



## Ruby on Rails Plugin

November 2008

## Overview

This document is a complementary reference to the integration guide [3scale-integration](#)<sup>1</sup>. In that document you will find information on the whole integration process and the generic connection using the 3scale API. The current document just provides specific details for connecting to the 3scale backend using the Ruby on Rails plugin. This plugin encapsulates the call to the 3scale API.

## Ruby on Rails Plugin

To ease the integration task of your service with the 3scale backend, 3scale offers a library (plugin) that encapsulates the communication actions with 3scale, taking complete control of the data connection, data encryption, and data caching to ensure reliability and performance.

The RoR plugin is distributed as a gem. You can install it by issuing the following commands:

```
gem source --add http://3scale.net/gems/  
gem install 3scale_interface
```

You can update to its newest version by doing:

```
gem update 3scale_interface
```

You will find extended information on: [http://github.com/3scale/3scale\\_ws\\_api\\_for\\_ruby/tree/master](http://github.com/3scale/3scale_ws_api_for_ruby/tree/master)

Within your application code you need to create an interface object to access the plugin functions. You need to provide the 3scale url, and the private provider key for your service (see key descriptions in step 4):

```
threescaleInterface = ThreeScale::Interface.new("http://3scale.net", "your_private_service_key")
```

Please review section 4 and 5 on document **3scale-connection** to learn more on where and when to use the functions provided in the API. The syntax of the functions for this plugin is described as follows:

---

<sup>1</sup> Available at <http://www.3scale.net/home/downloads>

## Start function

### Parameters:

- `:user_key` (to identify the user that is requesting the service)
- `:usage` (optional parameter to indicate Predicted resource consumption of this transaction. This should be associative array (a hash) of the form: {metric => value, ...}, where *metric* is name of the metric that measures particular resource (for example "*storage*" or "*hits*") and *value* is any amount of corresponding resource that is expected to be spend. This parameter may contain only approximate values, or it can be left out entirely. In that case, the actual usage should be reported in **Confirm** operation, described in the next section.

### Result:

If 3scale grants access to the user indicated, this function will return a hash containing the following data:

- `:id` (Identifier of the transaction. This is needed for confirmation/cancellation of the transaction later )
- `:contract_name` (Name of contract the user has signed for. The provider can use this information for example to send different results according to contract types, if that is desired)
- `:provider_public_key` (Provider service public key. It can be send back to the user to let him verify the authenticity of the provider, although this is not mandatory)

If the access is not granted by whatever reason, this function will issue an exception indicating the nature of the problem:

- `ThreeScale::MetricInvalid` (if the "usage" parameter contains some undefined metric. The error identifier in this case is: `provider.invalid_metric`)
- `ThreeScale::UserKeyInvalid` (if the user key is invalid)
- `ThreeScale::ProviderKeyInvalid` (if the provider private key is invalid)
- `ThreeScale::ContractNotActive` (when the contract referenced is pending, sespended or canceled)
- `ThreeScale::LimitsExceeded` (if the usage limits of the contract are exceeded)
- `ThreeScale::SystemError` (if there has been some unexpected error)

### Example Request:

```
transaction = threescaleInterface.start('fds8798dfgjnjhj34hj5l34jdsjdfgdf', 'hits' => 1, 'storage' => 42000)
```

### Confirm Function

#### Parameters:

- :transaction\_id (identifier of the transactions, obtained by previous "Start" operation)
- :usage (final resource usage. This parameter is only needed when the predicted resource usage was missing or different from the initial prediction)

#### Result:

If there was no problem with the reported usage operation, no exception will be thrown. Otherwise, this call might issue two different types of exceptions:

- ThreeScale::MetricInvalid (if the "usage" parameter contains some undefined metric. The error identifier in this case is: provider.invalid\_metric)
- ThreeScale::ProviderKeyInvalid (if the provider private key is invalid)
- ThreeScale::TransactionNotFound (if the transaction id is not correct)
- ThreeScale::SystemError (if there has been some unexpected error)

#### Example requests:

```
threescaleInterface.confirm(transaction[:id])
```

```
threescaleInterface.confirm(transaction[:id], 'hits' => 1, 'storage' => 40500)
```

## **Cancel Function**

### **Parameters:**

- :transaction\_id (identifier of the transactions, obtained by previous "Start" operation)

### **Result:**

If there was no problem with the reported usage operation, no exception will be thrown. Otherwise, this call will might issue two different types of exceptions:

- ThreeScale::ProviderKeyInvalid (if the provider private key is invalid)
- ThreeScale::TransactionNotFound (if the transaction id is not correct)
- ThreeScale::SystemError (if there has been some unexpected error)

### **Example requests:**

```
threescaleInterface.cancel(transaction_id)
```

## **Support**

Email support is available from [beta@3scale.net](mailto:beta@3scale.net). A developers forum is also available from <https://www.3scale.net/forums> and phone support will be soon available. 3scale is also constantly trying to improve it's service, so if you have any feedback whatsoever, please don't hesitate to let us know at the same address.