$this->requiredValue      ||      $this->strict      &&
$value!==$this->requiredValue)
                        {
                                $message=$this->message!==null?$this->message:Yii::t('yii','{attribute}   must
be {value}.',
                                        array('{value}'=>$this->requiredValue));
                                $this->addError($object,$attribute,$message);
                        }
                }
                elseif($this->isEmpty($value,true))
                {
                        $message=$this->message!==null?$this->message:Yii::t('yii','{attribute}  cannot  be
blank.');
                        $this->addError($object,$attribute,$message);
                }
        }
        public function clientValidateAttribute($object,$attribute)
        {
                $message=$this->message;
                if($this->requiredValue!==null)
                {
                        if($message===null)
                                $message=Yii::t('yii','{attribute} must be {value}.');
                        $message=strtr($message, array(
                                '{value}'=>$this->requiredValue,
                                '{attribute}'=>$object->getAttributeLabel($attribute),
                        ));
                        return "
if(value!=" . CJSON::encode($this->requiredValue) . ") {
        messages.push(".CJSON::encode($message).");
}
";
                }
```

```
			else
			{
				if($message===null)
					$message=Yii::t('yii','{attribute} cannot be blank.');
				$message=strtr($message, array(
					'{attribute}'=>$object->getAttributeLabel($attribute),
				));
				return "
if(jQuery.trim(value)=='') {
	messages.push(".CJSON::encode($message).");
}
";
			}
		}
}
class CNumberValidator extends CValidator
{
	public $integerOnly=false;
	public $allowEmpty=true;
	public $max;
	public $min;
	public $tooBig;
	public $tooSmall;
	public $integerPattern='/^\s*[+-]?\d+\s*$/';
	public $numberPattern='/^\s*[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?\s*$/';
	protected function validateAttribute($object,$attribute)
	{
		$value=$object->$attribute;
		if($this->allowEmpty && $this->isEmpty($value))
			return;
		if($this->integerOnly)
		{
			if(!preg_match($this->integerPattern,"$value"))
			{
				$message=$this->message!==null?$this->message:Yii::t('yii','{attribute}   must
be an integer.');
				$this->addError($object,$attribute,$message);
			}
		}
		else
		{
			if(!preg_match($this->numberPattern,"$value"))
			{
				$message=$this->message!==null?$this->message:Yii::t('yii','{attribute}   must
be a number.');
                $this->addError($object,$attribute,$message);
            }
        }
        if($this->min!==null && $value<$this->min)
        {
            $message=$this->tooSmall!==null?$this->tooSmall:Yii::t('yii','{attribute} is too
small (minimum is {min}).');
            $this->addError($object,$attribute,$message,array('{min}'=>$this->min));
        }
        if($this->max!==null && $value>$this->max)
        {
            $message=$this->tooBig!==null?$this->tooBig:Yii::t('yii','{attribute} is too big
(maximum is {max}).');
            $this->addError($object,$attribute,$message,array('{max}'=>$this->max));
        }
    }
    public function clientValidateAttribute($object,$attribute)
    {
        $label=$object->getAttributeLabel($attribute);
        if(($message=$this->message)===null)
            $message=$this->integerOnly ? Yii::t('yii','{attribute} must be an integer.') :
Yii::t('yii','{attribute} must be a number.');
        $message=strtr($message, array(
            '{attribute}'=>$label,
        ));
        if(($tooBig=$this->tooBig)===null)
            $tooBig=Yii::t('yii','{attribute} is too big (maximum is {max}).');
        $tooBig=strtr($tooBig, array(
            '{attribute}'=>$label,
            '{max}'=>$this->max,
        ));
        if(($tooSmall=$this->tooSmall)===null)
            $tooSmall=Yii::t('yii','{attribute} is too small (minimum is {min}).');
        $tooSmall=strtr($tooSmall, array(
            '{attribute}'=>$label,
            '{min}'=>$this->min,
        ));
        $pattern=$this->integerOnly ? $this->integerPattern : $this->numberPattern;
        $js="
if(!value.match($pattern)) {
    messages.push(".CJSON::encode($message).");
}
";
```

