Purpose

This document describes how to write custom configuration wizards in Nagios XI. This document will cover how to create a new wizard using a custom plugin and also cover how to utilize some of the programming tools within the Nagios XI framework.

Target Audience

This document is intended for use by Nagios XI Administrators and Developers wanting to create custom configuration wizards, and is intended for an audience that has some familiarity with programming and HTML.

Overview

The following topics will be covered in this document:

Setting Up The Development Environment

Developers have unique preferences as to how to set up their development environment in PHP however, the following setup will be the simplest way to work with, and debug a wizard while it's in development.

Open the `/etc/php.ini` file in your preferred text editor, and change the existing settings to match the following:

```
error_reporting  =  E_ALL; & ~E_WARNING
display_errors = On
ignore_repeated_errors = On
```

After making these changes the **`httpd`** service will need to be restarted.

RHEL/CentOS 6.x:

```
service httpd restart
```

RHEL/CentOS 7.x:

```
systemctl restart httpd.service
```

This will generate error output directly to the browser window, but also create enough filtering so that the error output is easier to decipher. Before proceeding to the actual wizard code structure, see the section below on Debugging Tips to save time in the development process.

XI Wizard Development Guidelines

The development guidelines for Nagios XI Wizards are still somewhat loosely defined, but the following conditions will maximize compatibility, security, and reliability of the Configuration Wizard. Contact the Nagios XI Support Team if you have questions about your code, and see php.net for the best reference on PHP syntax and built-in functions.

- Wizards must be free from all **fatal**, **syntax**, and **notice** error messages. This includes accounting for undefined variables and array indices.
- Wizards should *never* run **`UPDATE`** or **`INSERT SQL`** queries directly into the `nagios` or `nagiosql` databases. This will have unpredictable results and will most likely break a monitoring configuration.
- To maintain security within Nagios XI, avoid interacting directly with the `$_POST`, `$_GET` or `$_GLOBALS` arrays. To access variables submitted in wizard forms, use the `grab_array_var()` function as documented below. The `$inargs` array contains all of the **`POST`** data from each stage of the form.
    - `$form_variable = grab_array_var($inargs, $variable_name, $default_value)`
    - This will use some of the security feature built into Nagios XI to clean any input variables and prevent XSS vulnerabilities.
    - There is an exception to this in the wizard example, but the input variable is processed and cleaned of vulnerabilities.
- Wizard data can be passed forward with either a `$_SESSION` array, which is demonstrated in the `wizarddemo.zip`, or also by serializing the data and passing it along through hidden form inputs.
    - Serializing the data which will be seen in most wizards prior to Nagios XI 2011R1.3.
    - The new `$_SESSION` method is simpler for repopulating the form if a user selects the **Back** button.

Sample Wizard Code

This document does not go into great detail on how to structure or write PHP code for a wizard. A sample wizard has been created that can be used in conjunction with this document and used as the basis of a new wizard.

You can download the example wizard code using the following link:
https://assets.nagios.com/downloads/nagiosxi/wizards/wizarddemo.zip

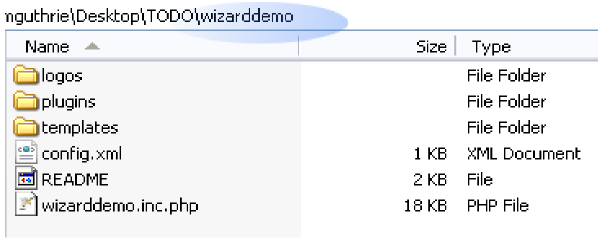
Upload it via **Admin** > **System Extensions** > **Manage Config Wizards**. After uploading:

- You will see it in the list of wizards called **Weather Alerts**.
- The wizard will be extracted and placed into the location `/usr/local/nagiosxi/html/includes/configwizards/` in it's own directory (explained in the next section)

Wizard File Structure Overview

The files in a configuration wizard must be structured and named according to the following conventions in order for the wizard to function correctly. The following example will demonstrate naming conventions for a wizard with the name wizarddemo. See the zip file from the **Sample Wizard Code** for more details.

