|  |  |
| --- | --- |
|  | Aricie.PortalKeeper  Documentation  Latest update: April, 2012 Module version: 3.0.0 |

**Table of contents**

[1 About the module 3](#_Toc322357862)

[2 Step-by-step installation 4](#_Toc322357863)

[2.1 Prerequisites 4](#_Toc322357864)

[2.2 Install the module 4](#_Toc322357865)

[2.2.1 Install latest Aricie libraries (required) 4](#_Toc322357866)

[2.2.2 Install the PortalKeeper PA 4](#_Toc322357867)

[3 QuickStart 5](#_Toc322357868)

[3.1 Setting up the module 5](#_Toc322357869)

[3.2 Configuring PKP 7](#_Toc322357870)

[3.2.1 Setting up firewall rules 7](#_Toc322357871)

[*3.2.2* Setting up a bot 12](#_Toc322357872)

[4 Concepts 18](#_Toc322357873)

[4.1 Global concepts 18](#_Toc322357874)

[4.1.1 Lists, imports and exports 18](#_Toc322357875)

[4.1.2 Rules 19](#_Toc322357876)

[4.1.3 Providers 25](#_Toc322357877)

[4.2 Firewall concepts 27](#_Toc322357878)

[4.2.1 Custom error pages 27](#_Toc322357879)

[4.2.2 Parameters 28](#_Toc322357880)

[4.2.3 Technical Settings 31](#_Toc322357881)

[4.3 Bots concepts 34](#_Toc322357882)

[4.3.1 Technical Settings 34](#_Toc322357883)

[4.3.2 Bots 35](#_Toc322357884)

[4.3.3 Master and User Bots 38](#_Toc322357885)

[5 How to 42](#_Toc322357886)

[5.1 How to use the safe word to regain access to my website? 42](#_Toc322357887)

[6 Support 44](#_Toc322357888)

# About the module

**Aricie – Portal Keeper (PKP)** is a versatile module that adds a workflow engine to your DotNetNuke installation; an instance of this engine can plug your user requests to act a feature-rich firewall and allowing you to create rules that will upgrade your DotNetNuke experience. The engine is also used as the basis for bots that are runfrom the DotNetNuke Scheduler; the robots can be configured to act in the background of your website without human supervision and following your rules. The bots can even be configured to be available to users; you can define aspects of the bots a user can manage so he can benefit from the system without managing the full bot.

Out of the box, PKP provides firewall and behavior rules that you can activate immediately to add intelligence to your portal, such as DDOS protection, auto login, multiple connections detection, mobile browsers management, etc. It also comes pre-configured with bots that can ping urls, query search engines and act as autonomous Bitcoin agents. Of course these rules and bots are provided mainly as an example, and soon you’ll be creating your own custom scenarios on PKP.

All this is configurable from a single module for a whole DotNetNuke instance. The module lets you edit the configuration for PKP directly on your web site so you don’t need to handle configuration files.

Whether you need to secure your portal, automate processes or add custom features to your DotNetNuke instance, Aricie – PortalKeeper is the module you need.

**Required DNN version**: 4.8.1 or above.

This documentation was made on DNN 6.1.

# Step-by-step installation

## Prerequisites

To install the module on your DotNetNuke site, you need the followings:

1. DotNetNuke 4.8.1 or greater
2. SQL Server 2000 or greater
3. File permissions have been set correctly on DotNetNuke, e.g. ASPNET Account on Windows Server 2000, Network Service on Windows Server 2003
4. You have access to an account with “Super User privileges” (typically the host account)
5. You have access to the Download section on [www.aricie.com](http://www.aricie.com)

## Install the module

The Private Assembly is a zip file that contains all the necessary files for the module. This section explains how to install it on your DotNetNuke portal.

### Install latest Aricie libraries (required)

All Aricie modules require some specific libraries to function properly. Please install **Aricie.Shared** on your portal following the same steps as 2.2.1.

NOTE: Aricie modules won't work without these required libraries. Install them like a module on your portal. Get the latest release from our website [www.aricie.com](http://www.aricie.com).

### Install the PortalKeeper PA

1. Download **Aricie.PortalKeeper.XX.XX.XX\_Install.zip** in the Download section of [www.aricie.com](http://www.aricie.com)
2. Connect to your DotNetNuke site as Host
3. Go to **Host > Module Definitions**
4. Click **Install** **module**
5. Click on the Browse button and select the **Aricie.PortalKeeper.XX.XX.XX\_Install.zip** file
6. Click **Install New module**
7. Click **Next**
8. Once the module is successfully installed, click **Return to the site**
   * + That’s it! Now you are ready to add your module to a page.

IMPORTANT: If you receive any other red errors during installation, please copy and paste them in an email to: support@aricie.fr

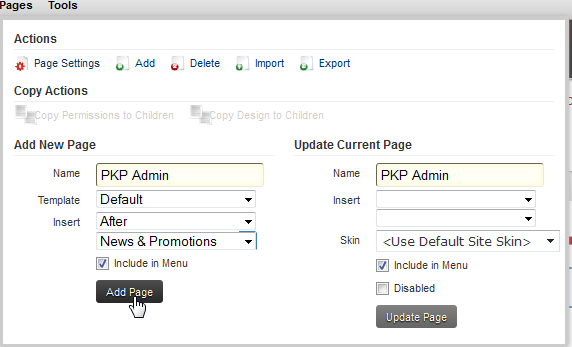
# QuickStart

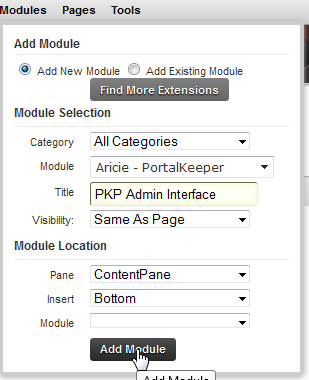
To install PKP, please follow the standard DotNetNuke installation procedure. Below is a step by step setting up procedure; it uses DotNetNuke 6.1 but the process should remain the same for all supported versions.

## Setting up the module

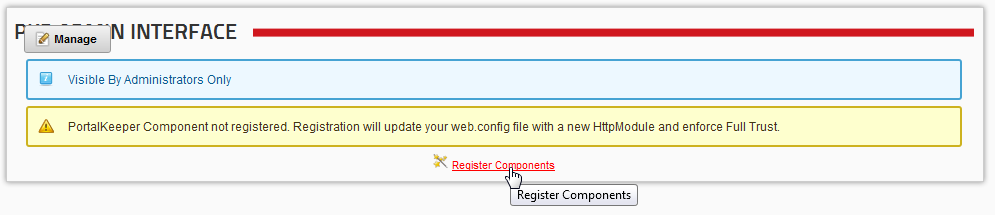
Go to the Extensions menu under Host, and under the Manage menu, click on Install Extension Wizard. Follow the wizard steps to install PKP on your website.

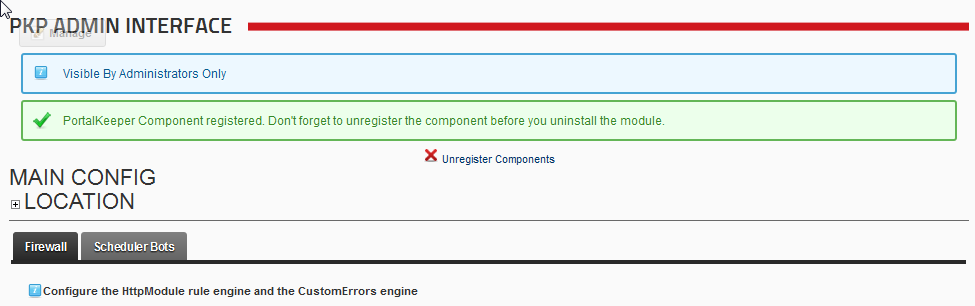
Now let’s add a webpage to install the module’s main interface. I’ll name mine PKP Admin and add the PKP Module on it.





Now you have installed the module on the page; click the Register Components link to create the entries in the web.config file needed to interface PKP to your installation.



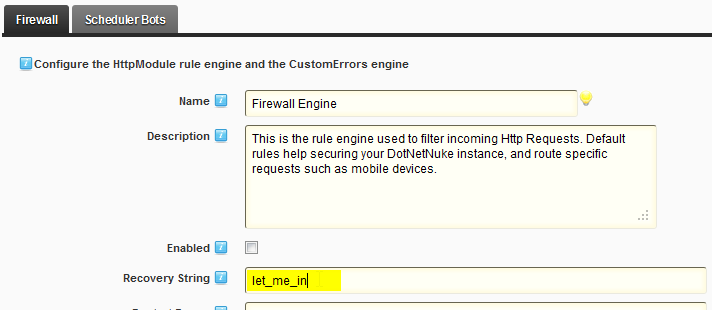
Now the interface is correctly setup on your DotNetNuke instance. 

We are going to quickly setup some of PKP’s features that come with the module by default in the next step.

IMPORTANT: before going any further, you must understand that PKP is a powerful module. As such you could very well find yourself locked out of your website in case of a bad configuration, just as you could lock yourself out of your house by forgetting your keys inside. Caution is advised when changing some parameters.

However in order to help mitigate such problems, PKP allows you to use a “safe word” in order to regain access to your website and correct any parameters that you may have configured incorrectly. We strongly recommend configuring the safeword to a value you’ll be able to remember if things go wrong.

Under the Firewall tab, type in your recovery string and save the settings.

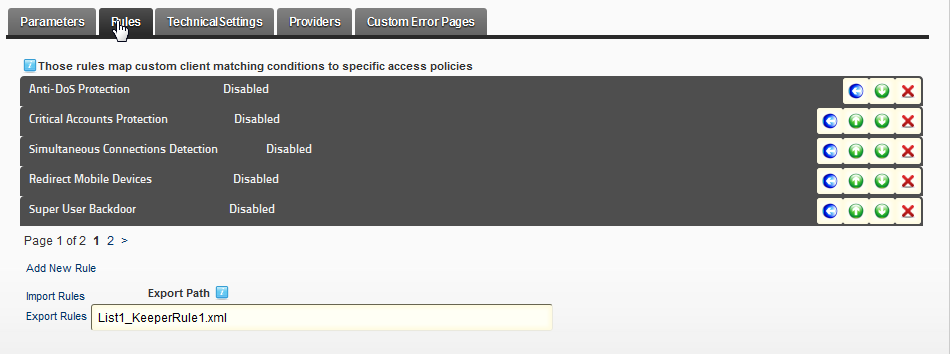


Voila! Now that we have the backdoor key we can resume our step-by-step… You can also read how to unlock yourself in the Howtos.

