Magento – Cache

Żeby wyłączyć w Magento Cache należy wykonać następującą modyfikację:

```
Admin Panel > System > Cache Management > Select All > Actions: Disable > Submit
```

Lokalizacja modułów

Wszystkie moduły znajdują się w katalogu:

```
ŚcieżkaDoMagento/app/code/
```

Katalog ten składa się z trzech podkatalogów:

- **core** – zarezerwowany dla bazowych modułów Magento. Nie należy umieszczać w nim własnych modułów w celu uniknięcia konfliktów z obecnymi oraz przyszłymi modułami Magento. Modyfikacja bazowych modułów bez dokładnej znajomości Magento może zakończyć się bardzo źle, np. uszkodzeniem Magento.
- **community** – znajdują się tutaj moduły dostarczane i rozwijane przez społeczność Magento. Są to moduły zainstalowane przez użytkownika za pomocą Magento Connect.
- **local** – miejsce na moduły rozwijane lokalnie (nie są dostępne dla społeczności Magento). Automatycznie żaden moduł nie jest tutaj instalowany, chyba że zostanie on umieszczony w tym katalogu przez użytkownika.

Powyższe podkatalogi powszechnie określa się jako: **codePools**.

Uwaga:
W naszej zainstalowanej instancji Magento brakuje katalogu local. Należy utworzyć ten katalog w celu tworzenia własnych modułów.

Struktura katalogu tworzonego modułu

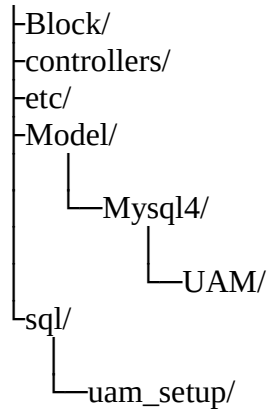
Przestrzeń nazw modułu
Pierwszym katalogiem jaki należy utworzyć jest *Namespace*, czyli przestrzeń nazw. Katalog ten można nazwać dowolnie, jednak istnieje pewna konwencja głosząca, że nazwą katalogu *Namespace* jest nazwa firmy (w całości lub skróconej formie) lub autor danego modułu. Magento jako przestrzeni nazw używa: „Mage”.

Nazwa modułu
Następnie tworzymy podkatalog (katalogu *Namespace*) o nazwie opisującej nasz moduł, np. jeżeli nasz produkt tworzy wpisy w logach za kazdym razem jak produkt jest zapisywany, to logiczną nazwą tego podkatalogu jest, np. LogProductUpdate.

Przykładowa ścieżka do naszego modułu:

```
ŚcieżkaDoMagento/app/code/local/UAM/LogProductUpdate
```

Pełna struktura katalogów rozwijanego modułu wygląda następująco (katalogi należy umieścić w katalogu naszego modułu, czyli np. w przykładowej ścieżce podanej powyżej):



Zazwyczaj deweloper moduły wybiera folder lub je wszystkie – w zależności od tego, czy będzie, czy nie będzie ich używał w swoim module.

Rozpiska katalogów wraz z opisem ich przeznaczenia:

- ***Block/*** - przetwarzenie wszystkich wyświetlonych bloków wywołanych przez system dla tego modułu. Są to kontrolery (ang. controllers), które będą wywołane przez pliki XML layoutu w danym motywie (ang. theme) w celu wyświetlenia zawartości.
- **controllers/** - nasze kontrolery, które obsługują aplikację oraz utrzymuwanie całości w strukturze.
- **etc/** - pliki konfiguracyjne modułu, które służą do zadeklarowania domyślnych opcji podczas instalacji oraz deklaracji wszystkich bloków, modeli i akcji instalacji/upgrade'u.
- **Model/** - umiejscowienie wszystkich modeli w celu wsparcia kontrolerów modułu.
- **sql/** - akcje SQL podczas instalowania/upgrade'u/odinstalowywania modułu.