All files must be placed inside of a directory called **`wizarddemo`** and the entire directory should be zipped into a file called `wizarddemo.zip`.



```
nguthrie\Desktop\TODO\wizarddemo
Name                    Size  Type
logos                         File Folder
plugins                       File Folder
templates                     File Folder
config.xml              1 KB  XML Document
README                  2 KB  File
wizarddemo.inc.php     18 KB  PHP File
```

PHP SCRIPTS

- The main include file must be named `wizarddemo.inc.php`
- Any additional includes referenced by the `wizarddemo.inc.php` file don't require a naming convention

LOGOS

- Must be placed in the logos directory and named `wizarddemo.png` or `wizarddemo.jpg`, etc
- The image size for the wizard logo should be 40 x 40 px

PLUGINS

- Check plugins can be placed in a directory called plugins and must match the name specified in the `config.xml` file

CONFIG TEMPLATES

- Config templates and definitions must be placed in a **`templates`** directory and defined in a file named `wizarddemo.cfg`
- New host or service template definitions must be named with the following convention:
  - `xiwizard_wizarddemo_service`
  - `xiwizard_wizarddemo_host`

ADDITIONAL FILES

Any additional files that are not PHP scripts must be specified in a file called `config.xml`. Example of the wizarddemo's `config.xml` below:

```
<configwizard>
     <templates>
          <template filename="wizarddemo.cfg" />
     </templates>
     <plugins>
          <plugin filename="check_cap" />
     </plugins>
     <logos>
          <logo filename="wizarddemo.png" />
```

```
    </logos>
</configwizard>
```

How A Wizard Works

A wizard works through stages, it presents some information and choices and a **Next** button, and you get called back at the next stage when the user presses the **Next** button. You get callbacks to this one very important PHP function which you supply:

```
YourWizardName_configwizard_func
```

For example:

```
function wizarddemo_configwizard_func($mode="",$inargs=null,&$outargs,&$result)
```

You get passed in the various bits of info that the user filled in on the previous page. Since you get called back several times in the typical wizard, you need to save the info from previous wizard stages, a session array is a good place to put this info and it's conventionally named **`$_SESSION`**.

Using A Session Array For Wizard Data

The wizarddemo code demonstrates in detail how to use a `$_SESSION` array to work with wizard stages and data. The following code is an example of how to establish a session array for a wizard in the stage: `CONFIGWIZARD_MODE_GETSTAGE1HTML`.

```
//check to see if this is a fresh wizard run, or if we're coming back from a
later stage
$back = htmlentities(grab_array_var($_POST,'backButton',false),ENT_QUOTES);

//clear any previous session data for this wizard, start a new session array
if(!$back)
{
        unset($_SESSION['wizarddemo']);
        //create a new session array to hold data from different stages
        $_SESSION['wizarddemo'] = array();
}
```

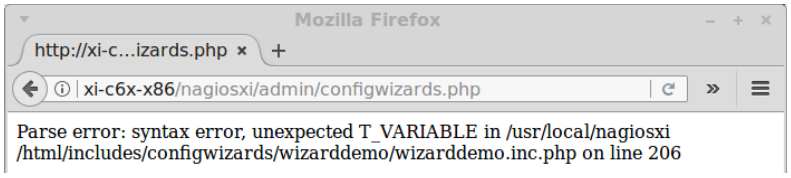
Debugging Tips

When working on a wizard it helps to have multiple browser windows/tabs open to the following pages:

- **Admin** > **System Extensions** and then right click on the **Manage Config Wizards** link and select **Open In New Tab** (or window)
- **Configure** and then right click on the **Run a Configuration Wizard** link and select **Open In New Tab** (or window)

Both of these tabs/windows will be beneficial when testing and developing the wizard.

