Module Methods

Table of Contents

- 1 Required Methods
  - 1.1 getName()
  - 1.2 getVersion()
  - 1.3 getAuthors()
  - 1.4 getServiceName($service)
  - 1.5 getPackageServiceName($package, array $vars = null)
  - 1.6 moduleRowName()
  - 1.7 moduleRowNamePlural()
  - 1.8 moduleGroupName()
  - 1.9 moduleRowMetaKey()
- 2 Registrar Methods
  - 2.1 checkTransferAvailability($domain, $module_row_id = null)
  - 2.2 checkAvailability($domain, $module_row_id = null)
  - 2.3 getDomainContacts($domain, $module_row_id = null)
  - 2.4 setDomainContacts($domain, array $vars = [], $module_row_id = null)
  - 2.5 getDomainInfo($domain, $module_row_id = null)
  - 2.6 getDomainIsLocked($domain, $module_row_id = null)
  - 2.7 getDomainNameServers($domain, $module_row_id = null)
  - 2.8 setDomainNameservers($domain, $module_row_id = null, array $vars = [])
  - 2.9 lockDomain($domain, $module_row_id = null)
  - 2.10 unlockDomain($domain, $module_row_id = null)
  - 2.11 registerDomain($domain, $module_row_id = null, array $vars = [])
  - 2.12 transferDomain($domain, $module_row_id = null, array $vars = [])
  - 2.13 resendTransferEmail($domain, $module_row_id = null)
  - 2.14 sendEppEmail($domain, $module_row_id = null)
  - 2.15 updateEppCode($domain, $epp_code, $module_row_id = null, array $vars = [])
  - 2.16 renewDomain($domain, $module_row_id = null, array $vars = [])
  - 2.17 restoreDomain($domain, $module_row_id = null, array $vars = [])
  - 2.18 getServiceDomain($service)
  - 2.19 getExpirationDate($service, $format = 'Y-m-d H:i:s')
  - 2.20 getTlds($module_row_id = null)
  - 2.21 getTldPricing($module_row_id = null)
- 3 Optional Methods
  - 3.1 install/upgrade/uninstall()
  - 3.2 cron($key)
  - 3.3 getDescription()
  - 3.4 getLogo()
  - 3.5 validateService($package, array $vars=null)
  - 3.6 addService($package, array $vars=null, $parent_package=null, $parent_service=null, $status="pending")
  - 3.7 editService($package, $service, array $vars=array(), $parent_package=null, $parent_service=null)
  - 3.8 cancelService($package, $service, $parent_package=null, $parent_service=null)
  - 3.9 suspendService($package, $service, $parent_package=null, $parent_service=null)
  - 3.10 unsuspendService($package, $service, $parent_package=null, $parent_service=null)
  - 3.11 renewService($package, $service, $parent_package=null, $parent_service=null)
  - 3.12 changeServicePackage($package_from, $package_to, $service, $parent_package=null, $parent_service=null)
  - 3.13 addPackage(array $vars=null)
  - 3.14 editPackage($package, array $vars=null)
  - 3.15 deletePackage($package)
  - 3.16 manageModule($module, array &$vars)
  - 3.17 manageAddRow(array &$vars)
  - 3.18 manageEditRow($module_row, array &$vars)
  - 3.19 addModuleRow(array &$vars)
  - 3.20 editModuleRow($module_row, array &$vars)
  - 3.21 deleteModuleRow($module_row)
  - 3.22 getGroupOrderOptions()
  - 3.23 selectModuleRow($module_group_id)
  - 3.24 getPackageFields($vars=null)
  - 3.25 getEmailTags()
  - 3.26 getAdminAddFields($package, $vars=null)
  - 3.27 getClientAddFields($package, $vars=null)
  - 3.28 getAdminEditFields($package, $vars=null)
  - 3.29 getAdminServiceInfo($service, $package)
  - 3.30 getClientServiceInfo($service, $package)
  - 3.31 getAdminTabs($package)
  - 3.32 getClientTabs($package)
  - 3.33 getAdminServiceTabs($service)
  - 3.34 getClientServiceTabs($service)

Required Methods

As of version 3.1 of Blesta, all of these required methods can be defined in a module configuration file instead.

The following methods are required to be implemented for each module.

getName()

The getName() method simply returns the name of the module. It's always best to define any language in your module using language files (see Translating Blesta for more information).

```
class MyModule extends Module {
...
        public function getName() {
                return Language::_("MyModule.name", true);
        }
...
}
```

getVersion()

This method must return the current version number of the module. Upon installation or upgrade, Blesta records the current code version so that it can tell when an upgrade/downgrade is available. The version number should be compatible with PHP's version_compare() function.

```
class MyModule extends Module {
        const VERSION = "1.0.0";
...
        public function getVersion() {
                return self::VERSION;
        }
...

}
```

getAuthors()

This method returns an array containing information about the authors of the module.

```
class MyModule extends Module {
    private static $authors = array(array('name' => "Phillips Data, Inc.", 'url' =>"http://www.blesta.com"));
...
        public function getAuthors() {
                return self::$authors;
        }
...
}
```

getServiceName($service)

