



Software Testing With Testopia

By Greg Hendricks

gregaryh@gmail.com

<http://mozilla.org/projects/testopia>

Table of Contents

| | |
|--|----|
| Introduction..... | 5 |
| Testopia - Test Case Management..... | 5 |
| Testopia and Bugzilla..... | 5 |
| The Mozilla Foundation and the Testopia Project..... | 5 |
| Installation..... | 6 |
| Requirements at a Glance..... | 6 |
| Recommended version Where to get it..... | 6 |
| Database..... | 6 |
| Additional Perl Modules..... | 6 |
| Libraries..... | 6 |
| Client..... | 6 |
| What You Need to Know..... | 6 |
| Downloading and Installing..... | 7 |
| Upgrading From Testopia 1.0.x..... | 7 |
| Configuration..... | 7 |
| Using Testopia..... | 8 |
| Why Test?..... | 8 |
| What Types of Testing can Testopia Help You With?..... | 9 |
| Black Box Testing..... | 10 |
| White box and Automated Testing..... | 10 |
| Getting Started..... | 10 |
| The Big Picture..... | 10 |
| Test Plans..... | 11 |
| Test Cases..... | 11 |
| Test Runs..... | 11 |
| Test Run Environments..... | 11 |
| Builds..... | 12 |
| Test Case-Runs..... | 12 |
| Testing In Testopia..... | 12 |
| Start With a Plan..... | 12 |
| Viewing A Test Plan..... | 13 |
| Attaching Files..... | 13 |
| Editing Plan Fields..... | 13 |
| Viewing Plan History..... | 13 |
| Adding Categories and Builds..... | 13 |
| Creating Test Cases..... | 14 |
| Shortcut – Creating a Test Case..... | 14 |
| Adding and Removing Components and Tags..... | 15 |
| Viewing Test Case-Run Results..... | 15 |
| Attaching Files..... | 15 |
| Attaching Bugs..... | 15 |

| | |
|---|----|
| Editing Case Fields..... | 15 |
| Test Case Dependencies..... | 15 |
| Creating Environments..... | 15 |
| Environment Administration..... | 16 |
| Categories..... | 16 |
| Elements..... | 16 |
| Properties..... | 17 |
| Property Values..... | 17 |
| Creating Your Environment..... | 17 |
| Creating A Test Run..... | 18 |
| Shortcut – Creating a Test Run..... | 18 |
| Viewing Your Test Run | 18 |
| Adding Cases..... | 19 |
| Editing Test Run Fields..... | 19 |
| Running Your Tests..... | 19 |
| Filtering Your Test Cases in a Run..... | 19 |
| Sorting Test Cases..... | 20 |
| Passing and Failing Test Cases..... | 20 |
| Adding Notes..... | 20 |
| Attaching Bugs..... | 20 |
| Reassigning Tests..... | 20 |
| Changing Build or Environment on a Test Case..... | 21 |
| Classic View..... | 21 |
| Deleting Case-Runs..... | 21 |
| Updating Multiple Cases at Once..... | 21 |
| Wrapping Up..... | 22 |
| Getting Around Testopia..... | 22 |
| Searching..... | 23 |
| Sorting Search Results..... | 23 |
| Paging..... | 23 |
| Batch Processing..... | 23 |
| Saved Searches..... | 23 |
| Reporting..... | 23 |
| Plan Reports..... | 24 |
| Build Coverage Report..... | 24 |
| Top Bugs..... | 24 |
| Bugs Found In This Plan..... | 24 |
| Printable Percentage Report..... | 24 |
| Test Case Reports..... | 24 |
| Estimated vs Actual Time..... | 24 |
| Historic Status Breakdown..... | 24 |
| Cloning..... | 24 |
| Cloning Test Plans..... | 25 |
| Cloning Test Cases..... | 25 |

| | |
|---|----|
| Cloning a Run..... | 25 |
| Deleting..... | 26 |
| Test Cases..... | 26 |
| Test Plans..... | 26 |
| Test Runs..... | 26 |
| Importing and Exporting..... | 26 |
| Importing Test Cases..... | 27 |
| Testopia Security..... | 27 |
| The Testers Group..... | 27 |
| Test Plan Access Control Lists..... | 27 |
| User Regular Expression..... | 27 |
| Explicit Inclusion..... | 28 |
| Access Rights..... | 28 |
| Read | 28 |
| Write | 28 |
| Delete | 28 |
| Admin | 28 |
| Tags..... | 29 |
| Adding Tags to an Object..... | 29 |
| Viewing Tags..... | 29 |
| Using Testopia With Automated Test Scripts..... | 29 |
| Field Descriptions..... | 29 |
| Test Plans..... | 29 |
| Test Cases..... | 30 |
| Test Runs..... | 32 |
| Test Case-Runs..... | 33 |
| Glossary..... | 35 |
| Getting Help..... | 38 |
| Bibliography..... | 39 |