```
		if($this->min!==null)
		{
			$js.="
if(value<{$this->min}) {
	messages.push(".CJSON::encode($tooSmall).");
}
";
		}
		if($this->max!==null)
		{
			$js.="
if(value>{$this->max}) {
	messages.push(".CJSON::encode($tooBig).");
}
";
		}
		if($this->allowEmpty)
		{
			$js="
if(jQuery.trim(value)!='') {
	$js
}
";
		}
		return $js;
	}
}
class CListIterator implements Iterator
{
	private $_d;
	private $_i;
	private $_c;
	public function __construct(&$data)
	{
		$this->_d=&$data;
		$this->_i=0;
		$this->_c=count($this->_d);
	}
	public function rewind()
	{
		$this->_i=0;
	}
	public function key()
	{
			return $this->_i;
		}
		public function current()
		{
			return $this->_d[$this->_i];
		}
		public function next()
		{
			$this->_i++;
		}
		public function valid()
		{
			return $this->_i<$this->_c;
		}
}
interface IApplicationComponent
{
		public function init();
		public function getIsInitialized();
}
interface ICache
{
		public function get($id);
		public function mget($ids);
		public function set($id,$value,$expire=0,$dependency=null);
		public function add($id,$value,$expire=0,$dependency=null);
		public function delete($id);
		public function flush();
}
interface ICacheDependency
{
		public function evaluateDependency();
		public function getHasChanged();
}
interface IStatePersister
{
		public function load();
		public function save($state);
}
interface IFilter
{
		public function filter($filterChain);
}
interface IAction
```

```php
{
	public function getId();
	public function getController();
}
interface IWebServiceProvider
{
	public function beforeWebMethod($service);
	public function afterWebMethod($service);
}
interface IViewRenderer
{
	public function renderFile($context,$file,$data,$return);
}
interface IUserIdentity
{
	public function authenticate();
	public function getIsAuthenticated();
	public function getId();
	public function getName();
	public function getPersistentStates();
}
interface IWebUser
{
	public function getId();
	public function getName();
	public function getIsGuest();
	public function checkAccess($operation,$params=array());
	public function loginRequired();
}
interface IAuthManager
{
	public function checkAccess($itemName,$userId,$params=array());
	public function createAuthItem($name,$type,$description='',$bizRule=null,$data=null);
	public function removeAuthItem($name);
	public function getAuthItems($type=null,$userId=null);
	public function getAuthItem($name);
	public function saveAuthItem($item,$oldName=null);
	public function addItemChild($itemName,$childName);
	public function removeItemChild($itemName,$childName);
	public function hasItemChild($itemName,$childName);
	public function getItemChildren($itemName);
	public function assign($itemName,$userId,$bizRule=null,$data=null);
	public function revoke($itemName,$userId);
	public function isAssigned($itemName,$userId);
	public function getAuthAssignment($itemName,$userId);
	public function getAuthAssignments($userId);
	public function saveAuthAssignment($assignment);
	public function clearAll();
	public function clearAuthAssignments();
	public function save();
	public function executeBizRule($bizRule,$params,$data);
}
interface IBehavior
{
	public function attach($component);
	public function detach($component);
	public function getEnabled();
	public function setEnabled($value);
}
interface IWidgetFactory
{
	public function createWidget($owner,$className,$properties=array());
}
interface IDataProvider
{
	public function getId();
	public function getItemCount($refresh=false);
	public function getTotalItemCount($refresh=false);
	public function getData($refresh=false);
	public function getKeys($refresh=false);
	public function getSort();
	public function getPagination();
}
interface ILogFilter
{
	public function filter(&$logs);
}
?>
```