Given a particular service, this method should return the label used to identify the service. For example, if the module were representing VOIP services, the value returned by this method might be the phone number associated with the VOIP service.

```
class MyModule extends Module {
...
        public function getServiceName($service) {
                $key = "phone_number";
        foreach ($service->fields as $field) {
            if ($field->key == $key)
                return $field->value;
        }
        return null;
        }
...
}
```

getPackageServiceName($package, array $vars = null)

Given a particular package and input data for the configuration, this method should return the label used to identify the service name prior to it becoming a service. This is similar to "getServiceName($service)". For example, if the module were representing VOIP services, the value returned by this method might be the phone number associated with the VOIP service.

```
class MyModule extends Module {
...
        public function getPackageServiceName($package, array $vars = null) {
                $key = "phone_number";
                 return (isset($vars[$key]) ? $vars[$key] : null);
        }
...
}
```

moduleRowName()

The moduleRowName() method returns the value used to represent a module row. For example, if the module were representing shared hosting services and each module row were a physical server the value returned by this method might be "Server".

```
class MyModule extends Module {
...
        public function moduleRowName() {
                return Language::_("MyModule.module_row", true);
        }
...
}
```

moduleRowNamePlural()

A plural representation of the moduleRowName() method (e.g. "Servers").

```
class MyModule extends Module {
...
        public function moduleRowNamePlural() {
                return Language::_("MyModule.module_row_plural", true);
        }
...
}
```

moduleGroupName()

The moduleGroupName() method returns the value used to represent a module group.

```
class MyModule extends Module {
...
        public function moduleGroupName() {
                return Language::_("MyModule.module_group", true);
        }
...
}
```

moduleRowMetaKey()

This method returns a string identifying the module meta key value that is used to identify module rows. This is used to identify module rows from one another. For example, servers may be identified by their host name.

```
class MyModule extends Module {
...
        public function moduleRowMetaKey() {
                return "host_name";
        }
...
}
```

Registrar Methods

The methods below are available for modules extending the RegistrarModule class.

checkTransferAvailability($domain, $module_row_id = null)

(optional) The checkTransferAvailability() method is called when an availability check is made for a domain transfer from an order form. It must return true if the domain is available for transfer or false otherwise. If this is not implemented then all domains will be considered available by the module.

```
class MyModule extends RegistrarModule {
...
        public function checkTransferAvailability($domain, $module_row_id = null)
    {
        // If the domain is available for registration, then it is not available for transfer
        return !$this->checkAvailability($domain, $module_row_id);
        }
...
}
```

checkAvailability($domain, $module_row_id = null)

(required) The checkAvailability() method is called when an availability check is made for a domain from an order form. It must return true if the domain is available or false otherwise. If this is not implemented then all domains will be considered available by the module.

```
class MyModule extends RegistrarModule {
...
        public function checkAvailability($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Check if the domain is available (this is dependent on the module's API, omitted here)
                $domain = $api->checkDomain($domain);

                return isset($domain->availability) ? $domain->availability : false;
        }
...
}
```

getDomainContacts($domain, $module_row_id = null)

(optional) Returns an array with all the contacts for a given domain.

```
class MyModule extends RegistrarModule {
...
    public function getDomainContacts($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Get the domain contacts (this is dependent on the module's API, omitted here)
                $contact = $api->getContact($domain);

                return [
            [
                'external_id' => $contact->id,
                'email' => $contact->email,
                'phone' => $contact->phone,
                'first_name' => $contact->firstName,
                'last_name' => $contact->lastName,
                'address1' => $contact->address1,
                'address2' => $contact->address2,
                'city' => $contact->city,
                'state' => $contact->state,
                'zip' => $contact->zip,
                'country' => $contact->country
            ]
        ];
        }
...
}
```

setDomainContacts($domain, array $vars = [], $module_row_id = null)

(optional) Updates the list of contacts associated with a domain.

```
class MyModule extends RegistrarModule {
...
    public function setDomainContacts($domain, array $vars = [], $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Set the domain contacts (this is dependent on the module's API, omitted here)
                $contacts = $api->setContacts($domain);

                return ($contacts->status == 'success');
        }
...
}
```

getDomainInfo($domain, $module_row_id = null)

(optional) Gets a list of basic information for a domain.

```
class MyModule extends RegistrarModule {
...
    public function getDomainInfo($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Get the domain information (this is dependent on the module's API, omitted here)
                $domain = $api->getInformation($domain);

                return (array) $domain;
        }
...
}
```

getDomainIsLocked($domain, $module_row_id = null)

(optional) Returns whether the domain has a registrar lock.

```
class MyModule extends RegistrarModule {
...
    public function getDomainIsLocked($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Get the domain information (this is dependent on the module's API, omitted here)
                $domain = $api->getInformation($domain);

                return ($domain->status == 'locked');
        }
...
}
```

getDomainNameServers($domain, $module_row_id = null)

(optional) Returns a list of name server data associated with the domain.

```
class MyModule extends RegistrarModule {
...
    public function getDomainNameServers($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Get the domain information (this is dependent on the module's API, omitted here)
                $domain = $api->getInformation($domain);

                return $domain->name_servers;
        }
...
}
```

setDomainNameservers($domain, $module_row_id = null, array $vars = [])