## Configuring PKP

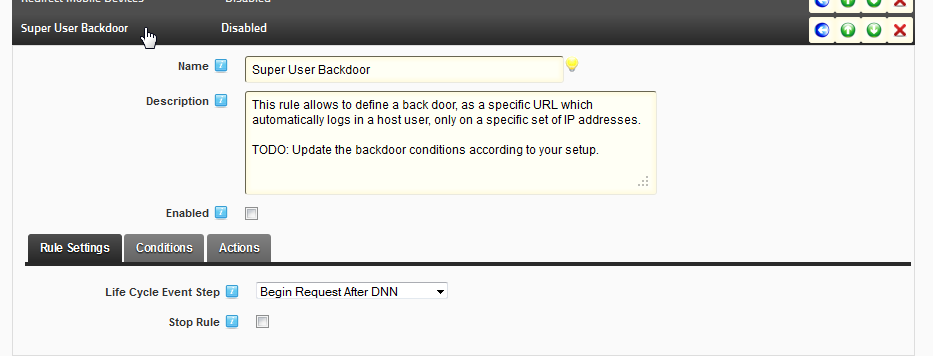
### Setting up firewall rules

We will start by configuring some firewall rules. The rules are the workflows that will be evaluated when a request is sent to your website. They are located under the Rules tab in the interface for the Firewall.

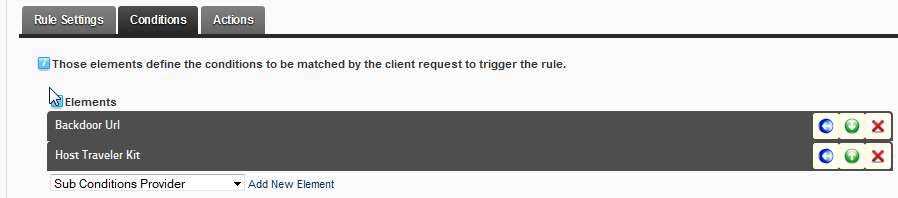


A rule is composed of conditions and actions. When a request is sent to your website, PKP evaluates the conditions of each rule you have set up and if they are triggered it executes the actions of the rule.

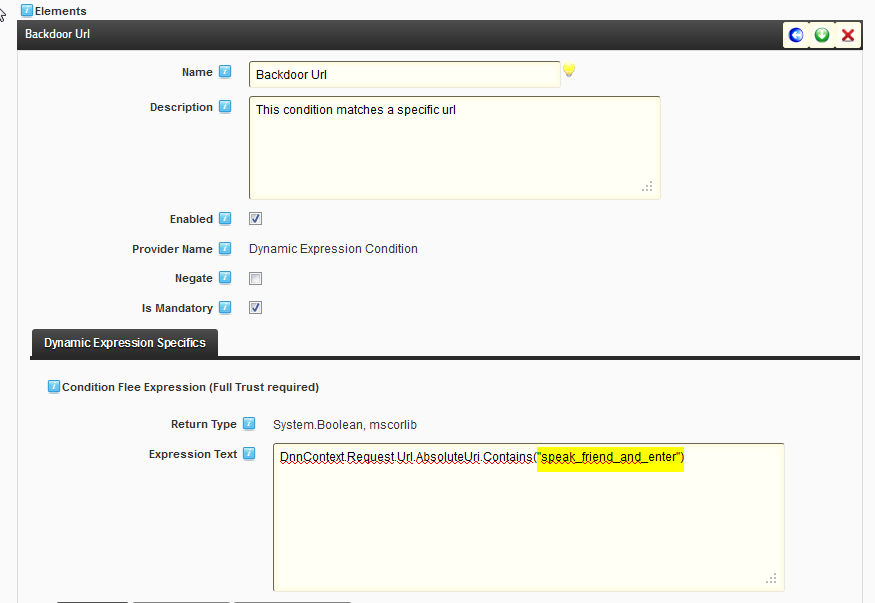
As you can see, PKP comes with some default rules. These cover mainly basic uses of PKP and you will create custom ones later. For now we are going to enable one rule that will make your life easier. It is a rule that will allow us to be automatically connected as Host when coming from a specific ip address with a specific keyword. Let’s click on the Super User Backdoor rule.



If we click on the Conditions tab, we can see the two conditions that have to be satisfied to execute the actions of the rule.

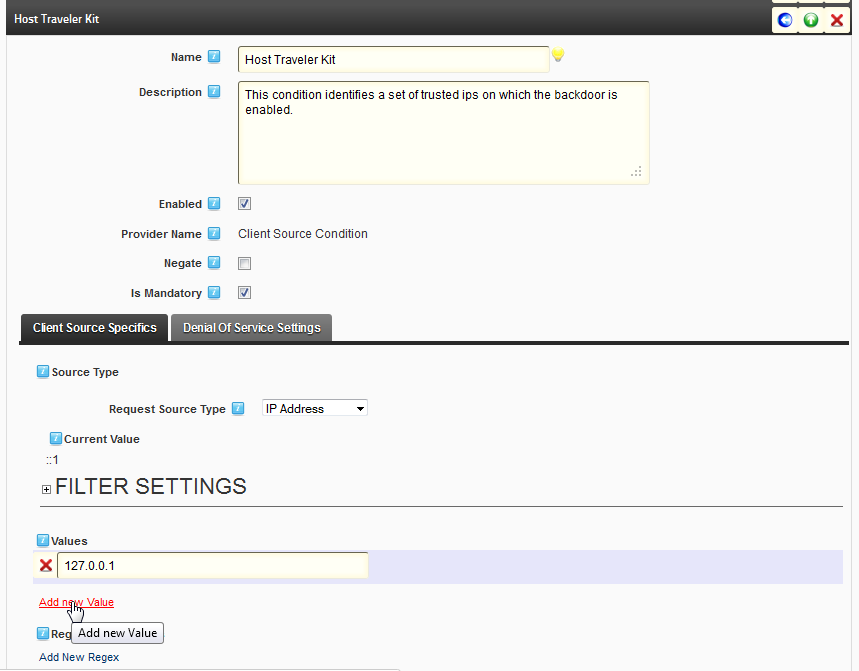


Let’s click on the Backdoor Url condition. As you can see, the condition is expressed as code. We will describe the different conditions you can use later in this document. For now, let’s just change what the url must contain to satisfy the condition. By default, the condition will be triggered if the url contains “opensesame”, but let’s edit this to whatever value suits your needs.



I chose to use the key “speak\_friend\_and\_enter” as a necessary part of the url for the condition to be satisfied. Now let’s click on the second condition Host traveler kit.

PROTIP: You’ll notice that the rule is checked as mandatory. It means that this condition must be satisfied for the rule to proceed to its actions. If any mandatory condition in a rule is not satisfied, the rule will not run any action.

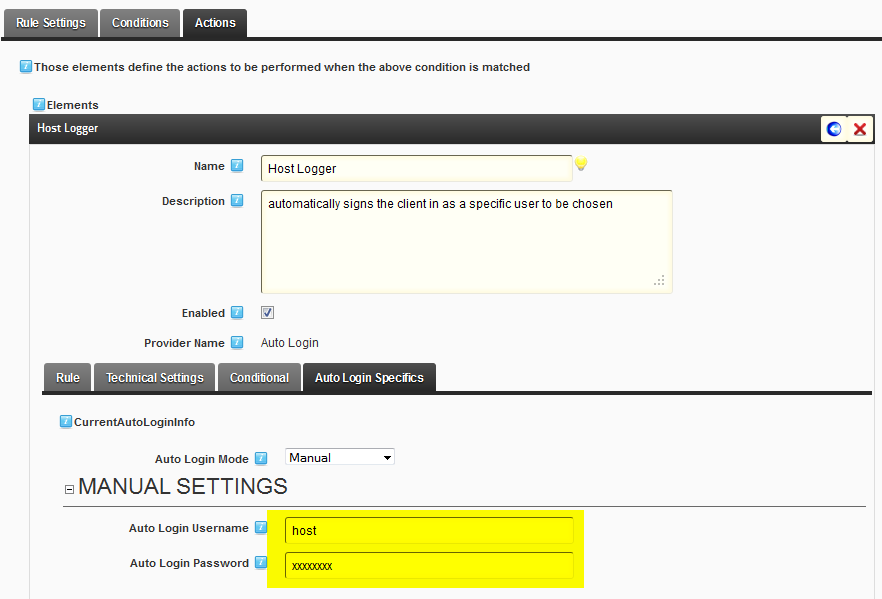


In this condition you can specify your own IP address, or use a regular expression to indicate ranges of addresses that will match the condition. For now let’s say your IP address is 222.173.179.63. I’ll add it to the list directly.



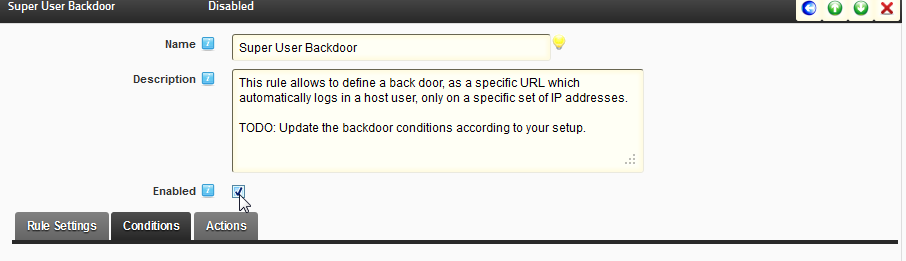
PROTIP: your current IP is displayed in the upper part of the condition, should you need to use it as a reference. As you can see on the screenshot I am working locally on my computer, hence the ::1 that is displayed.

We have configured the two conditions of our rule, now we are going to look at the rule actions. In fact there is only one, called Host Logger. Click on it, and select the Auto Login Specifics tab. Then fill out the credentials of the account you want to connect automatically as.

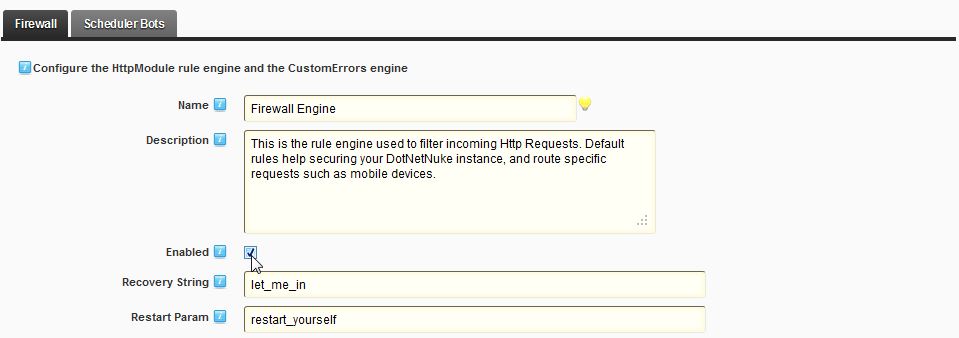


PROTIP: there are many configuration options for the actions. We will detail them later. If you want to know more, you can go to the Global concepts > Rules > Actions section.

We are done. Let’s enable the rule by clicking on the Enabled check box



We do the same for the whole PKP engine at the top of the administration control and save the settings.



Now the firewall will inspect each request to see if the conditions of the Super User Backdoor are satisfied. If the user comes from the defined IPs and if the requested url contains the “speak\_friend\_and\_enter” string, then it authenticates the user directly.

