iVend Programmer's Guide

What is an iVend Module?

An iVend Module is a piece of pike code that is loaded into the iVend engine that adds extra capabilities above what would be normally provided by iVend. Some of the features that a module can provide:

- IVML Containers and Tags.
- Path handlers such as /checkout/ and /admin/.
- Administrative handlers that add extra features to the admin interface.
- Register events.

As an example, iVend's checkout procedure and tags are handled by an iVend Module.

iVend Modules are located in the src/modules directory of the source distribution. In general, new features added by third parties should be fabricated using iVend Add−Ins.

What is an iVend Add−In?

An iVend Add−In is an iVend Module that can be loaded dynamically by a particular store. By using a store's administration interface, Add−Ins may be loaded and unloaded on a per store bases. Add−Ins are usually located in storedir/addins. iVend's Upsell handler is an example of an iVend Add−In.

How do I create a new iVend Add−In?

Creating a new Add−In is a relatively simple task. Listed below is the simplest example, `sample.pike`, an Add−In that loads and does nothing:

```
#include <ivend.h>
inherit "roxenlib";

constant module_name = "Sample Add-In";
constant module_type = "addin";
```

The first two lines includes definitions that will come in handy when writing modules. The second two lines define the file as an iVend module. `module_name` is the name of the module that will appear in selection lists and preference boxes. `module_type` identifies the type of module. For most applications, select a `module_type` of `addin`.

To enable this new Add−In, place the newly created file in the addins directory of your store. Go to your store's admin interface and select Add−Ins Manager. You should see the listing "Sample Add−In" in the list. That's all there is to it!

Getting the Add−In to do something

Congratulations! You have an iVend Add−In that does absolutely nothing after it's loaded. Now, suppose we want to have an Add−In that actually does something. iVend has a set of standard interfaces that make it easy to add functionality to the iVend engine.

Preparing the Add−In for use

`void start(mapping config)`

This function is called when the module is loaded. The store's configuration is passed as an argument.

`void stop(mapping config)`

This function is called when the module is unloaded or when the store is stopped.

Registering Functionality

The following functions are used to register functionality with the iVend engine. All return a mapping in the following manner:

```
mapping register_paths(){
  return ([ "demo":demo_function ]);
}
```

The previous function will register a handler for the path "`demo`" in a store that the Add−In is loaded into. When a user requests the path "`demo`" from that store, the request will be generated by the function `demo_function`.

Each type of handler takes different arguments, and are described later in this document. You may register multiple types of handlers for each type by including more than one element in the returned mapping.

`mapping register_admin()`

Returns a mapping of admin key and admin handler pairs. For more information, see the section "*Admin Handlers*."

`mapping register_paths()`

Returns a mapping of path key and path handler function pairs. For more information,

see the section "*Path Handlers.*"

`mapping query_event_callers()`

Returns a mapping of event name key and event handler function pairs. See the section "*Event Handlers*."

`mapping query_tag_callers()`

Returns a mapping of tag name and tag function pairs. This function works the same as the similarly named function in Roxen modules.

`mapping query_container_callers()`

Returns a mapping of container name and container function pairs. This function works the same as the similarly named function in Roxen modules.

iVend Convenience Definitions

In order to simplify Add−In writing, the file `ivend.h` contains several definitions that correspond to important iVend structures. By including this file, you gain the following access points:

Definition	Points to
`STORE`	*Store config name*
`CONFIG_ROOT`	*Base of store's config tree*
`DB`	*Store's database connection*
`KEYS`	*Database key cache*

Other access points are defined in this file, but may not function properly within Add−Ins.

Path Handlers

iVend can be extended to handle requests for paths within a store by using *Path Handlers*. For example, if your store is located at http://www.mystore.com/, you can configure iVend to funnel requests for any path below, such as /orderstatus/. iVend's checkout procedure is an example of a path handler.

Functions that act as path handlers take the path and request object as arguments and return either a `string` or an `http_*` function mapping:

`string|mapping demo_handler(string path, object id)`

`path` is the full path requested within the store, and `id` is the Roxen request object.

Event Handlers

Add−Ins can register code to be executed when certain "Events" occur. For example, one may want an email invoice sent when an order has been shipped. This may be

accomplished by using *Event Handlers*.