(optional) Assigns new name servers to the domain.

```
class MyModule extends RegistrarModule {
...
    public function setDomainNameservers($domain, $module_row_id = null, array $vars = [])
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Set the domain name servers (this is dependent on the module's API, omitted here)
                $nameservers = $api->setNameservers($vars);

                return ($nameservers->status == 'success');
        }
...
}
```

lockDomain($domain, $module_row_id = null)

(optional) Locks the domain.

```
class MyModule extends RegistrarModule {
...
    public function lockDomain($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Set the domain status (this is dependent on the module's API, omitted here)
                $status = $api->setStatus($domain, 'locked');

                return ($nameservers->status == 'success');
        }
...
}
```

unlockDomain($domain, $module_row_id = null)

(optional) Unlocks the domain.

```
class MyModule extends RegistrarModule {
...
    public function unlockDomain($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Set the domain status (this is dependent on the module's API, omitted here)
                $status = $api->setStatus($domain, 'active');

                return ($nameservers->status == 'success');
        }
...
}
```

registerDomain($domain, $module_row_id = null, array $vars = [])

(required) Register a new domain with the registrar.

```
class MyModule extends RegistrarModule {
...
    public function registerDomain($domain, $module_row_id = null, array $vars = [])
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Register the new domain (this is dependent on the module's API, omitted here)
                $domain = $api->newDomain($domain, $vars);

                return ($domain->status == 'success');
        }
...
}
```

transferDomain($domain, $module_row_id = null, array $vars = [])

(optional) Transfers the domain to a new registrar.

```
class MyModule extends RegistrarModule {
...
    public function transferDomain($domain, $module_row_id = null, array $vars = [])
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Transfer the domain (this is dependent on the module's API, omitted here)
                $transfer = $api->transferDomain($domain, $vars['epp_code'], $vars);

                // Send confirmation email
                $this->resendTransferEmail($domain, $module_row_id);

                return ($transfer->status == 'pending');
        }
...
}
```

resendTransferEmail($domain, $module_row_id = null)

(optional) Resends the domain transfer verification email.

```
class MyModule extends RegistrarModule {
...
    public function resendTransferEmail($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Send transfer email (this is dependent on the module's API, omitted here)
                $transfer = $api->sendTransferEmail($domain);

                return ($transfer->status == 'success');
        }
...
}
```

sendEppEmail($domain, $module_row_id = null)

(optional) Sends the domain transfer auth code to the admin email.

```
class MyModule extends RegistrarModule {
...
    public function sendEppEmail($domain, $module_row_id = null)
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Send transfer email (this is dependent on the module's API, omitted here)
                $transfer = $api->sendTransferEmail($domain);

                return ($transfer->status == 'success');
        }
...
}
```

updateEppCode($domain, $epp_code, $module_row_id = null, array $vars = [])

(optional) Updates the EPP code (Authorization Code) of the domain.

```
class MyModule extends RegistrarModule {
...
    public function updateEppCode($domain, $epp_code, $module_row_id = null, array $vars = [])
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Update the domain EPP code (this is dependent on the module's API, omitted here)
                $domain = $api->updateDomain($domain, ['auth_code' => $epp_code]);

                return ($domain->status == 'success');
        }
...
}
```

renewDomain($domain, $module_row_id = null, array $vars = [])

(required) Renews the domain with the registrar.

```
class MyModule extends RegistrarModule {
...
    public function renewDomain($domain, $module_row_id = null, array $vars = [])
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Renew domain (this is dependent on the module's API, omitted here)
                $domain = $api->renewDomain($domain, $vars);

                return ($domain->status == 'success');
        }
...
}
```

restoreDomain($domain, $module_row_id = null, array $vars = [])

(required) Restores the domain through the registrar.

```
class MyModule extends RegistrarModule {
...
    public function restoreDomain($domain, $module_row_id = null, array $vars = [])
    {
        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Restore domain (this is dependent on the module's API, omitted here)
                $domain = $api->restoreDomain($domain, $vars);

                return ($domain->status == 'success');
        }
...
}
```

getServiceDomain($service)

(optional) Registrar modules should use the 'domain' key for their domain field, but if they choose something different, this method can be overridden to get the domain from the appropriate service field.

```
class MyModule extends RegistrarModule {
...
    public function getServiceDomain($service)
    {
        if (isset($service->fields)) {
            foreach ($service->fields as $service_field) {
                if ($service_field->key == 'domain') {
                    return $service_field->value;
                }
            }
        }

        return $this->getServiceName($service);
    }
...
}
```

getExpirationDate($service, $format = 'Y-m-d H:i:s')

(optional) The getExpirationDate() method is called by the Domain Manager plugin to synchronize the expiration date of the domain with the renewal date of the service.

```
class MyModule extends RegistrarModule {
...
    public function getExpirationDate($service, $format = 'Y-m-d H:i:s')
    {
        $domain = $this->getServiceDomain($service);
        $module_row_id = $service->module_row_id ?? null;

        $row = $this->getModuleRow($module_row_id);
        $api = $this->getApi(...);

                // Get the domain information (this is dependent on the module's API, omitted here)
                $domain = $api->getDomain($domain);

                return date(strtotime($domain->expirationDate), $format);
        }
...
}
```

getTlds($module_row_id = null)