To try it out, open a webbrowser on the machine whose IP you have authorized to connect, navigate to your DotNetNuke website and make sure you’re not logged in; then type the following in your address bar:

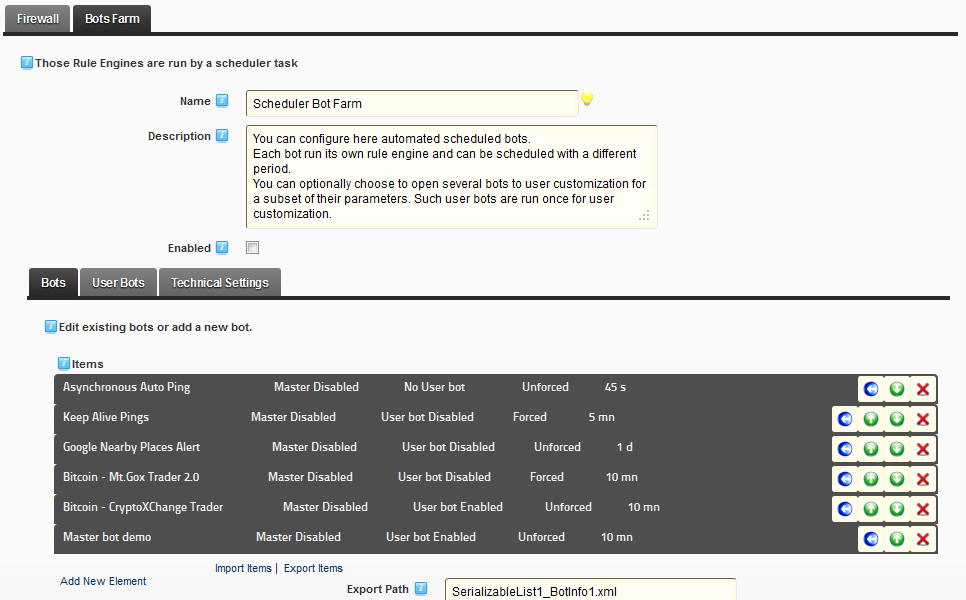
http://www.MyDotNetNuke.com/Default.aspx?speak\_friend\_and\_enter

You should now be authenticated on your DotNetNuke instance as the user whose account you defined in the action.

IMPORTANT: this rule can be dangerous if it is not configured correctly. Make sure that your key is not easily guessable, and that the IPs you have authorized are correctly filtered by the parameters you chose.

### Setting up a bot

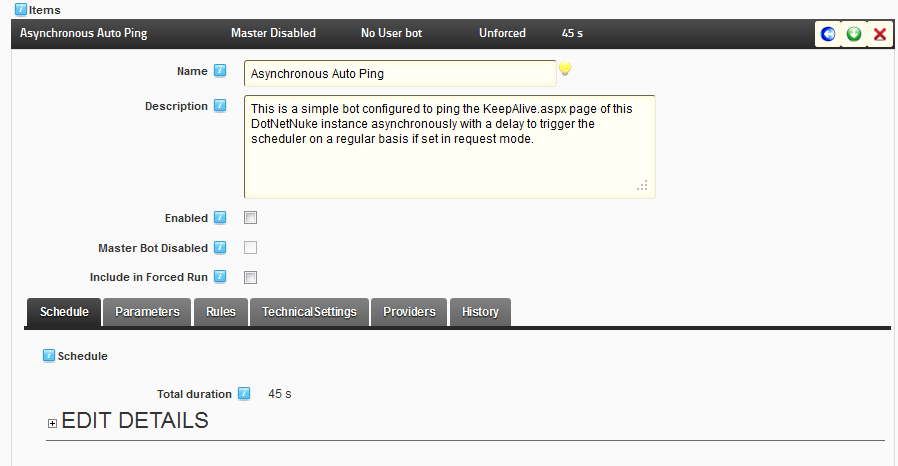
The PKP engine and its versatility is also used to power bots. You can define processes that will run at desired intervals, based on the same system of rules as the firewall part of PKP. In order to demonstrate how to enable and use bots, let’s activate a bot that keeps your portal “alive”. Let’s click on the Bots Farm tab, then select the Bots tab.



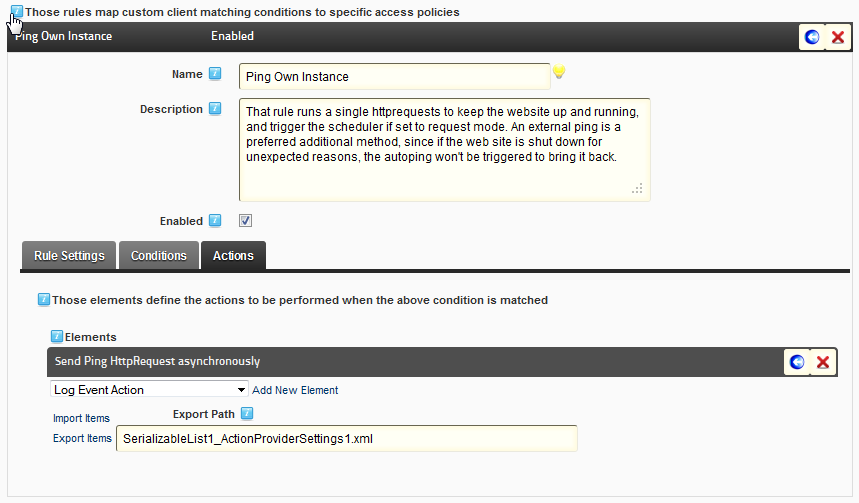
As in the Firewall section, PKP comes with bots already configured by default. You can install PKP and use it immediately with the bots provided or create your own. For now, let’s have a look at the bot that we will set up on this DotNetNuke instance. Click on the Asynchronous Auto Ping bot.

PROTIP: if your website experiences periods without any request from users, it may be unloaded from the server memory. The next user request will be handled very slowly because the server has to load your website back in memory before being able to answer the user. To keep your website alive you need to make requests at intervals that won’t let the server unload it from memory. DotNetNuke has a special page named KeepAlive.aspx that returns only the time and fulfills this need.

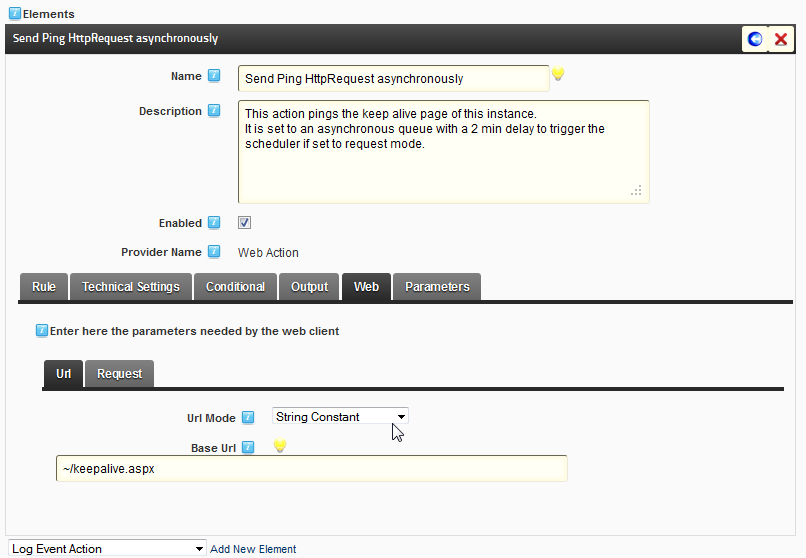
We are not going to change anything to the bot here, let’s just have a look at its parameters. Since a bot is based on the PKP engine, we are already familiar with some items.



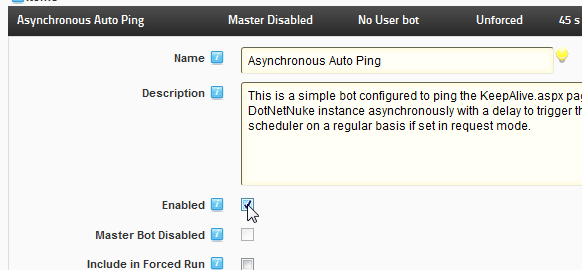
Apart from the Schedule and History tabs that handle the asynchronous nature of a bot, we already have encountered the same interface in the Firewall. Let’s change the scheduling: calling our website asynchronously every 45 seconds looks good enough, so we will focus on what the bot does. To see what conditions and actions this bot will undertake, we click on the Rules tab.

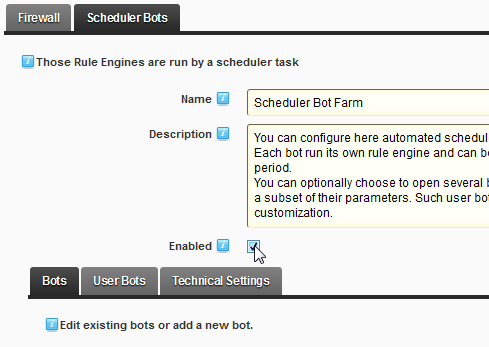


There is only one rule- Ping Own Instance. This rule does not contain any conditions so it will always execute its actions. There is one action that will make a web request to the “~/keepalive.aspx” url.

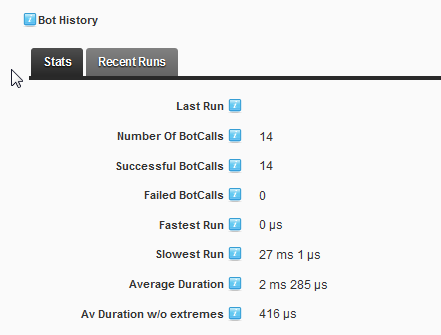


Let’s enable this bot and the bot farm before saving.





After some time, we can consult the History tab for the bot. We can see some statistics regarding the bot execution, and there is also a list of recent runs that you can consult:



There has been 14 bot calls, all of which were successful.

PROTIP: in the context of a bot, a success does not imply the task succeeded. It means that the bot followed all orders configured through the actions without encountering an error. If an action consists of sending an email to an invalid mailbox, the error will not be caught because of the asynchronous nature of email protocols.

This basic setup of PKP touched on the subject of firewall rules and bots. We have only skimmed the surface of what is possible with PKP and there are many other subjects to talk about. That is what we will cover in the following sections where we explain the concepts PKP is working with.

# Concepts

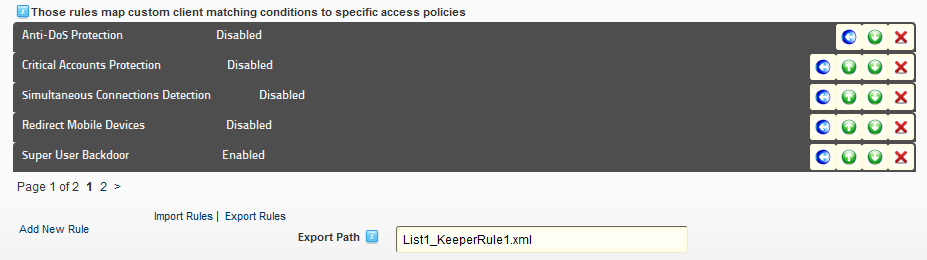
## Global concepts

### Lists, imports and exports

PKP is a configuration-intensive module so let’s cover immediately a very useful part of the module.