Konfiguracja modułu

Pliki konfiguracyjne naszego modułu znajdują się w katalogu: **etc**
Utwórzmy w nim plik: **config.xml** który będzie informował Magento o położeniu plików modułu, jego wersji, zdarzeniach na jakie ma reagować, itp.

```
ŚcieżkaDoMagento/app/code/local/UAM/LogProductUpdate/etc/config.xml
```

Wygląd pliku z komentarzami opisującymi każdą sekcję:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Główny (ang. root) węzeł konfigurujący moduł Magento  -->
<config>

    <!--
        Węzeł modułu zawierający podstawowe informacje
        na temat każdego modułu Magento
    -->
    <modules>

        <!--
            Poniższy węzeł musi dokładnie odpowiadać nazwie katalogu
            przestrzeni nazw modułu i katalogowi nazwy modułu (separator
            między katalogami zastępuje się znakiem podkreślenia)
        -->
        <UAM_LogProductUpdate>

            <!-- Wersja naszego modułu, począwszy od 0.0.1 -->
            <version>0.0.1</version>

        </UAM_LogProductUpdate>

    </modules>

</config>
```

Aktywacja modułu

Następnie należy poinformować Magento o istnieniu naszego modułu. Do tego celu tworzymy nowy plik XML w ścieżce **…/app/etc/modules**
Nazwa wspomnianego pliku jest dowolna, ponieważ Magento zaznajamia się z treścią każdego pliku XML w tym katalogu. Niemniej jednak zgodnie z przyjętą konwencją nazwa pliku powinna odpowiadać nazwie tworzonego modułu – w naszym przypadku:

```
ŚcieżkaDoMagento/app/etc/modules/UAM_LogProductUpdate.xml
```

Plik ten powinien zawierać następującą treść:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <UAM_LogProductUpdate>

            <!-- Czy nasz moduł jest aktywny: true lub false -->
            <active>true</active>

            <!-- Który code pool używamy: core, community lub local -->
            <codePool>local</codePool>

        </UAM_LogProductUpdate>
    </modules>
</config>
```

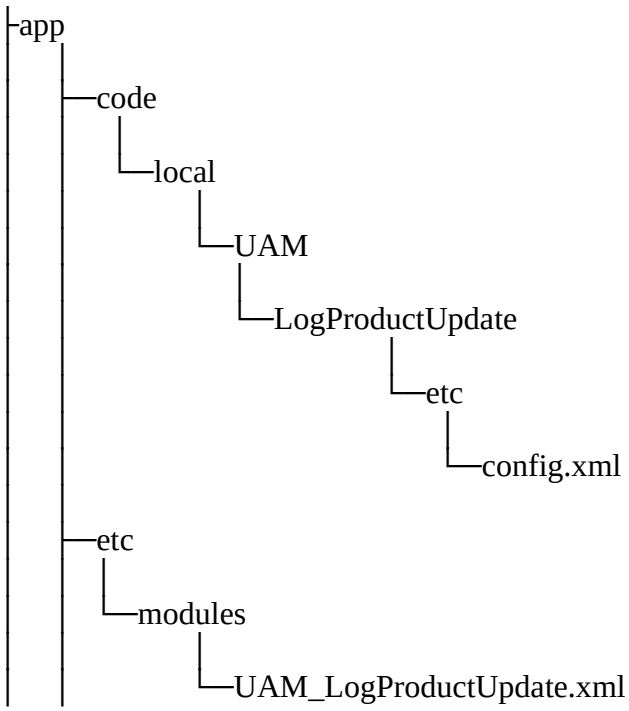
Sprawdzenie, czy moduł jest włączony
Posiadamy teraz w pełni funkcjonalny moduł (jeszcze nic nie robiący, ale w całości poprawny [ang. valid module]), który jest włączony w Magento. Czas sprawdzić, czy wszystko zostało poprawnie skonfigurowane.
W panelu administracyjnym Magento wejdź w:

```
System > Configuration > Advanced > Advanced
```

i rozwiń widoczny listing: **Disable Modules Output.** Powinniśmy widzieć na liście nasz moduł *UAM_LogProductUpdate* z opcją **enabled**.

Jeżeli moduł nie widnieje na liście, to znaczy, że coś zostało zrobione źle i należy przejść jeszcze raz przez powyższe kroki aż do tego momentu. Zazwyczaj w tym miejscu deweloperzy Magento odkrywają cache!