(required) The getTlds() method returns a list of the TLDs supported by the registrar module. Without implementing this method the module will not support any TLDs.

```
class MyModule extends RegistrarModule {
...
        public function getTlds($module_row_id = null) {
                return [
                        '.com',
                        '.net',
                        '.org'
                ];
        }
...
}
```

getTldPricing($module_row_id = null)

(optional) The getTldPricing() method returns a list of the TLD prices.

```
class MyModule extends RegistrarModule {
...
    public function getTldPricing($module_row_id = null)
    {
        // Returns an array containing the pricing for each tld
        return [
            '.com' => [
                'USD' => [
                    1 => ['register' => 10, 'transfer' => 10, 'renew' => 10],
                    2 => ['register' => 20, 'transfer' => 20, 'renew' => 20]
                ]
            ]
        ];
    }
...
}
```

Optional Methods

The methods below are optional, but may be required to implement a module of any utility.

install/upgrade/uninstall()

The methods are invoked when the module is installed, upgraded, or uninstalled respectively.

```
class MyModule extends Module {
...
        public function install() {
                // Perform install logic, such as installing cron tasks
                Loader::loadModels($this, ['CronTasks']);

                // Retrieve the cron task if it already exists for another company
                $task = $this->CronTasks->getByKey('task_one', 'my_module', 'module');

                if (!$task) {
                        // Create the automation task
                        $task = [
                                'key' => 'task_one', // a string used to identify this cron task (see
MyPluginPlugin::cron())
                                'task_type' => 'module', // this cron task is for a module, so it must be set
to 'module'
                                'dir' => 'my_module', // the directory of this module
                                'name' => 'My Module Task', // the name of this cron task
                                'description' => 'This cron tasks does stuff', // the description of this task
                                'type' => 'time' // "time" = once per day at a defined time, "interval" = every
few minutes or hours
                        ];
                        $task_id = $this->CronTasks->add($task);
                } else {
                        $task_id = $task->id;
                }

                // Create the cron task run for this company
                if ($task_id) {
                        $task_run = array(
                                'time' => '14:25', // the daily 24-hour time that this task should run
(optional, required if interval is not given)
                                // 'interval' => '15', // the interval, in minutes, that this cron task should
run (optional, required if time is not given)
                                'enabled' => 1, // 1 if this cron task is enabled, 0 otherwise (optional,
default 1)
                        );
                        $this->CronTasks->addTaskRun($task_id, $task_run);
                }
        }

    public function upgrade($current_version) {
                // Perform upgrade logic
    }

        public function uninstall($module_id, $last_instance) {
                // Perform uninstall logic, such as deleting cron tasks
                Loader::loadModels($this, ['CronTasks']);

                // Retrieve the cron task run for this company
                $cron_task_run = $this->CronTasks->getTaskRunByKey('task_one', 'my_module', false, 'module');

                if ($last_instance) {
                        // Delete all trace of this module, such as cron tasks
            // Remove the cron task altogether
            $cron_task = $this->CronTasks->getByKey('task_one', 'my_module', 'module');
            if ($cron_task) {
                $this->CronTasks->deleteTask($cron_task->id, 'module', 'my_module');
            }
                }

        // Remove individual task run
        if ($cron_task_run) {
            $this->CronTasks->deleteTaskRun($cron_task_run->task_run_id);
        }
        }
...
}
```

cron($key)

The cron() method is called whenever a cron task (identified by $key and) registered for the module is ready to be run. This is similar to plugin cron tasks.

> **Create your cron tasks first**
>
> You must create your cron tasks during an install() or upgrade() in order for them to exist in Blesta and be run automatically. See *install/upgrade /uninstall* above for an example.

```
class MyModule extends Module {
...
        public function cron($key) {
                switch ($key) {
                        case "task_one":
                                // Perform any action the module should do based on the cron task
                                break;
                }
        }
...
}
```

getDescription()

The getDescription() method simply returns the description of the module. It was added in Blesta v4.9.0. It's always best to define any language in your module using language files (see Translating Blesta for more information).

```
class MyModule extends Module {
...
        public function getDescription() {
                return Language::_("MyModule.description", true);
        }
...
}
```

getLogo()

The getLogo() method returns the relative path (within the module's directory) to the logo used to represent the module. The default value is **views/default /images/logo.png**. This translates to **/install_dir/components/modules/my_module/views/default/images/logo.png**.

```
class MyModule extends Module {
...
        public function getLogo() {
                return "some/path/to/my/logo.png";
        }
...
}
```

validateService($package, array $vars=null)

The validateService() method performs any input validation against the selected package and vars, and sets any input errors. This is typically called before attempting to provision a service within the addService() or editService() methods. It returns a boolean value indicating whether the given input is valid.

```
class MyModule extends Module {
...
        public function validateService($package, array $vars=null) {
                // Set any input rules to validate against
                $rules = array(
                        'mymodule_field' => array(
                                'empty' => array(
                                        'rule' => "isEmpty",
                                        'negate' => true,
                                        'message' => Language::_("MyModule.!error.mymodule_field.empty", true)
                                );
                        )
                );

                $this->Input->setRules($rules);

                // Determine whether the input validates
                return $this->Input->validates($vars);
        }
...
}
```

addService($package, array $vars=null, $parent_package=null, $parent_service=null, $status="pending")