Functions that act as event handlers take the event name, the Roxen request object, and an mapping containing optional argument information as arguments and return void. The contents of the argument mapping depends upon the event being triggered:

```
void event_handler(string event, object id, mapping args)
```

`event` is the name of the event triggered and `args` is a mapping of optional arguments passed to the event handler.

For more information about the events iVend triggers and how they are triggered, please see the section following titled "*iVend Events.*"

Tags and Containers

Tags and Containers are added to the iVend engine in much the same way as in standard Roxen modules. iVend uses the same registration functions and sends tag and container functions the same arguments as Roxen. For more information, please see the Roxen Programmer's Guide.

Admin Handlers

Admin handlers are used to add functionality to the iVend administration interface.

```
string|mapping admin_handler(string mode, object id)
```

`mode` is the name of the mode and `id` is the Roxen request object.

An admin handler may return either string data which is returned to the client, or a mapping from any of the Roxen `http_*` functions. By default, strings returned by admin handlers will be formatted with the default iVend Administration headers. To disable this feature, set the following line before returning data:

```
ADMIN_FLAGS=NOBORDER;
```

The administration manager will insert any valid actions below the standard header. These actions will appear as buttons within an outlined box. To disable this behavior, set the following line before returning data:

```
ADMIN_FLAGS=NOACTIONS;
```

Note that `NOBORDER` also implies `NOACTIONS`, so you should only set one of these flags.

The mode is where your handler will appear in the iVend administration interface. Modes are identified by dotted strings such as `menu.main.Reload_Store`. iVend uses the following method for determining this:

1. If first two elements of the mode are set to "menu" and "main", then the handler's link

will be created within an outlined box with the title set to the third element. The fourth element specifies the link name. For example, a `mode` of `menu.main.Heebie.Jeebie` will cause an outlined box with a title of "Heebie" will be displayed on the main menu, and a link to the handler will be called "Jeebie".

2. If the first two elements of the mode are equal to the prestate of the page requested, then an "action" button will be displayed on that page, which when pressed, will cause the admin handler to appear within a popup browser box. For example, a mode of `getmodify.product.Upsell` will cause an action button called "Upsell" to appear on the modify products page.

Setting Preferences

```
array:array query_preferences(object id)
```

where `id` is the Roxen request object.

`query_preferences` returns an array of preferences. Each element of the array is an array containing the following elements:

Element	Description of element
0	*The variable name (no spaces allowed).*
1	*"English" variable name (a short description).*
2	*A longer more helpful description of the variable.*
3	*Variable type (see table below).*
4	*An optional default value.*
5	*A string or array containing valid values for the variable (optional).*

Variable Type	Description
VARIABLE_INTEGER	*Used for variables that should only be integers.*
VARIABLE_FLOAT	*Used for variables that should only be floats.*
VARIABLE_STRING	*This may be used for variables that may or may not be strings, but also floats and integers. No sanity checking is performed on this variable type.*
VARIABLE_MULTIPLE	*Allows multiple options to be selected from a list. Requires a valid list of options to be specified in an array.*

iVend Events

iVend can trigger code to run when certain things happen in it's operation. The points that iVend will trigger code to run are called events. For instance, an iVend event occurs when an order has been confirmed by the user, or when an error occurs. Modules and Add−Ins can register code to run when events occur using the event handler registration function mentioned above.

Modules can also trigger events themselves, which will cause code registered for that event to run, regardless of the module in which the event was triggered. To trigger an event, the following function is used:

```
void T_O->trigger_event(string event, object id, mapping
args)
```

`event` is the name of the event, `id` is the Roxen request object, and `args` is a mapping of additional arguments to pass to the event handlers. You may specify an existing event name, or create a unique one. All handlers registered for that event will be called. You should assume that all event handlers are called in a random order.

The following table lists some of the events that the iVend engine triggers and short description of the circumstances in which the event is triggered. The third column lists any arguments that are passed to handlers by the event manager.

Event Name	Circumstances	Arguments Passed
newsessionid	*A new sessionid has been created.*	*sessionid: id of new session*
additem	*An item has been added to a user's cart*	*item: product id being added* *quantity: quantity being added*
deleteitem	An item has been removed from a user's cart	*item: product id being deleted* *series: instance of item being deleted*
updateitem	*User has altered the quantity of an item in their cart.*	*item: product id being updated* *series: instance of item being updated*
confirmorder	*Order has been successfully confirmed by user.*	*orderid: confirmed order id*
ship	*Order has been completely shipped.*	*orderid: id of the shipped order*