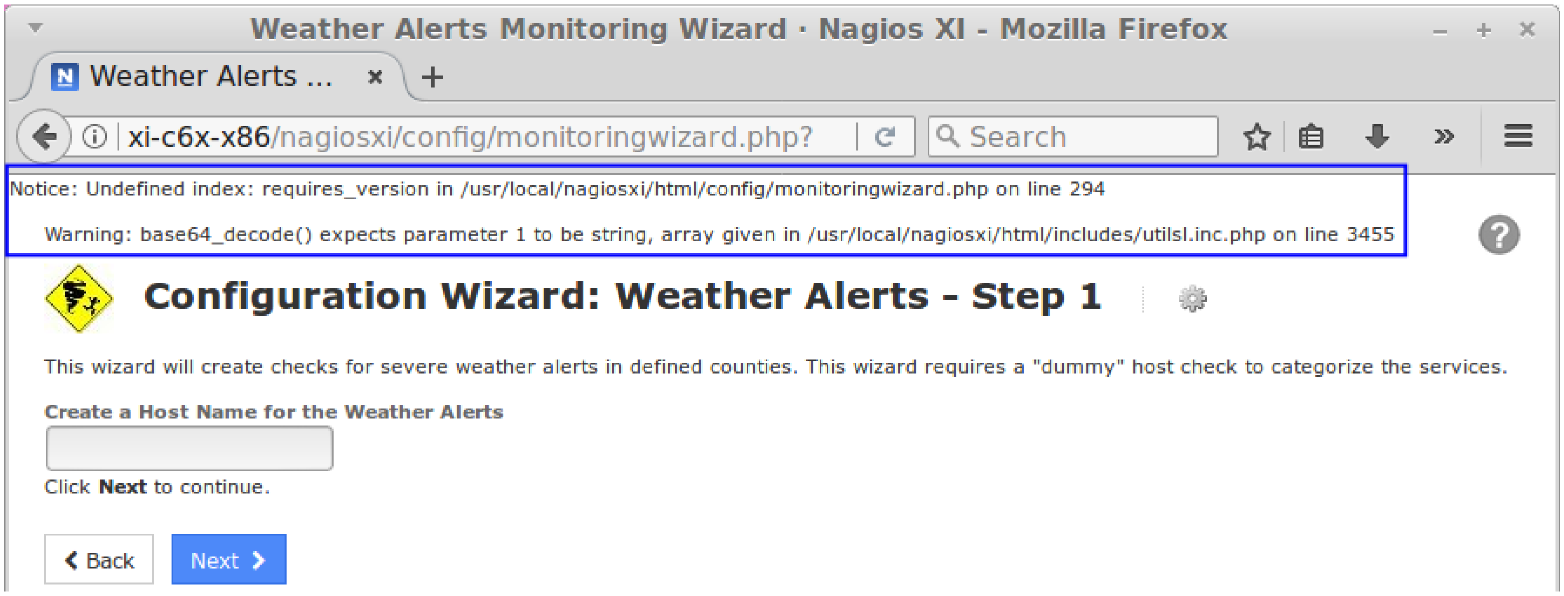
While developing the wizard, we recommend developing the code on a local workstation and then uploading the zip file periodically to ensure the full functionality and compatibility of the wizard. If you were to access the **Manage Config Wizards** page directly and you uploaded a wizard that had a php syntax error on it, you should clearly see the error output on screen. This allows you to clearly identify any fatal errors in the code before proceeding to the stages of the wizard. The screenshot below shows the error and directs you to the line number in the file.



```
Parse error: syntax error, unexpected T_VARIABLE in /usr/local/nagiosxi
/html/includes/configwizards/wizarddemo/wizarddemo.inc.php on line 206
```

To fix the syntax error, modify your local copy, rezip the directory, and then upload the wizard again. The old wizard will simply be overwritten. Once the wizard loads with no syntax errors, leave this page open, but select the tab/window opened to the **Run a Configuration Wizard** page. You will likely have to upload the wizard several more times until debugging is complete.

For debugging the wizard stages (`monitoringwizard.php`), error outputs will appear at the top of the page. Be sure to account for any undefined variables to prevent the Apache logs from being cluttered with error messages, and also to prevent bugs in the wizard itself. The screenshot below is an example of this.


Weather Alerts Monitoring Wizard · Nagios XI - Mozilla Firefox
Weather Alerts ...
xi-c6x-x86/nagiosxi/config/monitoringwizard.php?
Search

```
Notice: Undefined index: requires_version in /usr/local/nagiosxi/html/config/monitoringwizard.php on line 294

Warning: base64_decode() expects parameter 1 to be string, array given in /usr/local/nagiosxi/html/includes/utilsl.inc.php on line 3455
```

Configuration Wizard: Weather Alerts - Step 1

This wizard will create checks for severe weather alerts in defined counties. This wizard requires a "dummy" host check to categorize the services.

Create a Host Name for the Weather Alerts

Click **Next** to continue.