You will find a lot of lists in PKP; the rules, the conditions and the actions are displayed in ordered lists. The order might be important as some elements may depend on previous ones.

Below is a typical list interface:



4

3

2

1

1. The existing list items
2. The commands for the existing items
3. The new item command
4. The import and export commands and parameters

Each button has a specific role:

* export the content of a specific item 
* raise the item in the list (if available) 
* lower the item in the list (if available) 
* delete the item 

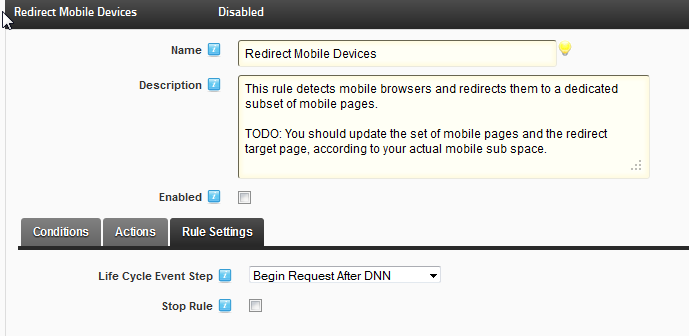
When you export an item, it is saved on the server as an xml file that will take the value of the “Export Path” configured under the list. The same value is used when you click on the Export button under the list. This time the file contains the whole list.

When you click on the Import button under the list, the content of the file is added to the list. No replacement will take place, so clicking on Export then Import will effectively double all your settings.

PROTIP: the default name for the import/export file is dependent of the type of item you are working with. So you can very easily copy an action from one rule to the other, just export the action you’re interested in, go to the rule where you want to inject the action you just exported and click import. Of course, this will not work if you change the name for the default file.

### Rules

At the heart of PKP lies the concept of rules. Rules are the basic bricks which are used to tell PKP what to do and when to do it. They are composed of conditions, actions and rules settings.



#### Settings

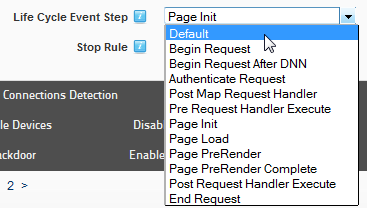
The rule settings gives you the ability to tell the rule when it must start working. The request lifecycle in ASP.Net sets up and handles all the necessary behaviors for a request. Activating your rule at the right moment is an important configuration step.

IMPORTANT: The steps used in PKP are a combination of:

- ASP.Net events (<http://msdn.microsoft.com/en-us/library/ms178472.aspx#lifecycle_events>)

- HTTP request steps (<http://msdn.microsoft.com/en-us/library/bb470252.aspx>)

They are displayed in the order they are evaluated.



Here are two notable points to consider when configuring this value:

* The Begin Request is used for the first handling of the request by PKP. DotNetNuke has not taken the request into account yet, which is why there are two Begin Request steps. If you want to use elements from the DotNetNuke framework, you must start your rule processing on Begin Request After DNN

By default, you should configure your rules to always start at the earliest moment possible.

* The Default value is a step that is useful for actions and conditions and is a reference to the step used by the wrapping context. Since the rule is a first class entity and has no wrapping context, setting a rule to evaluate on the Default value is equivalent to disabling it.

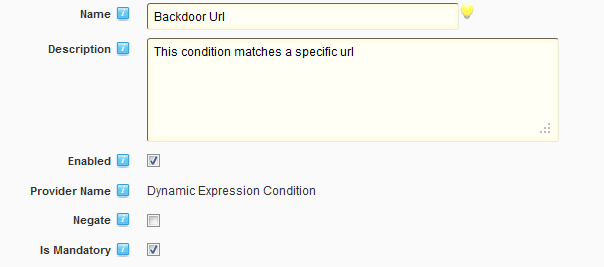
Stop Rule allows you to stop any other rule evaluation in the list if the rule matches. So if you don’t want PKP to continue evaluating rules after this one - for example if an user should be banned, it may not be useful to continue checking the other rules and use CPU cycles to process the requests he sends - you can check this parameter.

#### Conditions

Conditions are statements about what you expect the rule to react to. They allow you to express what will activate the rule via Boolean logic.

PKP comes with existing conditions which will be covered later on.

First, let us look at common condition settings.

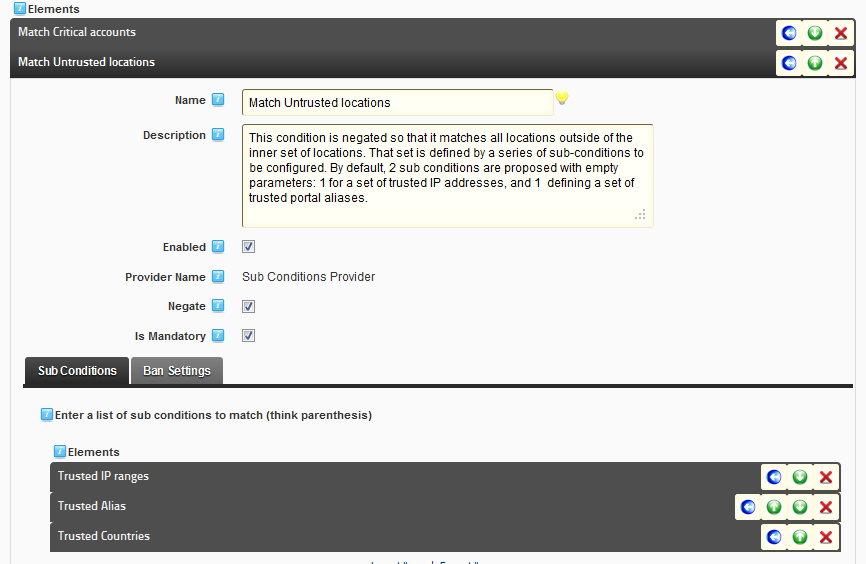


A condition has a name and a description. We recommend filling the name with a meaningful text as it is displayed under the Conditions tab for Rules so that you see all set up conditions at a glance.



The checkboxes give you options as to how the condition will be evaluated. Enabled gives you the option to remove the condition from evaluation. Negate inverts the result of the evaluation. Is Mandatory indicates to the rule that if this condition is false, the rule must not execute.

You can configure your rules to be evaluated against complex conditions. If you want to link conditions together, you must use the Subconditions type of condition that lets you combine child conditions.



Here is an example of conditions in the Critical Accounts Protection rule that comes with PKP by default. It can be expressed this way: this rule will be activated if the request matches a critical account AND the location of request does not come from a trusted IP, to a trusted alias, from a trusted country.

Under the common settings in a condition you will find tabbed controls. You can have one or more tabs.



The first tab is usually used for condition-specifics parameters. It is here that you can configure the values needed for this condition.

The Ban Settings tab only appears on conditions that are able to identify some users according to their configuration; you can configure these conditions so that they can be used as triggers to ban users if necessary. We will talk again about banning conditions in the Firewall concepts > Technical Settings > Banning section of this documentation.

#### Default conditions

PKP comes with a series of existing conditions providers. Here we are going to quickly list them for reference:

* Sub Conditions Provider: allows the user to combine sub conditions
* Static Condition: is provided as a sample to develop against
* Client Source Condition: matches the source that is making the request against different sources: IP, Forwarded IP, Country, Url
* Portal Alias Condition: matches the portal alias used in the url
* DotNetNuke Page Condition: matches the target DNN Page as defined in the url
* Membership Condition: matches the user membership
* Requests caps Condition: matches a maximum number of requests in a time window
* Multiple Connections Condition: matches connections from different locations for a DNN user
* Dynamic Expression Condition: matches a code expression written in Flee. This allows you to access complex properties by coding directly in the interface

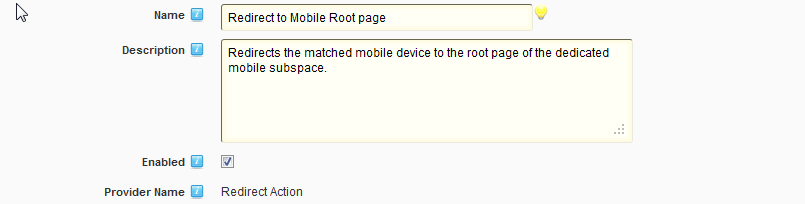
Each of these providers has its own settings that you will use to configure the condition when used in a rule.

#### Actions

Actions are what the PKP Rule will run if its conditions are satisfied.

PKP comes with existing actions which will be covered later on.

First, let us look at common action settings.



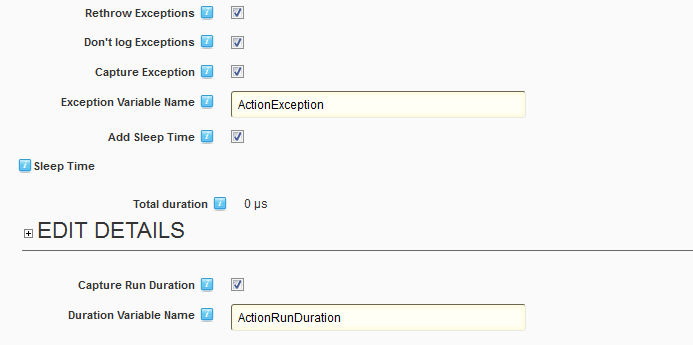
The name and description for an action fulfills the same objective as for a condition. The name is used in the interface so it needs to be meaningful. The Enabled Checkbox gives you the ability to quickly disable an action without deleting it from the configuration.

Below these settings are some tabbed controls. Here are for example the controls for a Web Action - which is able to make a request on the internet.



First 3 tabs are specific to this action. Let us describe what Technical Settings, Rule and Conditional are useful for.

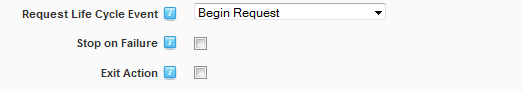
##### Technical Settings



The Technical Settings for an action give you the opportunity to configure how the action will execute and what it will handle:

* Rethrow exceptions: check if you want an exception to be passed to the upper level of error management in your portal.
* Don’t log exceptions: in some cases, logging the exception in an action is useless (for example in the case of a web action, you may not have any control regarding the endpoint you’re trying to reach and not be interested in the potential errors from a time out or a network error) so you can check this to avoid logging the exceptions.
* Capture exception & Exception variable name: you may want to pass an exception to another action in the rule for processing. Check and type the name of the variable that will be created with this exception.
* Add Sleep Time: some actions may need a cool down time before executing; you can check this parameter and enter the sleep delay for this action.
* Capture Run Duration & Duration variable name: you may want to pass information regarding the running time of the action to other actions in the rule for processing. Check and type the variable name that will be created with the running time value.

