Кеш можно отключить в админке на странице System/Cache Management

**Создание модуля Magento:**

**1.** Создать директорию */app/code/local/DS/News*

**2.** Создать файл *etc/config.xml*

<?xml version="1.0" ?>

<config>

<modules>

<DS\_News>

<version>0.0.1</version>

</DS\_News>

</modules>

</config>

Общая структура конфигурации модуля :

<config>  
 <modules>.................</modules>  
 <frontend>

Все касательно frontend части магазина: роуты, макеты, переводы, обсерверы

</frontend>  
 <adminhtml>

Все касательно admin части магазина: макеты, переводы, обсерверы

</adminhtml>  
 <global>

<models></models>

<blocks></blocks>

<helpers></helpers>

<resourses></resourses>

<url\_rewrite></url\_rewrite >

<request\_rewrite></request\_rewrite>

<cache></cache>

<session></session>

<request></request>

<template></template>

<log></log>

<events></events>

</global>  
 <default>

Все касательно admin части магазина: макеты, переводы, обсерверы

</default>  
 <crontab>.................</crontab>  
 <admin>

Все касательно admin части магазина: роуты, макеты, переводы

</admin>  
</config>

**3.** Создать файл *DS\_News.xml* в директории */app/etc/modules/*

<?xml version="1.0" ?>

<config>

<modules>

<DS\_News>

<active>true</active>

<codePool>local</codePool>

</DS\_News>

</modules>

</config>

В результате создания данной структуры директорий и файлов, в системе будет зарегистрирован модуль DS\_News. Данный модуль можно увидеть в списке модулей в админке по пути System/Configuration/Advanced/Advanced/Disable Modules Output.

В пункте 1 создаётся директория /app/code/local/DS/News, в которой будет храниться основной код модуля: модели, контроллеры, хелперы и т.д…

1. **Block** — классы блоков, отвечают за рендеринг страниц
2. **controllers** — контроллеры принимают запросы
3. **etc** — тут всякие конфигурационные файлы
4. **Helper** — дополнительные классы помощники
5. **Model** — модели
6. **sql** — инсталяционные скрипты

В пункте 2 создаётся директория etc, в которой хранятся файлы конфигурации модуля, а также создаётся основной файл конфигурации config.xml, в котором пока указывается только версия модуля. В пункте 3 модуль регистрируется в системе.  
  
По факту, для задачи регистрации модуля самым важным является пункт 3 — модуль будет показан в админке даже при отсутствии директории модуля. Название файла DS\_News.xml в директории /app/etc/modules/ тоже не является обязательным — при загрузке система парсит все XML файлы из данной директории. Это может быть удобно при разработке в случае, когда устанавливаются несколько взаимосвязанных модулей — все модули можно указать в одном XML файле.  
  
Названия модуля DS\_News в xml файлах состоит из двух частей: [название директории namespace]\_[название директории модуля]. В связи с данным подходом именования, название директорий желательно составлять только из букв, и, также, важно учитывать регистр — название модуля в файлах конфигурации должно точно совпадать с названием директорий для корректной работы на \*nix системах.  
  
В файле /app/etc/modules/DS\_News.xml используются два тега: active и codePool. Первый тег отвечает за включение/отключение модуля, а второй — за местоположение в директории /app/code/: **core** (ядро системы), **community** (модули, разработанные сообществом Magento) или **local** (модули, разработанные другими разработчиками).

Подгрузка модулей довольно проста: Magento сперва грузит **app/etc/local.xml** с настройками БД, сессий, кэша и т.д., затем подгружает bootstraps модулей из **app/etc/modules/\*.xml**, сортирует их с учетом зависимостей в **<depends>**, подгружает **app/code/\*/\*/\*/etc/config.xml**, сливает все XML в один глобальный XML, и снова грузит app/etc/local.xml в глобальный XML, чтобы избежать утери данных из local.xml.  
  