This method attempts to add a service given the package and input vars, as well as the intended status. If this service is an addon service, the parent package will be given. The parent service will also be given if the parent service has already been provisioned. This method returns an array containing an array of key=>value fields for each service field and its value, as well as whether the value should be encrypted.

```
class MyModule extends Module {
...
        public function addService($package, array $vars=null, $parent_package=null, $parent_service=null,
$status="pending") {
                // Get the module row used for this service
                $row = $this->getModuleRow();

        // Filter the given $vars into an array of key/value pairs that will be passed to the API
        $params = $this->getFieldsFromInput((array)$vars, $package);

                // Attempt to validate the input and return on failure
                $this->validateService($package, $vars);
                if ($this->Input->errors())
                        return;

                // Only provision the service remotely if 'use_module' is true
                if (isset($vars['use_module']) && $vars['use_module'] == "true") {
                        // Log the input being sent remotely, careful to mask any sensitive information
                        $masked_params = $params;
                        $masked_params['mymodule_field'] = "***";

                        // Set the URL to where the remote request is being sent (assuming 'host_name' is a
valid module row meta field)
                        $remote_url = $row->meta->host_name;
                        $this->log($remote_url . "|api_command", serialize($masked_params), "input", true);

                        // Provision the service remotely (this is dependent on the module's API, omitted here)
                        $response = $this->makeRequest($params);

                        // Return on error
                        if ($this->Input->errors())
                                return;
                }

                // Return the service fields
                return array(
                        array(
```

```
                                'key' => "mymodule_field",
                                'value' => (isset($vars['mymodule_field']) ? $vars['mymodule_field'] : null),
                                'encrypted' => 0
                        )
                );
        }

        private function makeRequest($params) {
                // Get the module row used for this service
                $row = $this->getModuleRow();

                // Perform the remote request (this is dependent on the module's API, omitted here)
                $response = $this->apiCall($params);

                // Retrieve the response from the module and evaluate its result as true/false, setting any
input errors
                $success = true;
                if (isset($response->status) && !$response->status) {
                        $this->Input->setErrors(array('api' => array('response' => Language::_("MyModule.!error.
api.response", true))));
                        $success = false;
                }

                // Log the response
                $this->log($row->meta->host_name, $response, "output", $success);

                // Return the result
                if (!$success)
                        return;
                return $response;
        }

        private function apiCall($params) {
                // Make the API call to the module (this is dependent on the module's API, omitted here)
                return (object)array('status' => false);
        }
...
}
```

editService($package, $service, array $vars=array(), $parent_package=null, $parent_service=null)

This method attempts to update an existing service given the package, the service, and any input vars. If this service is an addon service, the parent package will be given. The parent service will also be given if the parent service has already been provisioned. This method returns an array containing an array of key=>value fields for each service field and its value, as well as whether the value should be encrypted.

This method is very similar to addService().

cancelService($package, $service, $parent_package=null, $parent_service=null)

This method attempts to cancel an existing service given the package and the service. If this service is an addon service, the parent package will be given. The parent service will also be given if the parent service has already been provisioned. This method may return null, or the service fields as an array containing an array of key=>value fields for each service field and its value, as well as whether the value should be encrypted.

```
class MyModule extends Module {
...
        public function cancelService($package, $service, $parent_package=null, $parent_service=null) {
                // Get the module row used for this service
                if (($row = $this->getModuleRow())) {
                        // Format the list of service fields as an object
                        $service_fields = $this->serviceFieldsToObject($service->fields);

                        // Set the URL to where the remote request is being sent (assuming 'host_name' is a
valid module row meta field)
                        $remote_url = $row->meta->host_name;
                        $this->log($remote_url . "|api_command", serialize($service_fields), "input", true);

                        // Provision the service remotely (this is dependent on the module's API, omitted here)
                        $response = $this->makeRequest($service_fields);
                }
                return null;
        }

        private function makeRequest($params) {
                // Get the module row used for this service
                $row = $this->getModuleRow();

                // Perform the remote request (this is dependent on the module's API, omitted here)
                $response = $this->apiCall($params);

                // Retrieve the response from the module and evaluate its result as true/false, setting any
input errors
                $success = true;
                if (isset($response->status) && !$response->status) {
                        $this->Input->setErrors(array('api' => array('response' => Language::_("MyModule.!error.
api.response", true))));
                        $success = false;
                }

                // Log the response
                $this->log($row->meta->host_name, $response, "output", $success);

                // Return the result
                if (!$success)
                        return;
                return $response;
        }

        private function apiCall($params) {
                // Make the API call to the module (this is dependent on the module's API, omitted here)
                return (object)array('status' => false);
        }
...
}
```

suspendService($package, $service, $parent_package=null, $parent_service=null)

This method attempts to suspend an existing service given the package and service. If this service is an addon service, the parent package will be given. The parent service will also be given if the parent service has already been provisioned. This method may return null, or the service fields as an array containing an array of key=>value fields for each service field and its value, as well as whether the value should be encrypted.