##### Rule



The Rule tab describes how the action will integrate in the rule it belongs to.

* Request Life Cycle Event : allows you to configure at what step the action will be executed. If an action is configured for execution at an earlier moment than its parent rule, it is not executed. However, if the action is configured to be executed at a later time than its rule, it will be executed at the configured step. The default parameter indicates that the action can be executed at the time its provider defines by default.
* Stop on failure: If the action fails, this value indicates whether or not the rest of the execution of the rule must be stopped.
* Exit action: if checked the action stops any other action from executing.

##### Conditional

The Conditional tab allows you to configure optional actions. It brings the conditions evaluation for the rule at the action level. You can define conditions that if evaluated to true will trigger the action. It also allows you to define an alternate action set that will be executed if the conditions don’t evaluate to a true value.



The same concepts as the ones we have covered regarding Rule edition apply here.

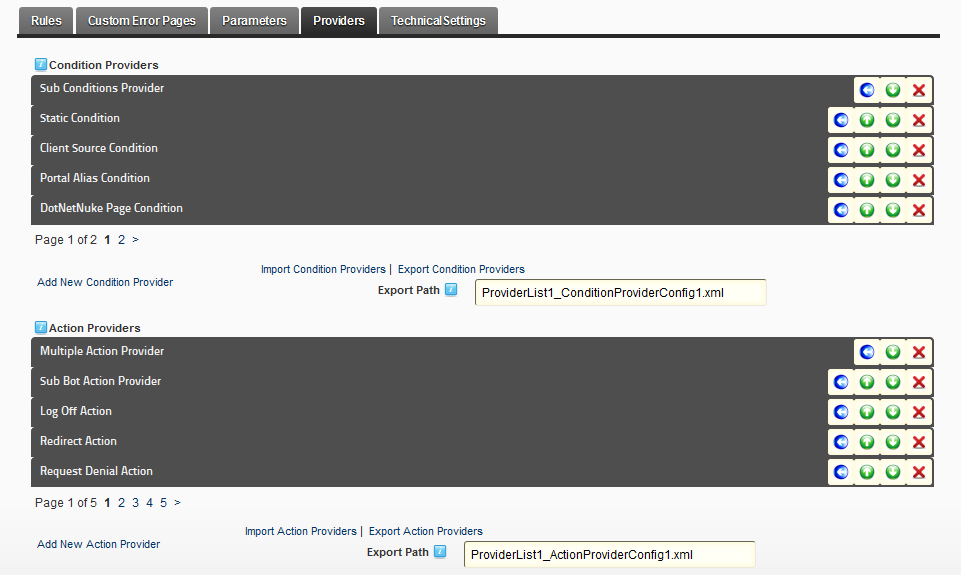
#### Default Actions

As for the conditions, PKP comes installed with a series of actions.

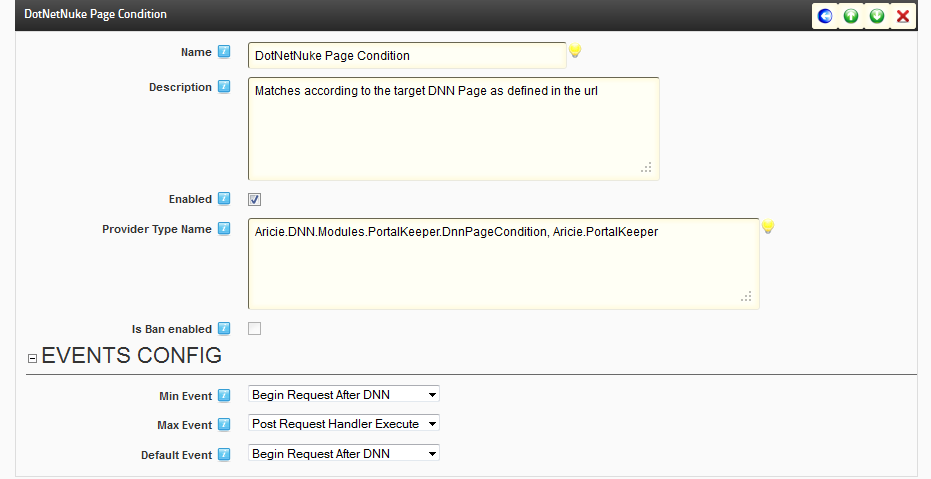
* Multiple Action Provider: Runs a sequence of actions
* Sub bot Action Provider: Runs a bot as an action
* Log off: signs the current user off
* Redirect: redirects the request to a given url
* Request Denial: blocks the request and return the configured status code or closes the connection
* Log event: adds an entry in the DotNetNuke log viewer
* Display message: displays a DotNetNuke message to the user, optionally in a target module.
* Send email: sends an email to a list of recipients, with optional token replacement.
* Auto login: logs the user with configured credentials
* Customize Environment: tweaks the DotNetNuke environment
* Empty action: no action provided, can be used for sleeping time
* Define variables: allows to declare and instantiate variables which will be passed to the next actions in the queue
* Object action Provider: allows to call objects methods and set object properties
* Loop action Provider: allows to loop over a custom collection to execute sub actions; the current item is passed through a variable
* While action Provider: allow to run a set of sub actions as long as a value evaluates to true
* Serialize / Deserialize action Provider: transforms a given entity to and from a serialized form
* String filter: allows string manipulation
* Web action: performs calls to a web address
* File read/write: reads/writes a file from/to the filesystem

### Providers

To add conditions and actions to your rules, you add their provider to the PKP engine. Every engine instance has a list of providers that you can configure. To access this list, you must go to the Providers tab in the engine configuration.



Apart from a name, description and enabled flag, a provider needs 2 important information: what class it must instantiate and what its range will be.



The Provider Type Name property is the name of a class type that will be used for instances of this provider. The name of the type uses the standard .Net way of naming classes (the full namespace and class name, followed by the assembly name after a comma).

The Events Config section is very important and shouldn’t be changed lightly. It controls the range of steps that are valid for a condition or an action. A condition or an action may make sense only during some steps for the request. For example there is no way to retrieve information from the DotNetNuke framework before it’s initialized, so if a condition needs to access DotNetNuke, it must start at the earliest during the “Begin Request After DNN”.

Since conditions are evaluated as soon as the rule executes, the default event property is not used. However for the conditions evaluation to succeed correctly, the rule event must match the event range of all its conditions. If this is not the case, an error will be logged in the event log of DotNetNuke.

As for a condition, an action can have an event range that will define the time when it’s possible to run it. But the action provider also uses the Default Event property. If an action is configured to execute on the Default event, it will then execute on the Default Event configured in the provider. If the provider is configured with the Default value, the action will then be executed on the same step as the rule.

## Firewall concepts

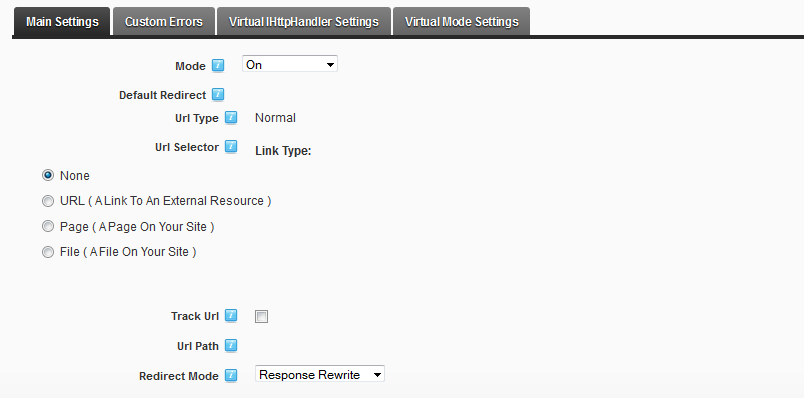
The firewall is probably the first usage of the PKP engine you will configure. Since it plugs directly in the request pipeline of your DotNetNuke website, it has the ability to react to these requests and add intelligence to the process. As such, it has some specific options which we will explore here.



### Custom error pages

The firewall can redirect users to specific error pages depending on various parameters. This configuration is done by modifying the web.config file of your website to follow the [customErrors tag syntax](http://msdn.microsoft.com/en-us/library/h0hfz6fc%28vs.71%29.aspx).

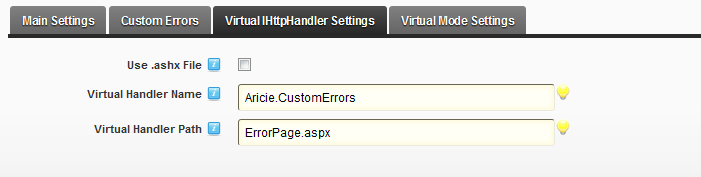
The Main Settings tab configure the customError tag directly, whereas the Custom Errors tab is used to manage individual error codes.



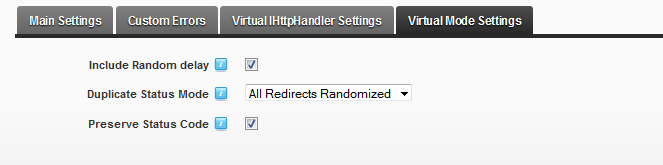
PKP does not limit itself to the modification of the web.config file. You can also define a virtual handler to manage the errors. The point of the virtual handler is to let you configure additional parameters. To use the virtual handler, go to the Custom Errors tab



You can configure some technical aspects of how the Virtual Handler will be registered in the web.config file under the Virtual HttpHandler Settings tab.

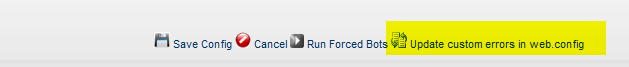


Finally the Virtual Mode Settings tab adds some configuration for the handler



Basically, these settings add a protection level to help mitigate [Oracle padding attacks](http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3332). However it is also recommended that you patch your server if possible.

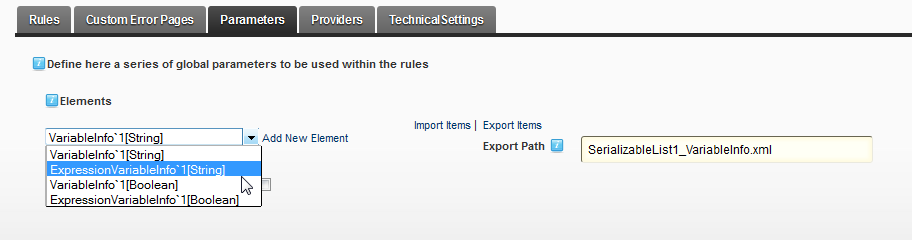
After changing values in the Custom Error Pages section, click on the Update custom errors in web.config link at the bottom of the interface to change your web.config file.



