Mercury API: Table of Contents

Chapter 1: Installation Instructions

Mercury API is distributed as a packaged Magento Connect extension. This tgz package may be used to install the extension via Magento's Connect interface, or used to install the package manually.

Installing WSDL Cache Controller Class Override

Regardless of which installation method you choose, The WSDL caching feature of Mercury API must be installed manually, as it relies on a code pool class override which may conflict with your existing system configuration. To install this class override and enable the WSDL caching features, rename the following Mercury API file

```
app/code/community/Mage/Api/Model/Server/Adapter/Soap.php.override
```

to

```
app/code/community/Mage/Api/Model/Server/Adapter/Soap.php
```

If your system already has a community code pool override for this class file, or you have a local code pool override at

```
app/code/local/Mage/Api/Model/Server/Adapter/Soap.php
```

you will need to manually diff and merge these files to get the benefit of WSDL caching. If you require assistance please contact Pulse Storm support

http://www.pulsestorm.net/contact-us/

Manual Installation Instructions

1. Expand the contents of your Mercury API tgz package
2. Copy/Upload the contents of the "app" folder to the corresponding "app" folder of your Magento system
3. Ensure the uploaded files and folders have *nix permissions, ownership and groups identical to the rest of your Magento files and folder
4. Clear the Magento Cache. This may be done under System -> Cache Management
5. Log out of the Magento Admin Console Application to clear your admin ACL information caches in the session
6. Also, if you're using compilation, you'll need to re-compile your classes.
7. You may configure Theme Doctor at System -> Configuration -> Advanced -> Mercury

Chapter 2: Performant API Calls

Magento has a bit of a reputation for being a less than performant platform. Because of the heavy use of Java style design patterns and XML based configuration trees, Magento requires more server power than the average LAMP stack application.

However, even beyond Magento's standard performance profile, the Magento SOAP API has a reputation for being particularly slow. This manual section covers a few general techniques you can use to speed up your use of the Magento API.

WSDL Caching

Before we get to ways to improve the performance of Magento's SOAP api, there's a few steps we can take on the PHP side to speed things up. Before we can explain **that**, we need to explain a bit about how SOAP works.

For most web developers a SOAP API endpoint is just another web-service that needs to be called. The difference is using SOAP vs. say, an XML-RPC API or a restful API lies mainly in the semantics of its syntax. To a service **developer**, the story's a little bit different.

SOAP was, in many ways, a formal response to what the web community was doing with things like XML-RPC and informal HTTP services. It's focus, in large part, was bringing the rigor of a specification to what some computer scientists viewed as loosely defined and dangerous informal API systems.

One bit of rigor that's worth our attention is a SOAP's WSDL. WSDL stands for *Web Services Development Language*. Every SOAP based web services is supposed to have a WSDL endpoint that describes the methods of a particular soap service. You can view Magento's v1 WSL file by requesting the following URL

```
http://store.example.com/api/soap/?wsdl
```

This WSDL file is used by a variety of SOAP tool, **including SOAP Clients**. When a SOAP client implementation (such as PHP's SoapClient object) needs to make a call, it will download and check the WSDL to ensure the method exists and for hints as to how the return value should be serialized.

What this means, and the reason we're discussing it, is that every-time you make an HTTP request for a SOAP call, the PHP SoapClient code itself needs to make a request for the SOAP's WSDL file. So that's at least one additional HTTP request that needs to be made, which can easily end up impacting performance.

Fortunately, PHP has some mechanisms to deal with this performance issue. Specifically, WSDL requests can be cached with the following ini settings.

```
soap.wsdl_cache_enabled
soap.wsdl_cache_ttl
```

If you've ever been working with a SOAP API that seems to be ignoring changes you've made to your client code, this caching is often the culprit. To work around this caching confusing, some version of Magento ship with WSDL caching turned off. This increases the up-to-dateness of the WSDL, but is one of the things that will negatively impact the performance of the SOAP API.

Mercury API contains a code-pool override which allows you to automatically control the caching of WSDL files from the Magento System Configuration Admin Console. (See installation instructions for more information on enabling this functionality)

Ensuring that WSDL caching is on both for system **calling** the API and the Magento system serving the API can dramatically improve the performance of your Magento SOAP API calls.