Every time you upload a new wizard into Nagios XI, select the URL for the `monitoringwizard.php` page, and press **Enter** to start the wizard over from stage one, and to clear any POST variables that will affect wizard navigation. You will likely need to repeat the process of rezipping the files, uploading to Nagios XI, and restarting the wizard several times, so leave both tabs/windows open while you develop.

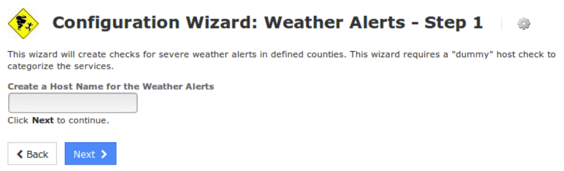
You might find the repeated process of rezipping the files, uploading to Nagios XI slightly annoying. You could write a script that copied the files from your workstation to the XI server using a protocol like SCP. The location that config wizards resides on the Nagios XI server is `/usr/local/nagiosxi/html/includes/configwizards/`.

It is advised however that your final testing follows the process of of rezipping the files and uploading to Nagios XI. It's advisable to use the **delete** function on the **Manage Config Wizards** to ensure a fresh upload of your wizard works.

Overview Of Configuration Wizard Stages

The Nagios XI wizard framework is designed to handle user-defined wizard stages in the following progression. Each of these stages are prepended with `CONFIGWIZARD_MODE_`

`GETSTAGE1HTML:`	HTML header reads this as **Step 1**
`VALIDATESTAGE1DATA:`	Form validation for stage 1 HTML



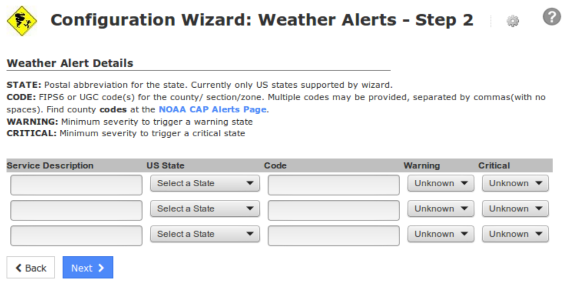
Configuration Wizard: Weather Alerts - Step 1

This wizard will create checks for severe weather alerts in defined counties. This wizard requires a "dummy" host check to categorize the services.

Create a Host Name for the Weather Alerts

Click **Next** to continue.

`GETSTAGE2HTML:`	HTML header reads this as **Step 2**
`VALIDATESTAGE2DATA:`	Form validation for stage 2 HTML



Configuration Wizard: Weather Alerts - Step 2

Weather Alert Details

STATE: Postal abbreviation for the state. Currently only US states supported by wizard.
CODE: FIPS6 or UGC code(s) for the county/ section/zone. Multiple codes may be provided, separated by commas(with no spaces). Find county **codes** at the **NOAA CAP Alerts Page**.
WARNING: Minimum severity to trigger a warning state
CRITICAL: Minimum severity to trigger a critical state

Service Description	US State	Code	Warning	Critical
	Select a State		Unknown	Unknown
	Select a State		Unknown	Unknown
	Select a State		Unknown	Unknown

`GETSTAGE3HTML:`	HTML header reads this as **Step 3**, the **Monitoring Settings**.
`VALIDATESTAGE3DATA:`	Form validation for stage 3 HTML
`GETSTAGE3OPTS: [optional]`	Allows **check settings** to be hidden and/or overridden (**Step 3**)
`GETSTAGE4OPTS: [optional]`	Allows **alert settings** to be hidden and/or overridden (**Step 4**)
`GETFINALSTAGEHTML:`	A final stage to confirm the Apply Configuration for the new settings

Overriding Stages

Currently, the wizard framework allows for a limited amount of customization to the later stages of the Configuration Wizard, primarily the check settings, alert settings. These override options have been documented internally in the wizarddemo example code for the available options and how to use them. Future versions of Nagios XI may include options to skip entire stages, and allow for manual definition of hostgroup, servicegroup, and parent relationships.

Finishing Up

This completes the documentation on guidelines for writing custom Wizards in Nagios XI.
If you have additional questions or other support related questions, please visit us at our Nagios Support Forums:

https://support.nagios.com/forum

The Nagios Support Knowledgebase is also a great support resource:

https://support.nagios.com/kb