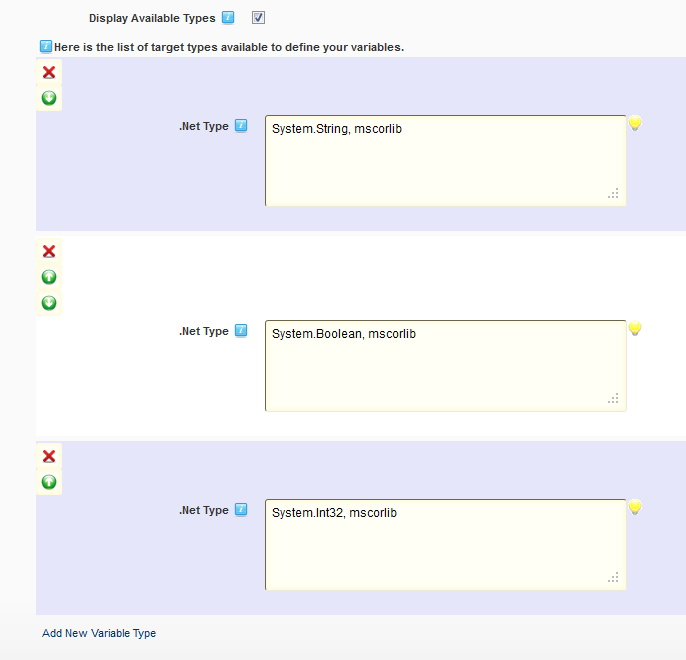
### Parameters

It is possible to define parameters for a PKP firewall. These parameters will be passed along to any rule configured in the engine.

To create a parameter, you select what .net type the parameter will wrap and add it to the parameters list. By default, PKP only lets you define parameters with a string or a Boolean value.

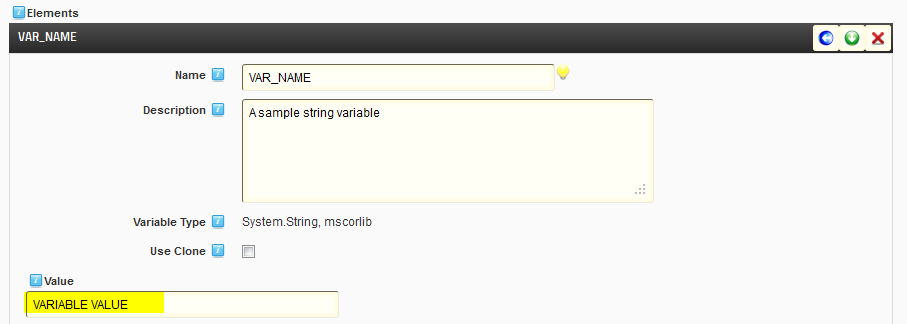


You can add additional parameter types by checking the Display Available Types checkbox in the interface and adding the type you want to wrap.



#### VariableInfo

The VariableInfo can be used as a kind of constant. It allows you to define a variable with a value you’ll assign during configuration. For example, I’m creating a string variable name VAR\_NAME with the value “VARIABLE VALUE”.

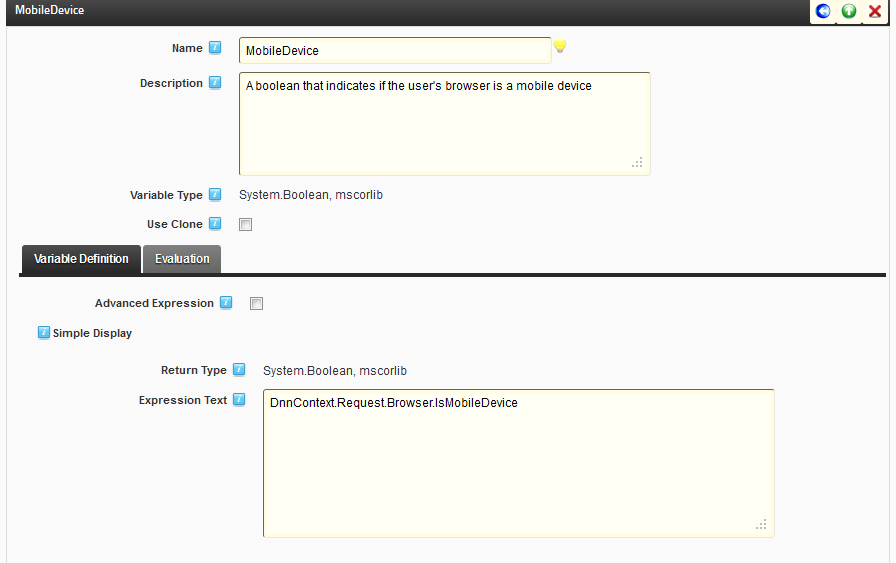


You could for example define a variable with the website name, to reuse it later in rules.

#### ExpressionVariableInfo

An ExpressionVariableInfo is a dynamic variable that you configure by writing code and that will be evaluated at runtime. The code is written using [FLEE](http://flee.codeplex.com/wikipage?title=LanguageReference).

Here I’m creating a Boolean variable by evaluating if the browser making the request is a mobile device.



Since all engines run with a default context, you can use properties from the context the engine is running in. The most interesting property you’ll want to explore would be the DNNContext: this property gives information about the DotNetNuke current request processing. It exposes among other properties the following

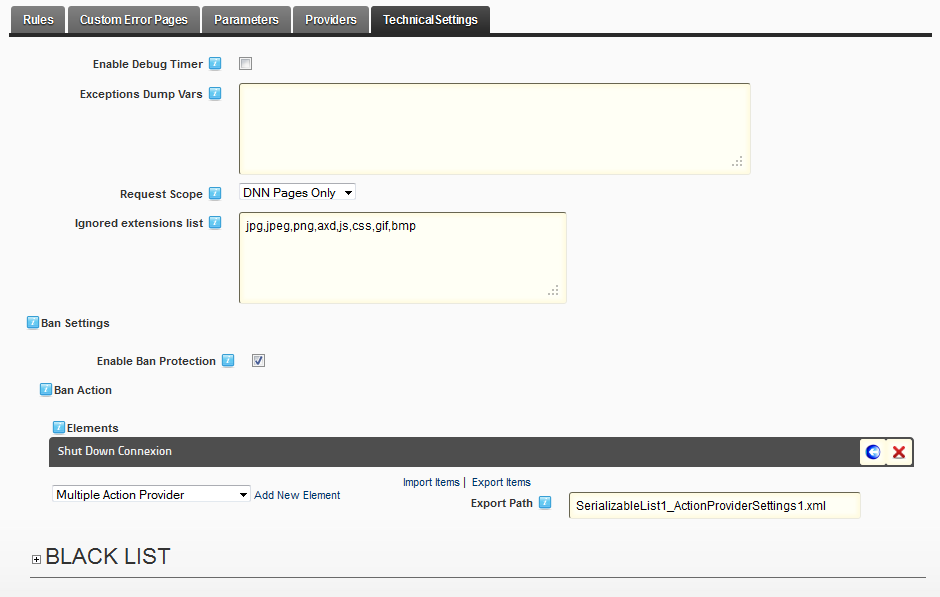
* Administrator: a UserInfo class with information about the administrator
* HostSettings: a hashtable with all the host settings
* Portal: a PortalSettings class with info about the portal
* Request: the current HttpRequest

The best way to explore all that is available is to look through the code and classes you’ll be able to work with. You can call custom classes, DotNetNuke classes, the PKP classes, etc.

You can also access the variables that may have been defined in previous steps, whether globally in the engine or in other steps. These values will depend on your configuration.

### Technical Settings

Technical settings allows configuration of basic behaviors for the Firewall.



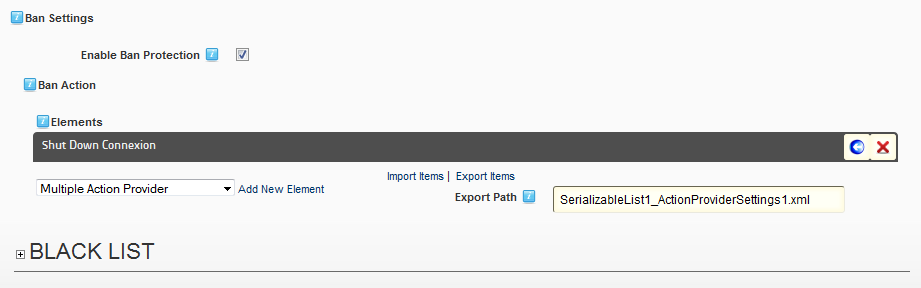
You can get information on the PKP process by logging information about its operations. To enable these logs, use the Enable Debug Timer.

The Exceptions Dump Vars parameter enables the engine to automatically retrieve data from an exception and place it in its data pipeline. You can define globally how the engine will process the exceptions that occur in its rules.

The Request Scope parameter narrows the focus of the PKP Engine to process only part of the requests that it will receive. This can in turn be fine-tuned by the Ignored extensions list parameter which will turn off the PKP engine for the listed extensions; it is filled by default with commonly served file types, but you should complete this list to reflect what is served from your website.

#### Banning

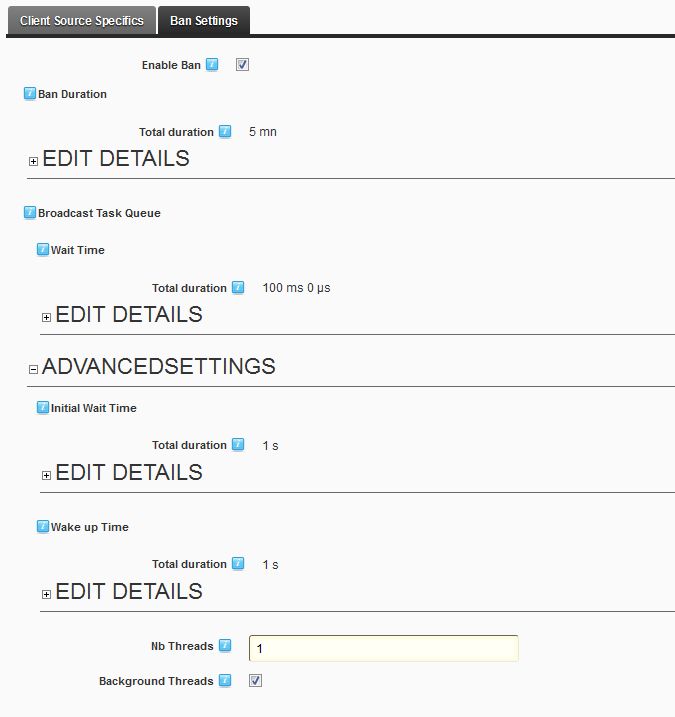
One useful and notable feature of the PKP firewall is its ability to quickly handle requests based on the condition system. Let’s take a look at how banning works in the context of the PKP firewall and how you can configure it.



The Technical Settings tab also contains the configuration for banning. You can enable banning globally at this level. It is also here that you configure the actions that will be executed when a user is banned. You can redirect its requests, add a warning message, etc. All the actions you have configured in the engine providers are available. By default, the connection is shut down.

PROTIP: The tab also contains the list of all currently banned users, along with the time when the ban will end.