This method is very similar to cancelService().

unsuspendService($package, $service, $parent_package=null, $parent_service=null)

This method attempts to unsuspend an existing service given the package and service. If this service is an addon service, the parent package will be given. The parent service will also be given if the parent service has already been provisioned. This method may return null, or the service fields as an array containing an array of key=>value fields for each service field and its value, as well as whether the value should be encrypted.

This method is very similar to cancelService().

renewService($package, $service, $parent_package=null, $parent_service=null)

This method attempts to renew an existing service given the package and service. If this service is an addon service, the parent package will be given. The parent service will also be given if the parent service has already been provisioned. This method may return null, or the service fields as an array containing an array of key=>value fields for each service field and its value, as well as whether the value should be encrypted.

This method is very similar to cancelService().

changeServicePackage($package_from, $package_to, $service, $parent_package=null, $parent_service=null)

This method attempts to update the package being used for an existing service given the current package, the new package, and the service. If this service is an addon service, the parent package will be given. The parent service will also be given if the parent service has already been provisioned. This method may return null, or the service fields as an array containing an array of key=>value fields for each service field and its value, as well as whether the value should be encrypted.

This method is very similar to cancelService().

addPackage(array $vars=null)

This method attempts to save any meta data related to a package, and may perform a remote request to add a package. It also performs any input validation against the input vars, and sets any input errors. This method returns meta fields as an array containing an array of key=>value fields for each meta field and its value, as well as whether the value should be encrypted.

```
class MyModule extends Module {
...
        public function addPackage(array $vars=null) {
                // Set any package meta field rules
                $rules = array(
                        'meta[field]' => array(
                                'empty' => array(
                                        'rule' => "isEmpty",
                                        'negate' => true,
                                        'message' => Language::_("MyModule.!error.meta[field].empty", true)
                                )
                        )
                );

                // Determine whether the input validates
                $meta = array();
                if ($this->Input->validates($vars)) {
                        // Set meta fields to save
                        foreach ($vars['meta'] as $key => $value) {
                                $meta[] = array(
                                        'key' => $key,
                                        'value' => $value,
                                        'encrypted' => 0
                                );
                        }
                }

                return $meta;
        }
...
}
```

editPackage($package, array $vars=null)

This method attempts to update any meta data related to a package, and may perform a remote request to add a package. It also performs any input validation against the input vars, and sets any input errors. This method returns meta fields as an array containing an array of key=>value fields for each meta field and its value, as well as whether the value should be encrypted.

This method is very similar to addPackage().

deletePackage($package)

This method attempts to delete a package from a remote server. It may set Input errors on failure.

```
class MyModule extends Module {
...
        public function deletePackage($package) {
                // Attempt to delete a package remotely (this is dependent on the module's API, omitted here)
                $response = $this->apiCall($package);

                if (isset($response->status) && !$response->status)
                        $this->Input->setErrors(array('api' => array('response' => Language::_("MyModule.!error.
api.response", true))));
        }

        private function apiCall($params) {
                // Make the API call to the module (this is dependent on the module's API, omitted here)
                return (object)array('status' => false);
        }
...
}
```

manageModule($module, array &$vars)

The manageModule() method returns HTML content for the manage module page for the given module. Any post data submitted will be passed by reference in $vars.

```
class MyModule extends Module {
...
        public function manageModule($module, array &$vars) {
                // Load the view into this object, so helpers can be automatically added to the view
        $this->view = new View("manage", "default");
        $this->view->base_uri = $this->base_uri;
        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);

        // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html", "Widget"));
        $this->view->set("module", $module);

        return $this->view->fetch();
        }
...
}
```

manageAddRow(array &$vars)

The manageAddRow() method returns HTML content for the add module row page. Any post data submitted will be passed by reference in $vars.

```
class MyModule extends Module {
...
        public function manageAddRow(array &$vars) {
                // Load the view into this object, so helpers can be automatically added to the view
        $this->view = new View("add_row", "default");
        $this->view->base_uri = $this->base_uri;
        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);

        // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html", "Widget"));
        $this->view->set("vars", (object)$vars);

        return $this->view->fetch();
        }
...
}
```

manageEditRow($module_row, array &$vars)

The manageEditRow() method returns HTML content for the edit module row page given the module row to update. Any post data submitted will be passed by reference in $vars.

```
class MyModule extends Module {
...
        public function manageEditRow($module_row, array &$vars) {
                // Load the view into this object, so helpers can be automatically added to the view
        $this->view = new View("edit_row", "default");
        $this->view->base_uri = $this->base_uri;
        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);

                // Set initial module row meta fields for vars
                if (empty($vars))
                        $vars = $module_row->meta;

                // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html", "Widget"));
        $this->view->set("vars", (object)$vars);

        return $this->view->fetch();
        }
...
}
```

addModuleRow(array &$vars)