По сути, смысл всей разработки в Magento заключается в том, чтобы вмешиваться в исходный код модулей ядра (или любого другого модуля) по минимуму, для этого в Magento существует несколько подходов:  
1) и, пожалуй, самый основной — события. Если нужно добавить/изменить какой-то существующий функционал, в большинстве случаев достаточно перехвата события.  
2) Rewrite класса  
3) Локальный override

app/code/core/Mage/Catalog модуль фронта

app/code/core/Mage/Adminhtml модуль бекенда

Секции конфигурации модуля:

**Создание моделей**

<config>

<modules>

<IGN\_Siteblocks>

<version>1.0.0</version>

</IGN\_Siteblocks>

</modules>

<global>

<models>

<siteblocks> <!-- Как правило тут namespace\_modulename или просто modulename -->

<class>IGN\_Siteblocks\_Model</class>

<resourceModel>siteblocks\_resource</resourceModel>

</siteblocks>

<siteblocks\_resource>

<class>IGN\_Siteblocks\_Resource</class>

<entities>

<block> <!-- наименование модели -->

<table>ign\_siteblock</table> <!-- название таблицы к которой будет "привязана" модель -->

</block>

</entities>

</siteblocks\_resource>

</models>

<resources>

<siteblocks\_setup> <!-- именно в папку с таким названием нужно складывать install и upgrade скрипты -->

<setup>

<module>IGN\_Siteblocks</module>

</setup>

</siteblocks\_setup>

</resources>

</global>

</config>

Важно определиться с названием префикса (не знаю какой термин тут подойдет лучше). Я выбрал siteblocks. Это произвольное название и как правило формируется из неймспейса и имя модуля или только имени модуля. Ну или для запутывания разработчиков, можно выбрать совершенно произвольную строку, заранее прикупив оберег от проклятий.  
  
Выбирайте четко и желательно без использования заглавных символов. Одна опечатка, и будете долго копаться, искать проблему. Название модели и привязка к таблице. Имя модели соответствует названию файла модели. Название таблицы в базе произвольное. В моем случае что бы обратиться к модели, нужно написать так:

Mage::getModel('siteblocks/block');

Теперь можно добавлять модели. Создадим модель Block. Для каждой, привязанной к таблице, модели нужно создавать 3 файла: модель, ресурсная модель, модель коллекции. Модель абстрагируется от работы с базой, ресурсные модели находятся уровнем ниже. Там мы реализуем логику фильтрации, сортировки, обработки данных до их сохранения и после загрузки из базы.  
Код модели Block.php:

class IGN\_Siteblocks\_Model\_Block extends Mage\_Core\_Model\_Abstract {

public function \_construct()

{

parent::\_construct();

$this->\_init('siteblocks/block'); //Все в соотвествии с указанными в config.xml параметрами

}

}

class IGN\_Siteblocks\_Model\_Resource\_Block extends

Mage\_Core\_Model\_Resource\_Db\_Abstract {

public function \_construct()

{

$this->\_init('siteblocks/block','block\_id'); //block\_id это наш PRIMARY KEY в таблице, по умолчанию entity\_id

}

}

-------------------------------------------------------------------------

class IGN\_Siteblocks\_Model\_Resource\_Block\_Collection extends Mage\_Core\_Model\_Resource\_Db\_Collection\_Abstract {

public function \_construct()

{

parent::\_construct();

$this->\_init('siteblocks/block');

}

}

Вместо Block может быть любое другое имя

Наши классы пусты, но они уже реализуют необходимый минимум функционала по наследованию.  
Добавлять в них код будем по необходимости. А если мы хотим добавить еще моделей, то добавляем еще одну привязку модели к таблице и новых 3 файла. Можно сколько угодно добавлять моделей, не привязанных к таблице (просто для реализации какого-то функционала), то просто добавляем новый файл, наследоваться от *Mage\_Core\_Model\_Abstract* не обязательно.  
  
Не забываем создать инсталяционный скрипт, который будет создавать таблицу для нашей модели.

Если вы уже пробовали зайти в админку, при установленом модуле, когда еще не было инсталяционного скрипта. Скорее всего ваш инсталл скрипт больше никогда не запустится. В этом случае необходимо найти и удалить запись siteblocks\_setup из таблицы core\_resource в базе магазина.  
  
При апгрейде версии модуля. Мы указываем новую версию в *config.xml*, например: *1.0.1*. И создаем апгрейд скрипт: *upgrade-1.0.0-1.0.1.php*. И в таком же духе при последующих апгрейдах.