To ban a user, you must make use of the Ban Settings tab some conditions will display. These conditions are the ones that are able to identify a user, either with an IP address, a Country, a request limit, etc.



You can enable the ban system if this condition is triggered, and add details about how long the ban will last. The other parameters are used to configure how the information about banned users will be distributed through the cache. It is used mainly for web farms.

When a banning condition is triggered, future requests will be evaluated against it immediately before entering the normal PKP workflow and if the condition matches again, the banning action in the Technical Settings of the PKP firewall engine will be run. So if you define a condition where ban is activated when a user comes from China, all users coming from China will be banned for the configured duration.

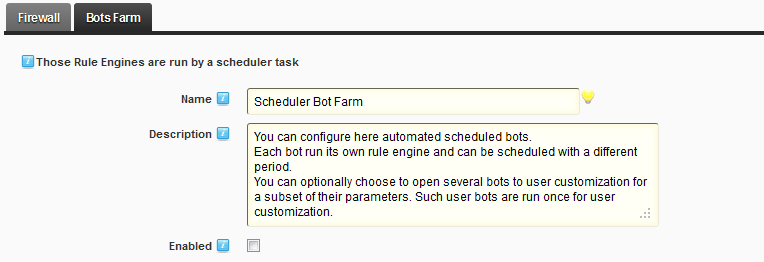
PROTIP: Since the banning conditions are evaluated as soon as possible by the engine, you can have a banning rule anywhere in your rule stack in the PKP engine. If it is triggered, it will be evaluated as soon as possible in future requests

Banning users will help your server handle malicious requests such as attempts at Distributed Denial of Services, Oracle attacks, brute-force guessing, etc.

## Bots concepts

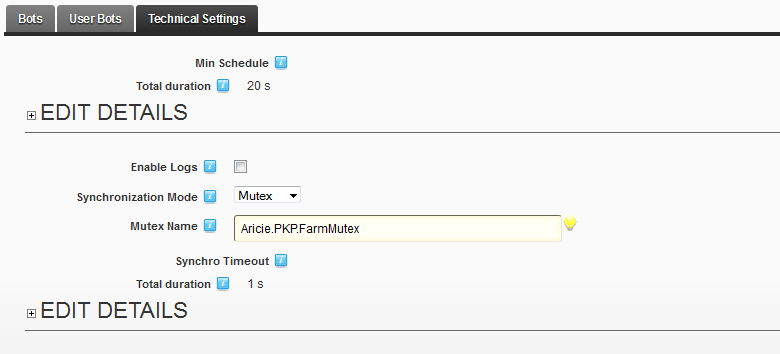
Bots are autonomous processes with a PKP engine embedded to handle configuration and rule processing. As such, most of their behavior and configuration can be inferred from what we have seen up to this point. However, they also have some specifics we need to go over.

The bot farm can be enabled under the Bots Farm tab in the configuration.



To run bots, you must enable the Bot farm. Once enabled, the bots will be run by a DotNetNuke scheduled task that was installed on your instance when you installed PKP. If you disable the scheduled task or change its settings, you impact the whole bot farm.

### Technical Settings

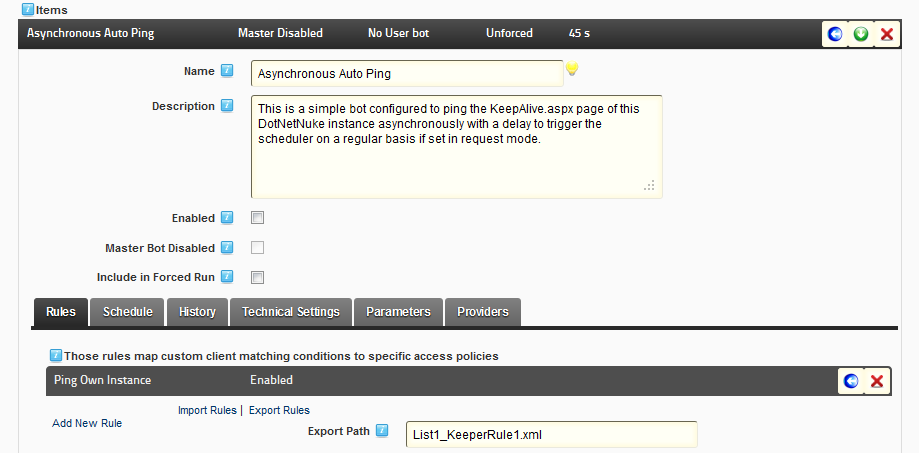


The Technical Details for the bots farm configures how it will integrate with the DotNetNuke scheduler. You can also enable logging for the bot farm with the Enable Logs parameter.

PROTIP: the default synchronization mode needs you to have full trust on the webserver. Before using the bot farm, change this value if you don’t have this level of credentials on the server.

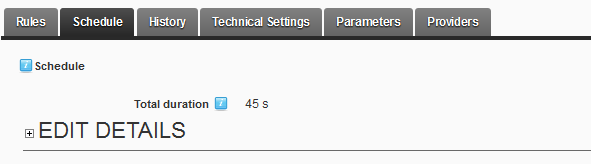
### Bots

As explained earlier, a bot is a process that embeds a PKP Engine. As such, part of its configuration is already familiar to the reader.



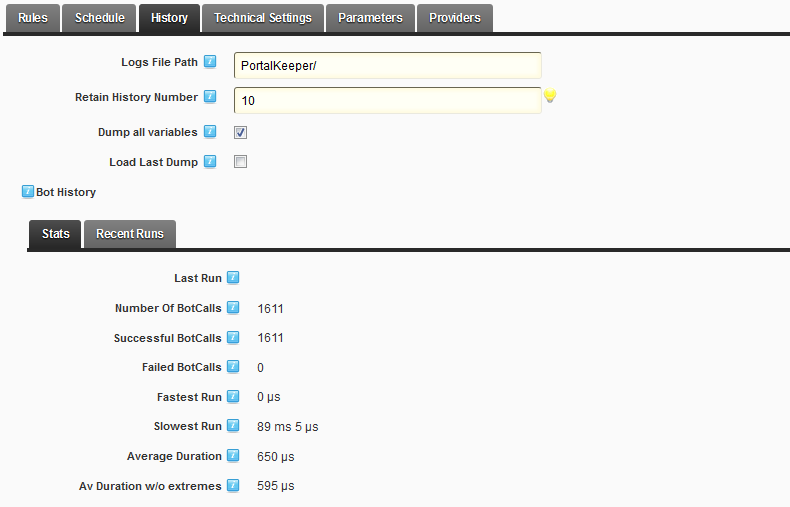
A bot has rules, which will work like the rules we have seen in the firewall, it has parameters and providers. We will concentrate on the following settings: Schedule, History and Technical Settings.

#### Schedule



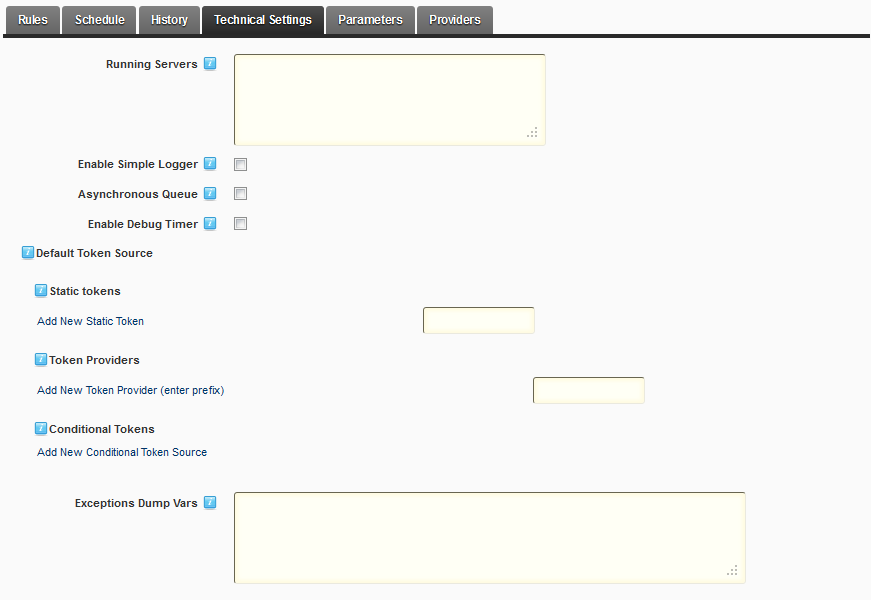
The Schedule settings allow the user to configure how long the bot will wait between runs. This value will complete the global scheduling parameters we’ve seen in the technical settings of the bot farm.

#### History



In the History section, you decide how long the bot farm will retain history between runs and how this information will be logged. This tab also contains statistics about the bot run and information about the latest runs.

#### Technical Settings



The bot technical settings allow you to configure what servers it must be run on. You can also activate the engine debug logging or activate a simpler logging mode that will limit the amount of information the bot outputs.

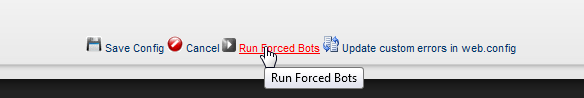
The Asynchronous queue option lets you run the bot asynchronously by configuring the delay, thread numbers and waiting times of a process queue.

#### Forced Run

During the configuration of your bot, you may want to run it without waiting for the scheduler to launch the bot. You can include the bot in a forced run by checking the corresponding parameter in the main bot configuration window.



After saving the changes, you can click the Run forced bots command to run the bot without waiting for the scheduler.



Since every bot that is included in a forced run will be launched when you click the command, make sure to uncheck the parameter after your tests are completed.

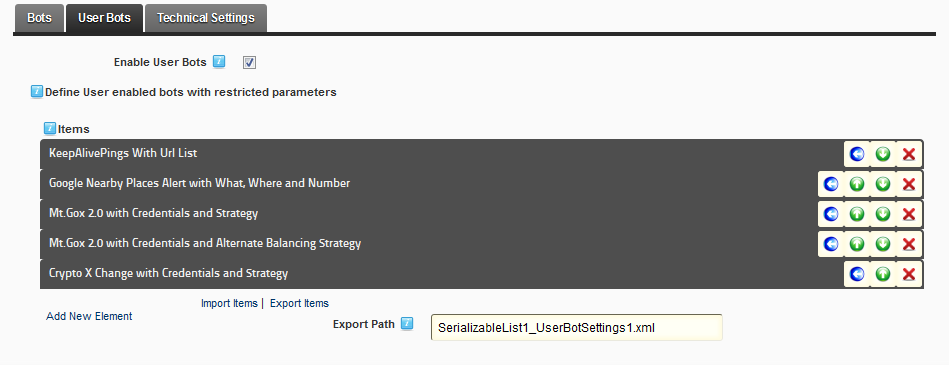
### Master and User Bots

A bot can be configured to run with global settings. But it can also be used as a “template” to run user bots. A user bot is a bot whose behavior - ie rules and actions - cannot be changed by the user but whose input are managed by the user. By letting the master bot accept parameters, the user can change these parameters to get the behavior he wants.

