**Difference between the manager and the locator**

Many people ask questions about the difference between the service locator and the service manager. The service locator (or SL) is an interface which is very slim:

namespace Zend\ServiceManager;

interface ServiceLocatorInterface

{

public function get($name);

public function has($name);

}

The service manager is a service locator implementation. By default the Zend Framework 2 implementation of the SL is the SM. Throughout the framework you see sometimes getServiceLocator() methods and sometimes getServiceManager() methods. For getServiceLocator(), you get the SL returned and for getServiceManager() you explicitly ask for the SM implementation.

It is not a big difference at this moment, since both methods will return the same object. However you can choose to have a different SL implementation. You keep yourself to the SL contract, but several zf2 components still need the specific SM implementation.

**Configuration of the service manager**

The service managers can be configured in two ways: the module class can return the SM config and the module configuration file (config/module.config.php in most cases) can return SM config. Both result in the exact same service config so it is only a matter of taste where you would like to put the config.

You can add services in either of these ways:

/\*\*

\* With the module class

\*/

namespace MyModule;

class Module

{

public function getServiceConfig()

{

return array(

'invokables' => array(

'my-foo' => 'MyModule\Foo\Bar',

),

);

}

}

/\*\*

\* With the module config

\*/

return array(

'service\_manager' => array(

'invokables' => array(

'my-foo' => 'MyModule\Foo\Bar'

),

),

);

As you see, for both methods the content of the array is the same.

### Invokables

An invokable is a string that contains a fully qualified class name. Fully qualified means that namespaces are included. When the service manager receives a request for an invokable, it instantiates the fully qualified class and returns the object. Below are examples of how an invokable can be added to the service manager as well as how to use it.

// Add invokable to service manager

$serviceManager->setInvokableClass('user\_mapper', 'User\Mapper\UserMapper');

In the above line of code, an invokable with the name user\_mapper is added to the service manager instance. When this invokable is invoked, an instance of the User\Mapper\UserMapper class is returned. This is demonstrated below.

// Use invokable to retrieve an object

$userMapper = $serviceManager->get('user\_mapper');

The service manager’s get method is a generic method for obtaining a service, independent of the way the service is fetched. Thus, it is used regardless of whether the service is fetched by using an invokable or factory, for instance. What happens if the key matches an invokable is simply that the invokable’s class is instantiated. When the invokable was added to the service manager, it specified the User\Mapper\User class. This value is looked up and used to instantiate the class, like shown below.

// $className contains 'User\Mapper\UserMapper'

return new $className();

### Factories

Another approach is to use so-called factories, which are used to perform any setup or dependency injection for the requested object.

A factory can be either a PHP callable, an object or a fully qualified class name. If an object or a class name is provided, the class must implement Zend\ServiceManager\FactoryInterface, which defines a createService method. This method takes a ServiceLocatorInterface instance as its sole parameter, meaning that the service locator can be used to resolve dependencies needed to instantiate the object. The method is called by the service manager and should return the object that it instantiates. The same principles apply if a PHP callable is used instead.

Expanding the previous example, imagine that the UserMapper class depends upon a database adapter. By using dependency injection, a database adapter can be injected into its constructor. To accomplish this, a factory can be used instead of an invokable. An anonymous function — or closure — is specified, which is responsible for instantiating the requested service along with its dependencies.

$serviceManager->setFactory('user\_mapper', function($serviceManager) {

$dbAdapter = $serviceManager->get('Zend\Db\Adapter\Adapter');

return new \User\Mapper\UserMapper($dbAdapter);

});

$userMapper = $serviceManager->get('user\_mapper'); // Get user mapper

In this particular example, the service manager is used to obtain a database adapter so that it can be injected into the user mapper’s constructor. The user mapper is then returned and is ready for use.