Struktura katalogów i plików naszego modułu powinna wyglądać teraz następująco:



Definiowanie obserwatora zdarzeń

Obserwatorzy zdarzeń stanowią niezwykle pomocny mechanizm i są jednym z najprostszych sposobów na rozszerzenie funkcjonalności Magento bez konieczności przepisywania lub zastępowani bazowej metody lub klasy.

Chcemy, aby nasz moduł obserwował zdarzenia, które Magento wywołuje bezpośrednio po zapisaniu/zaktualizowaniu produktu. Tak więc jesteśmy zainteresowani kodem takiego zdarzenia: **catalog_product_save_after**
W trakcie tworzenia nowego obserwatora ustalenie z jakiego kodu zdarzenia będziemy musieli skorzystać, wymaga podstawowej wiedzy z zakresy warstwy modelu Magento (temat nie zostanie poruszony w tym tutorialu).

Modyfikujemy plik (właściwie rozszerzamy): **config.xml** w celu zawarcia definicji obserwatora zdarzeń:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <UAM_LogProductUpdate>
            <version>0.0.1</version>
        </UAM_LogProductUpdate>
    </modules>

    <!-- Skonfigurowanie zachowania naszego modułu - zasięg globalny -->
    <global>

        <!-- Zdefiniowanie obserwatora zdarzeń -->
        <events>

            <!-- Kod zdarzenia, które chcemy obserwować -->
            <catalog_product_save_after>

                <!-- Ustawienie obserwatora na to zdarzenie -->
                <observers>

                    <!--
                        Unikalny identyfikator w węźle
                        catalog_product_save_after.
                        Zgodnie z konwencją, piszemy nazwę modułu
                        małymi literami.
                    -->
                    <uam_logproductupdate>

                        <!-- Model, który ma zostać uruchomiony -->
                        <class>uam_logproductupdate/observer</class>

                        <!-- Metoda klasy, która ma zostać wywołana -->
                        <method>logUpdate</method>

                        <!-- Rodzaj uruchamianej klasy -->
                        <type>singleton</type>

                    </uam_logproductupdate>

                </observers>

            </catalog_product_save_after>

        </events>

    </global>

</config>
```

Konfiguracja katalogu naszego modelu

W zdefiniowanym powyżej obserwatorze zdarzeń odnosiliśmy się do modelu, które nie zostały jeszcze utworzone. Informujemy Magento gdzie znajduje się model w naszym module poprzez aktualizację pliku: **config.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <modules>
        <UAM_LogProductUpdate>
            <version>0.0.1</version>
        </UAM_LogProductUpdate>
    </modules>

    <!-- Skonfigurowanie zachowania naszego modułu - zasięg globalny -->
    <global>

        <!-- Definiowanie modelu -->
        <models>

            <!--
                Unikalny identyfikator w węźle modułu.
                Zgodnie z konwencją, piszemy nazwę modułu
                małymi literami.
            -->
            <uam_logproductupdate>

                <!--
                    Ścieżka do katalogu naszego modelu (separator między
                    katalogami zastępuje się znakiem podkreślenia)
                -->
                <class>UAM_LogProductUpdate_Model</class>

            </uam_logproductupdate>

        </models>

        <events>
            <catalog_product_save_after>
                <observers>
                    <uam_logproductupdate>
                        <class>uam_logproductupdate/observer</class>
                        <method>logUpdate</method>
                        <type>singleton</type>
                    </uam_logproductupdate>
                </observers>
            </catalog_product_save_after>
        </events>

    </global>

</config>
```

Utworzenie modelu obserwatora

Utworzymy teraz model, który będzie uruchamiany w momencie wykrycia zdarzenia. W tym celu utworzymy w następującej lokalizacji plik PHP:

app/code/local/UAM/LogProductUpdate/Model/Observer.php

o następującej zawartości:

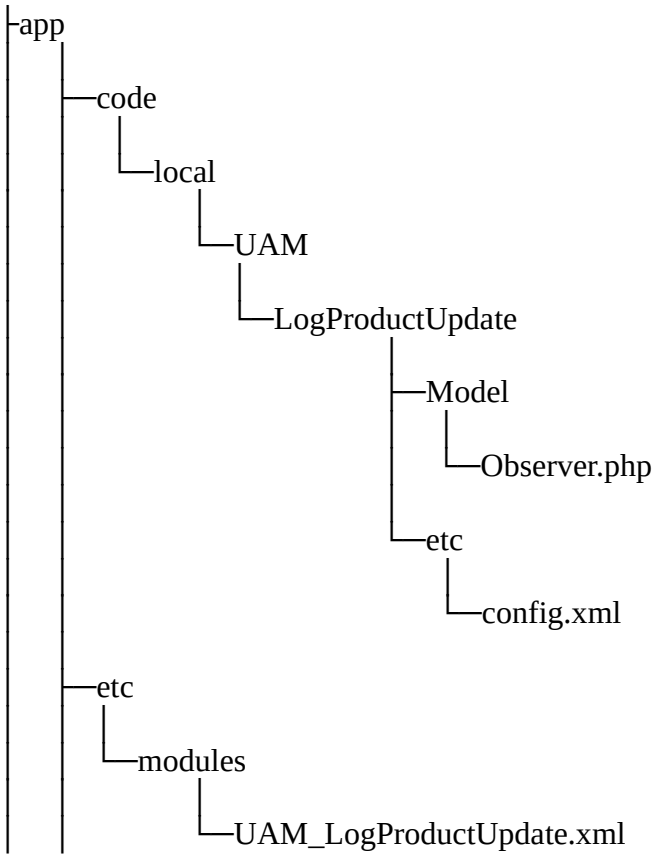
```
<?php
/**
 * Nasza nazwa klasy powinna odzwierciedlać strukturę katalogów naszego
 * modelu Observer.php począwszy od przestrzeni nazw
 * (separator między katalogami zastępuje się znakiem podkreślenia)
 * np. app/code/local/UAM/
 *                     LogProductUpdate/Model/Observer.php
 */
class UAM_LogProductUpdate_Model_Observer
{
    /**
     * Magento przekazuje obiekt Varien_Event_Observer jako
     * pierwszy parametr wysyłanych zdarzeń.
     */
    public function logUpdate(Varien_Event_Observer $observer)
    {
        // Przetwarzany produkt został zaktualizowany (informacja od
obserwatora zdarzeń)

        $product = $observer->getEvent()->getProduct();

        // Dopisanie logu do var/log/product-updates.log
        $name = $product->getName();
        $sku = $product->getSku();
        Mage::log(
            "{$name} ({$sku}) updated",
            null,
            'product-updates.log'
        );
    }
}
```

Moduł ukończony. Czas na testy

Struktura katalogów i plików naszego modułu powinna wyglądać teraz następująco:



Teraz, gdy nasz moduł jest już ukończony, możemy go przetestować. Zaloguj się do panelu administracyjnego Magento. Następnie utwórz lub zaktualizuj produkt w katalogu, a następnie sprawdź plik: **product-updates.log** w folderze: **var/log** w celu sprawdzenia, czy informacja o zaktualizowanym produkcie się pojawiła.

Jeśli nic się nie pojawiło lub katalog nie został utworzony, to należy upewnić się, że odpowiednie uprawnienia zostały ustawione (Magento posiada uprawnienia zapisu do tego katalogu) oraz że włączone jest opcja logów w Magento:

```
System > Configuration > Developer > Log Settings > Enabled
```

Kurs ten umożliwia ogólne poznanie działania modułów Magento. Po ukończeniu tego kursu, poświęć chwilę wolnego czasu na odkrywaniu modułów Magento znajdujących się w katalogu:

```
app/code/core
```

i sprawdź, czy teraz masz lepsze pojęcie o tym jak to wszystko działa.

Źródło:
http://www.smashingmagazine.com/2012/03/01/basics-creating-magento-module/
oraz książka:
Magento 1.3 – PHP Developer's Guide (Wydawnictwa: Packt Publishing)