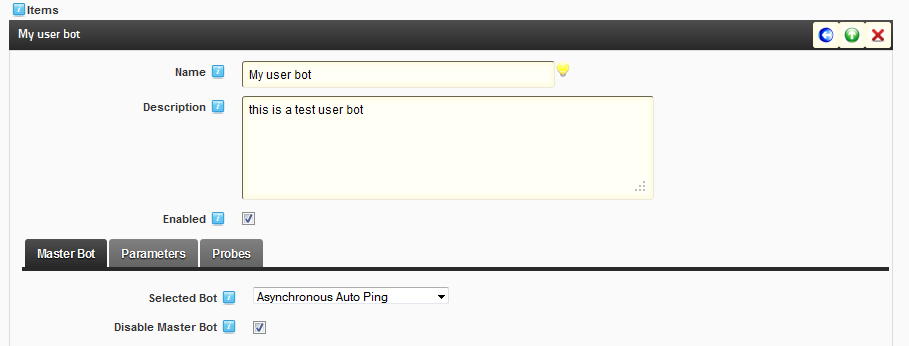
PKP comes with some user bots already configured as examples so you can try them out of the box. The main example of a user bot would be a trading bot. The master bot drives the main logic for the trading: querying the API, analyzing the results and sending orders. Each user can configure his own trading bot so the bot strategy is driven by the user parameters. Each user gets his unique bot.

#### Configuring a user bot

To create a user bot, you must add it to the user bot list under the User Bots tab.

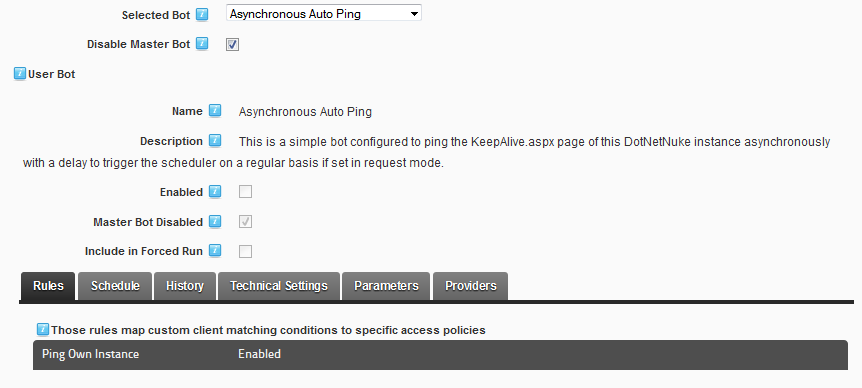


When creating a new user bot, you must give it a name and description. Then you must select a master bot it will be based on.

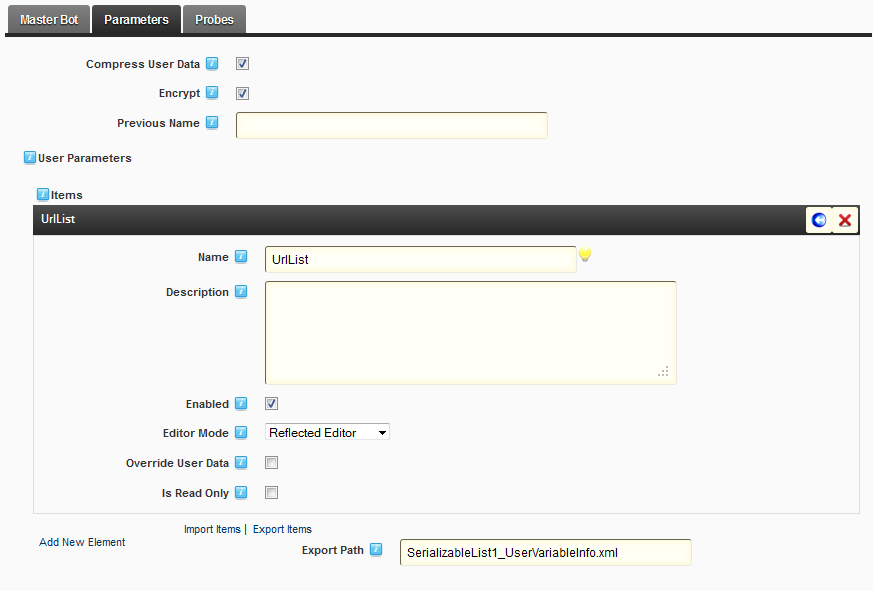


When declaring a user bot, you can decide to disable the master bot it is based on. Disabling the master bot means the only way to use this bot is if a user configures and enables it.

When you have selected the Master bot, its configuration is displayed below the configuration.

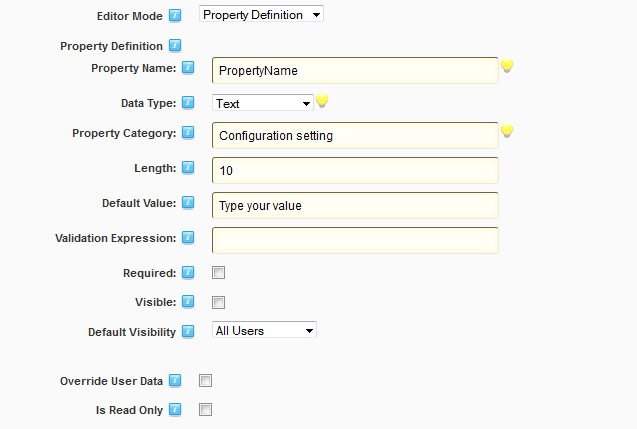


If you want your users to configure the user bot, you must now add parameters that will be accessible. For that, click on the Parameters tab.



Here you must add the master bot parameters your users will be allowed to configure. Add one parameter for each master bot parameter you want the user to customize using the same name as the source parameter. When adding a parameter, you can either choose to use the default editor mode from DotNetNuke or you can define the editor as you would in the profile editor in DotNetNuke.





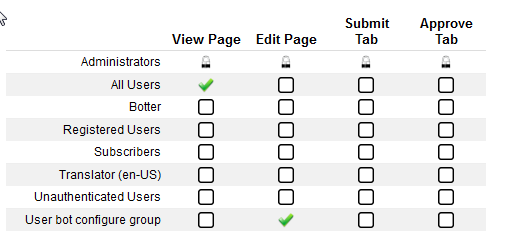
IMPORTANT: the reflected editor works if you are using a complex object. If you’re using basic .net classes, use the property definition mode for the variable, or your values won’t be updated.

You can also add probes to the user bot. These probes will fetch data from the user bot after it has run and will be able to return statistics which you’ll be able to define as public to create rankings between users.

#### Displaying a user bot

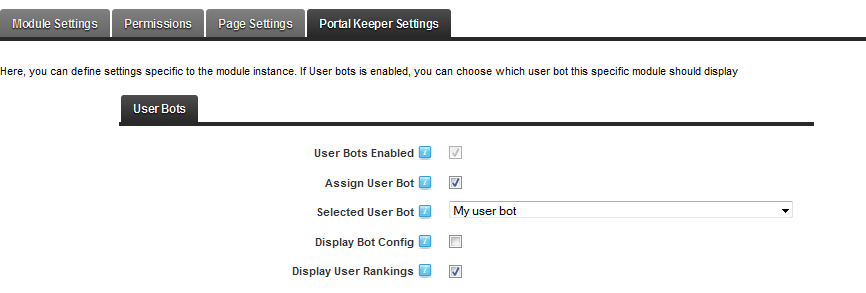
Once you have configured the parameters that must be filled by the user, you must display the configuration interface for the user.

Add the PKP module to a DotNetNuke page and grant access to the page to the users which will be able to configure the user bot. In this example I’ll use the role group “User bot configure group”

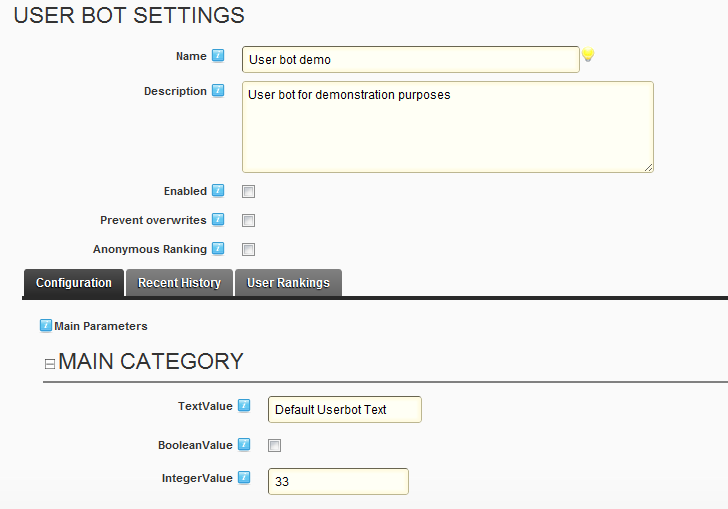


PROTIP: If the module inherits from the page permissions, you can configure the page, but you can also edit permissions for the module. Any user with view abilities will see a read only version of the bot. Any user with the Edit abilities on the module will be able to configure the user bot custom parameters.

Then edit the settings of the PKP module and select the PortalKeeper Settings tab



In this tab, check the Assign User Bot parameter and select the user bot you want the users to see. You can also configure the user bot so that it displays a read-only copy of the master bot settings for reference. Finally you can let the user bot display a ranking based on the Probes you have defined in the user bot.

Now if you go to your web page logged as your user, you will see that the user is able to configure the parameters of his bot.

He can enable his bot and protect his values from changes that may have occurred at the user bot level. He can also decide that if the bot has rankings enabled, he wants the statistics from his bot to be anonymous.

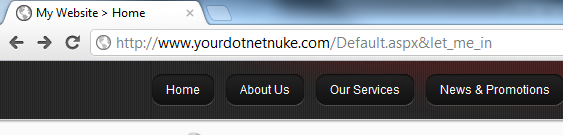
Below that, the user can configure his bot to change the parameters you configured as available to the user bot. He can also consult the history of his bot runs, and the statistics that are pulled from the bot runs.

# How to

## How to use the safe word to regain access to my website?

You have just activated by mistake a rule in PKP that logs off automatically anybody with an IP address containing a dot; now PKP automatically logs you off and in order to disable this rule in PKP you need to log in.

Fortunately, you can use the PKP “Recovery String” to disable PKP temporarily and regain access to the website. Just query your DotNetNuke instance with an url containing your recovery string.



The placement of the recovery string does not matter. Any url containing it will disable PKP temporarily, giving you plenty of time to disable the rule that poses a problem.

# Support

Our technical support team is committed to providing top support service. We will answer your enquiries within 48 hours max.

For best reactivity, please contact us on our Support forum: <http://www.aricie.com/en/resources/support.aspx>

**Need an special feature?** Just tell us what you need and we will make a free quote within 24 hours.

**Have a great Aricie experience with Aricie.PortalKeeper!**