Login Sessions

Whenever you're looking to improve the performance of a system, you want to look for the biggest wins first, and then move on to the micro-optimizations second. Because SOAP uses HTTP as a transport, small performance problems become larger problems as the number of HTTP requests increases.

Using the technique above allowed us to reduce the number of HTTP requests made by PHP during the request for, and serving of . Next, we'll look to reduce the number of HTTP requests made in a typical Magento API client script.

If you look at the example SOAP code on the Magento wiki

```
$client = new SoapClient('http://store.example.com/api/soap?wsdl');Pulse Storm LLC
// If somestuff requires api authentification,
// then get a session token
$session = $client->login('apiUser', 'apiKey');Pulse Storm LLC
$result = $client->call($session, 'somestuff.method');
```

you'll notice that there's **two** SOAP requests that need to happen. The first

```
$session = $client->login('apiUser', 'apiKey');
```

authenticates and authorizes a user for the API system. This call returns a session ID that's used in subsequent requests

```
$result = $client->call($session, 'somestuff.method');
```

That's **two** HTTP requests, even though we only want one to call one SOAP method. At first blush this may seem like an intractable problem. We need to login to the service to use it.

While that's true, it ignores one thing. Magento doesn't immediately discard the session ID after each request. If you cache this ID locally, subsequent calls to your API script can skip the login request.

```
$client = new SoapClient('http://store.example.xom/api/soap?wsdl');Pulse Storm LLC
if(!($session = file_get_contents('/safe/cache/location/session')))
{
    $session = $client->login('apiUser', 'apiKey');Pulse Storm LLC
    file_put_contents('/safe/cache/location/session',$session);
```

```
}Pulse Storm LLC
$result = $client->call($session, 'somestuff.method');
```

The above is a rather naive, and possibly insecure example, but it gets the point across. The first call to the above script would still make the two API requests, but subsequent calls to the same script would only make the request to the `call` method call, skipping the `login` method call.

Chapter 3. Configurable Product Resource

The configurable product resource fills a few holes in the API related to Magento's configurable products. Specifically, it allows you to assign a simple product as a member of a configurable product, as well as allow you to set a "price modifier" for any simple product product attached to a configurable product.

Method: `assign_simple_product_to_configurable`

A "Configurable Product" is a Magento product that allows users to create product objects where each option configuration resolves to an individual SKU/simple product. The `assign_simple_product_to_configurable` method allows you to add an additional simple product to the set associated with a configurable product.

Method Name: `assign_simple_product_to_configurable`

Arguments

1. int - The `entity_id` of the configurable product
2. int - The `entity_id` of the simple product

Returns boolean

Example Usage

```
$api_username   = 'api_docs';
$api_secret     = 'notsosecret';
$api_url        = 'http://store.example.com/api/soap/?wsdl';

$client         = new SoapClient($api_url);
$session_id     = $client->login($api_username,$api_secret);
$result         = $client->call($session_id,
                  'mercuryapi_configurable.assign_simple_product_to_configurable',array(
                  167,          //id of the configurable product
                  169           //id of the simple product
                  ));
```

Method: `assign_simple_products_to_configurable`

The `assign_simple_products_to_configurable` method server the sample purpose as `assign_simple_product_to_configurable`, except that is allows you to specify a list of simple product IDs to **replace** the current set of simple products associate with a configurable product.

Method Name: `assign_simple_products_to_configurable`

Arguments

1. int - The `entity_id` of the configurable product

2. array - The `entity_id`s of the simple products

Returns boolean

Example Usage

```
$api_username   = 'api_docs';
$api_secret     = 'notsosecret';
$api_url        = 'http://store.example.com/api/soap/?wsdl';

$client         = new SoapClient($api_url);
$session_id     = $client->login($api_username,$api_secret);
$result         = $client->call($session_id,
                  'mercuryapi_configurable.assign_simple_products_to_configurable',array(
                  167,              //id of the configurable product
                  array(168,169)    //ids of the simple product
                  ));
```

Method: `add_price_modifier_for_simple_product_option`

The price of as configurable product is baed on two things. First, the configurable product entity itself has a price data field. The forms the base price for a product. Then, each attribute of the simple product may have a price modifier associated with it.