This method attempts to add a module row given the input vars, and sets any Input errors on failure. This method returns meta fields as an array containing an array of key=>value fields for each meta field and its value, as well as whether the value should be encrypted.

```
class MyModule extends Module {
...
        public function addModuleRow(array &$vars) {
                // Set a white list of fields to add to the module row
                $fields = array("field1", "password");
                $encrypted_fields = array("password");

                // Set any rules for the module row fields
                $rules = array(
                        'field1' => array(
                                'empty' => array(
                                        'rule' => "isEmpty",
                                        'negate' => true,
                                        'message' => Language::_("MyModule.!error.field1.empty", true)
                                )
                        ),
                        'password' => array(
                                'empty' => array(
                                        'rule' => "isEmpty",
                                        'negate' => true,
                                        'message' => Language::_("MyModule.!error.password.empty", true)
                                )
                        )
                );

                $this->Input->setRules($rules);

                // Determine whether the input validates
                if ($this->Input->validates($vars)) {
                        // Add each field
                        $meta = array();
                        foreach ($vars as $key => $value) {
                                if (in_array($key, $fields)) {
                                        $meta[] = array(
                                                'key' => $key,
                                                'value' => $value,
                                                'encrypted' => in_array($key, $encrypted_fields) ? 1 : 0
                                        );
                                }
                        }

                        return $meta;
                }
        }
...
}
```

editModuleRow($module_row, array &$vars)

This method attempts to update a module row given the input vars and the module row, and sets any Input errors on failure. This method returns meta fields as an array containing an array of key=>value fields for each meta field and its value, as well as whether the value should be encrypted.

This method is very similar to addModuleRow().

deleteModuleRow($module_row)

This method attempts to delete a module row from a remote server. It may set Input errors on failure.

```
class MyModule extends Module {
...
        public function deleteModuleRow($module_row) {
                // Attempt to delete a module row remotely (this is dependent on the module's API, omitted here)
                $response = $this->apiCall($module_row);

                if (isset($response->status) && !$response->status)
                        $this->Input->setErrors(array('api' => array('response' => Language::_("MyModule.!error.
api.response", true))));
        }

        private function apiCall($params) {
                // Make the API call to the module (this is dependent on the module's API, omitted here)
                return (object)array('status' => false);
        }
...
}
```

getGroupOrderOptions()

This method returns an array of service delegation order methods. For example, if multiple module rows exist for a module, you may want to provide an option to assign new services to the module row with the least number of services already assigned to it.

```
class MyModule extends Module {
...
        public function getGroupOrderOptions() {
                return array('first' => Language::_("MyModule.order_options.first", true);
        }
...
}
```

selectModuleRow($module_group_id)

This method determines which module row should be used to provision a service based on the order method set for the given group. The module row ID of the chosen module row is returned.

```
class MyModule extends Module {
...
        public function selectModuleRow($module_group_id) {
                // Load the ModuleManager
                if (!isset($this->ModuleManager))
                        Loader::loadModels($this, array("ModuleManager"));

                // Select the group associated with the given module group ID
                if (($group = $this->ModuleManager->getGroup($module_group_id))) {
                        // Choose the module row to use
                        switch ($group->add_order) {
                                default:
                                case "first":
                                        // Return the first row encountered
                                        foreach ($group->rows as $row)
                                                return $row->id;
                                        break;
                        }
                }
                return 0;
        }
...
}
```

getPackageFields($vars=null)

This method returns a ModuleFields object containing all fields used when adding or editing a package, including any javascript that can be executed when the page is rendered with those fields. Any post data submitted will be passed in $vars.

```
class MyModule extends Module {
...
        public function getPackageFields($vars=null) {
                // Load any helpers required to build the fields
                Loader::loadHelpers($this, array("Html"));

                // Set any module fields
                $fields = new ModuleFields();
                $fields->setHtml("
                        <script type=\"text/javascript\">
                                $(document).ready(function() {
                                        // Re-fetch module options
                                        $('#mymodule_group').change(function() {
                                                fetchModuleOptions();
                                        });
                                });
                        </script>
                ");

                // Fetch all packages available for the given server or server group
                $module_row = null;
                if (isset($vars->module_group) && $vars->module_group == "") {
                        // Set a module row if one is given
                        if (isset($vars->module_row) && $vars->module_row > 0)
                                $module_row = $this->getModuleRow($vars->module_row);
                        else {
                                // Set the first module row of any that exist
                                $rows = $this->getModuleRows();
                                if (isset($rows[0]))
                                        $module_row = $rows[0];
                                unset($rows);
                        }
                }
                else {
                        // Set the first module row from the list of servers in the selected group
                        $rows = $this->getModuleRows($vars->module_group);
                        if (isset($rows[0]))
                                $module_row = $rows[0];
                        unset($rows);
                }

                // Build any HTML fields
                $select_options = array('one' => "One", 'two' => "Two");
                $field = $fields->label(Language::_("MyModule.package_fields.field1", true), "mymodule_field");
                $field->attach($fields->fieldSelect("meta[field]", $select_options,
                        $this->Html->ifSet($vars->meta['field']), array('id' => "mymodule_field")));
                $fields->setField($field);

                return $fields;
        }
...
}
```

getEmailTags()