When a store customer selects a particular configurable product, this price modifier (either a flat amount or a percentage) will be **added** to the base configurable product price to come up with a final product price. The `add_price_modifier_for_simple_product_option` method allows you to change the price modifier associated with a particular atribute of a configurable product.

Method Name: `add_price_modifier_for_simple_product_option`

Arguments

1. int - The `entity_id` of the configurable product
2. int - The `attribute_id`s of the attribute to add the price modifier to
3. string - The label of the specific attribute value this price modifier should apply to
4. float - The modifier, either a fixed value or percentage
5. boolean - optional default to false. Is true, the value of the fourth paramater will be treated as a percentage

Returns boolean

Example Usage

```
$api_username   = 'api_docs';
$api_secret     = 'notsosecret';
$api_url        = 'http://store.example.com/api/soap/?wsdl';

$client         = new SoapClient($api_url);
$session_id     = $client->login($api_username,$api_secret);

//using the sample data, this adds 10% to the price of a Man's shirt
$result         = $client->call($session_id,
```

```
'mercuryapi_configurable.add_price_modifier_for_simple_product_option',array(
167,   //id of the configurable product
501,   //id of the attribute, in this case the sample data's "Gender"
'Mens',//the label of teh specific attribute value we're setting a price for
10,    //10% price increase for Men's shirts
true   // is_percentage, ensure above is interpreted as a percent increase
));
```

Chapter 4: Pagination Resource

The Pagination Resource allows you to put the API into a "pagination mode". While in pagination mode, requests for Magento collections will return a limited number of rows. That is, the calls will return only a single "page" of objects.

This is most useful when fetching large record sets where a request for **all** the records would consume an unacceptable amount of server or clients resources, but fetching rows one at a time would produce results in an insufficient amount of time.

Method: `set_start_page_and_per_page`

The `set_start_page_and_per_page` method allows you to set the page number and the number of records "per page" that will be returned. These values will apply **only** to the current API session. Other users/sessionIDs of the API will not have their collections limited.

One pagination information is set, Magento will apply the page limits to any collection that issues either of the collection loading events

```
core_collection_abstract_load_beforePulse Storm LLC
eav_collection_abstract_load_before
```

This includes collections used by the Magento API `list` method. In addition to returning a paginated result, each returned row will increment the current page by one, allowing you to make successive calls to the `list` method (see code sample below). This is useful for "chunking" requests that may normally consume all available system memory.

Method Name: `set_start_page_and_per_page`

Arguments

1. int - starting page
2. int - number of records per page

Returns array - pagination session information

Example Usage

```
$api_username   = 'api_docs';
$api_secret     = 'notsosecret';
$api_url        = 'http://store.example.com/api/soap/?wsdl';

$client         = new SoapClient($api_url);
$session_id     = $client->login($api_username,$api_secret);
$result         = $client->call($session_id,'mercuryapi_pagination.set_start_page_and_per_page',array(
                  '1',      //starting page
                  '10'      //records per page
                  ));


//get first 10 productsPulse Storm LLC
```

```
$products       = $client->call($session_id, 'catalog_product.list',array());
foreach($products as $product)
{
    var_dump($product['product_id'] . "::" . $product['name']);
}

//get next 10 products, page has automatically incremented
$products       = $client->call($session_id, 'catalog_product.list',array());
foreach($products as $product)
{
    var_dump($product['product_id'] . "::" . $product['name']);
}
```

Method: `get_start_page_and_per_page`

The `get_start_page_and_per_page` method returns the currently set starting page, and per page, previously set by set_start_page_and_per_page.

Method Name: `get_start_page_and_per_page`

Arguments

- No Arguments

Returns array - ['cur*page*'=>'','*per*page'=>'']

Example Usage

```
$api_username   = 'api_docs';
$api_secret     = 'notsosecret';
$api_url        = 'http://store.example.com/api/soap/?wsdl';

$client         = new SoapClient($api_url);
$session_id     = $client->login($api_username,$api_secret);
$result         = $client->call($session_id,'mercuryapi_pagination.set_start_page_and_per_page',array(
                  '2','8'));

//re-use existing session idPulse Storm LLC
$result         = $client->call($session_id,'mercuryapi_pagination.get_start_page_and_per_page',array());Pulse Storm LLC

echo sprintf('The current page is %s, and %s records will be displayed on each page.',
$result['cur_page'],$result['per_page']
), "\n";
```

Method: `clear_page_information`

The `clear_page_information` method will "turn off" pagination for a particular session id. After a successful call to `clear_page_information` calls to the Magento API will revert to factory default behavior (in regards to pagination).