This method returns an array of key/value pairs with "module", "package", and "service" keys that refer to module, package, and service fields used in this module that may be used as tags in emails.

```
class MyModule extends Module {
...
        public function getEmailTags() {
        return array(
            'module' => array("field1", "password"),
            'package' => array("field"),
            'service' => array("mymodule_field")
        );
    }
...
}
```

getAdminAddFields($package, $vars=null)

This method returns a ModuleFields object containing fields displayed when an admin goes to create a service.

```
class MyModule extends Module {
...
        public function getAdminAddFields($package, $vars=null) {
        Loader::loadHelpers($this, array("Html"));

        $fields = new ModuleFields();

        // Create field label
        $mymodule_field = $fields->label(Language::_("MyModule.service_field.mymodule_field", true),
"mymodule_field");
        // Create field and attach to label
        $mymodule_field->attach($fields->fieldText("mymodule_field", $this->Html->ifSet($vars->mymodule_field),
array('id'=>"mymodule_field")));
                // Add a tooltip next to this field
        $tooltip = $fields->tooltip(Language::_("MyModule.service_field.tooltip.mymodule_field", true));
        $mymodule_field->attach($tooltip);
        // Set the field
        $fields->setField($mymodule_field);

                return $fields;
        }
...
}
```

getClientAddFields($package, $vars=null)

This method returns a ModuleFileds object containing fields displayed when a client goes to create a service.

This method is very similar to getAdminAddFields().

getAdminEditFields($package, $vars=null)

This method returns a ModuleFields object containing fields displayed when an admin goes to update a service.

This method is very similar to getAdminAddFields().

getAdminServiceInfo($service, $package)

This method returns the view that is displayed when an admin clicks an expandable service row to view details about a service.

```
class MyModule extends Module {
...
        public function getAdminServiceInfo($service, $package) {
        $row = $this->getModuleRow();

        // Load the view (admin_service_info.pdt) into this object, so helpers can be automatically added to
the view
        $this->view = new View("admin_service_info", "default");
        $this->view->base_uri = $this->base_uri;
        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);

        // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html"));

        $this->view->set("module_row", $row);
        $this->view->set("package", $package);
        $this->view->set("service", $service);
        $this->view->set("service_fields", $this->serviceFieldsToObject($service->fields));

        return $this->view->fetch();
    }
...
}
```

getClientServiceInfo($service, $package)

This method returns the view that is displayed when a client clicks an expandable service row to view details about their service.

This method is very similar to getAdminServiceInfo().

getAdminTabs($package)

(deprecated) This method returns a list of key/value pairs representing tab names to display when an admin goes to manage a service.

```
class MyModule extends Module {
...
        public function getAdminTabs($package) {
                // The keys (i.e. "tabOne", "tabTwo") representing the method name of the tab to call when an
admin clicks on it in the interface
        return array(
            'tabOne' => Language::_("MyModule.tab_one", true),
                        'tabTwo' => Language::_("MyModule.tab_two", true)
        );
    }

        public function tabOne($package, $service, array $get=null, array $post=null, array $files=null) {
        $this->view = new View("tab_one", "default");
        // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html"));

                // Set any specific data for this tab
                $tab_data = array();
        $this->view->set("tab_data", $tab_data);

        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);
        return $this->view->fetch();
    }

        public function tabTwo($package, $service, array $get=null, array $post=null, array $files=null) {
        $this->view = new View("tab_two", "default");
        // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html"));

                // Set any specific data for this tab
                $tab_data = array();
        $this->view->set("tab_data", $tab_data);

        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);
        return $this->view->fetch();
    }
...
}
```

getClientTabs($package)

(deprecated) This method returns a list of key/value pairs representing tab names to display when a client goes to manage their service.

This method is very similar to getAdminTabs().

getAdminServiceTabs($service)

This method returns a list of key/value pairs representing tab names to display when an admin goes to manage a service.

```
class MyModule extends Module {
...
        public function getAdminServiceTabs($package) {
                // The keys (i.e. "tabOne", "tabTwo") representing the method name of the tab to call when an
admin clicks on it in the interface
        return array(
            'tabOne' => Language::_("MyModule.tab_one", true),
                        'tabTwo' => Language::_("MyModule.tab_two", true)
        );
    }

        public function tabOne($package, $service, array $get=null, array $post=null, array $files=null) {
        $this->view = new View("tab_one", "default");
        // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html"));

                // Set any specific data for this tab
                $tab_data = array();
        $this->view->set("tab_data", $tab_data);

        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);
        return $this->view->fetch();
    }

        public function tabTwo($package, $service, array $get=null, array $post=null, array $files=null) {
        $this->view = new View("tab_two", "default");
        // Load the helpers required for this view
        Loader::loadHelpers($this, array("Form", "Html"));

                // Set any specific data for this tab
                $tab_data = array();
        $this->view->set("tab_data", $tab_data);

        $this->view->setDefaultView("components" . DS . "modules" . DS . "my_module" . DS);
        return $this->view->fetch();
    }
...
}
```

getClientServiceTabs($service)

This method returns a list of key/value pairs representing tab names to display when a client goes to manage their service.

This method is very similar to getAdminServiceTabs().