Method Name: `NAME111HERE`

Arguments

- No Arguments

Returns boolean

Example Usage

```
$api_username   = 'api_docs';
$api_secret     = 'notsosecret';
$api_url        = 'http://magento.example.com/api/soap/?wsdl';

$client         = new SoapClient($api_url);
$session_id     = $client->login($api_username,$api_secret);

//set the pagination information
$result         = $client->call($session_id,'mercuryapi_pagination.set_start_page_and_per_page',array(
                  '2','8'));

//re-use existing session id to get the pagination information
$result         = $client->call($session_id,'mercuryapi_pagination.get_start_page_and_per_page',array());Pulse Storm LLC

echo sprintf('The current page is %s, and %s records will be displayed on each page.',
$result['cur_page'],$result['per_page']
), "\n";

//clear pagination information for $session_idPulse Storm LLC
$result         = $client->call($session_id,'mercuryapi_pagination.clear_page_information',array());Pulse Storm LLC
var_dump($result);

//check that pagination information is cleared
$result         = $client->call($session_id,'mercuryapi_pagination.get_start_page_and_per_page',array());Pulse Storm LLC
var_dump($result);
```

Chapter 5: Pass Through Resource

The pass through resource allows a developer to bypass the business logic of the API and to fetch native Magento collections.

The Magento API was introduced, in part, to allow third developers a stable set of methods for performing common tasks associated with their system. This allows the Magento core team to change how the core PHP models work without worrying about maintaining backwards compatibility, so long as the API methods continue to perform the same tasks.

The trade-off is it's often not possible to performa a particular task in Magento via the API. The pass through resource seeks solves this problem by allowing you to fetch native collections.

Warning: This is for expert users only. By using the pass through method you're skipping any business logic encapsulated in the API. Also, just like writing native PHP code, it's up to you, the third party developer, to be aware of any changes in functionality version-to-version that might affect your script, as well as maintain any object-to-object relationships not automatically handled by Magento's ORM.

Method: `fetch_collection`

The `fetch_collection` method allows you to directly fetch a specific Magento model collection.

Remeber, optional paramaters may be skipped by passing in a boolean `false`.

Method Name: `fetch_collection`

Arguments

1. string - class alias for the collection **resource** model
2. string - optional column name to sort collection by
3. string - optional direction to sort the collection by (ASC|DESC)
4. array - optional an "associative array" of key/values to filter the collection
5. int - optional the page number to start the collection on
6. int - optional the size of each page returned by the collection

Example Usage

```
$api_username   = 'api_docs';
$api_secret     = 'notsosecret';
$api_url        = 'http://magento.example.com/api/soap/?wsdl';

$new_page_vals  = array('title'=>'Enable Cookies, Please',
                        'root_template'=>'one_column',);

$client         = new SoapClient($api_url);
$session_id     = $client->login($api_username,$api_secret);
$result         = $client->call($session_id,
                  'mercuryapi_passthrough.fetch_collection',array(
                  'cms/page_collection',                              //class alias of **resource** model to fetch
```

```
                'title',                                                    //column to sort collection by
                'ASC',                                                      // direction to sort collection
                array('page_id'=>array('in'=>array(1,2,3,4,5,6,7))),        //key/value column/filters
                1,
                5
                ));

foreach($result as $item)
{
    var_dump($item['title'].'::'.$item['page_id']);
}
```

Chapter 6: Configuration Options

The Mercury API configuration options are grouped under the

```
System -> Configuration -> Advanced -> Mercury API
```

in the Magento Admin Console.

Cache WSDL

Mercury API ships with a class override for the `Mage_Api_Model_Server_Adapter_Soap` which allows you turn PHP's WSDL caching on or off. Assuming you're using this override, the **Cache WSDL** setting allows you to turn this feature on and off.

WSDL TTL

Similar to the above, the configuration setting allows you to set, in seconds, PHP "soap.wsdl_cache_ttl" ini setting.