Introduction

Testing, especially software testing, can be a tedious and overwhelming task. Often there are many levels of testing requirements and many iterations during a project life cycle. Keeping track of the results of tests is important both to the customer, and the developer. Customers want to know that their requirements are being met and developers want to know if there are bugs that they missed. Testing is an important part of the software development process since “Software should be predictable and consistent, offering no surprises to users” (Meyers 2004, ch 1).

Testopia - Test Case Management

Test case management is the process of tracking test outcomes on a set of test cases for a given set of environments and development iterations. To do this effectively, an organizational structure needs to be set in place to track test cases and their outcomes in a given test scenario. Testopia was developed to provide a central repository for the collaborative efforts of distributed testers. It serves as both a test case repository and management system. Testopia is designed to meet the needs of software testers from all sizes of groups and organizations.

Though Testopia was designed primarily for software testing, it can be used to track any type of test cases. Also, being open source, Testopia can be easily adapted to fit just about any testing model.

Testopia and Bugzilla

Mozilla's Bugzilla bug tracking system is one of the most popular open source issue tracking systems available. Bugzilla “provides an easy to use, easy to maintain and cost effective solution with a rich feature set that easily can compete with its proprietary counterparts” (Lohmeyer).

Test cases are, and should be, closely tied to defects. As defects are found, test cases should be written to verify that the defect is fixed in future releases. To this end Testopia was designed as an add-on extension to Bugzilla. This allows a single user experience and point of product management for both defect tracking and test case management.

The Mozilla Foundation and the Testopia Project

Testopia began life as Bugzilla Testrunner. It was written by Maciej Maczynski in 2001 and later ownership passed to Ed Fuentetaja who passed development duties to Greg Hendricks in 2006. It was renamed Testopia in March 2006 prior to release of version 1.0 in May. It was adopted as a Mozilla project and now resides at <http://mozilla.org/projects/testopia>.

Since May 2006, Testopia has been increasing in popularity among testers in the open source community as well as in other organizations. In a recent blog posting, Rosie Sherry, a respected software testing expert, states that “This is probably the most comprehensive open source test management system and once up and running provides everything needed of a test management system” (Sherry 2006).

Installation

Requirements at a Glance

| | Recommended version | Where to get it |
|--------------------------------|---------------------|---|
| Bugzilla | 2.22.2 | http://bugzilla.org/downloads |
| Database | | |
| MySQL | 5.0 | http://mysql.com |
| Additional Perl Modules | | |
| JSON | | http://cpan.org |
| Text::Diff | | http://cpan.org |
| Libraries | | |
| Dojo toolkit | 0.4.1 | http://dojotoolkit.org/ |
| Client | | |
| Firefox | 2.0+ | http://getfirefox.com |

What You Need to Know

As our development has moved forward, we have decided to try to keep abreast of the latest stable release from Bugzilla (currently 3.0). This gives us a stable code base to work from. Developing plugins or extensions for any software is like trying to hit a moving target. This decision allows us to focus our time more on releasing new features often and early rather than back porting. However, this means that most major feature will not be back ported to earlier versions unless and until we have time to do so. Anyone wishing to help in this effort is more than welcome.

Though Bugzilla officially supports MySQL and Postgres databases, Testopia has only been tested with MySQL. Patches are welcome to make Testopia Postgres compliant.

Likewise, Testopia has only been thoroughly tested using Firefox browsers. This is not because we don't care about other browser platforms, but because we lack the time and resources to attempt thorough testing in other browsers. Testopia should run fine on mast modern browsers. If issues arise however, please log them at bugzilla.mozilla.org

Testopia makes use of some cutting edge AJAX technology provided by the Dojo toolkit. Our philosophy is that it is better to focus more processing on the client than to tie up the server. Dojo

places some extra weight on the client, but makes interactions with the server faster. This increases the load on the client machine however, which means the need for faster processing power and more memory. Trying to run Testopia on a Pentium 2 with 128 MB of RAM will probably not be the best user experience.

Downloading and Installing

Testopia is an extension to Bugzilla. It goes without saying then that you should have Bugzilla up and running first. Instructions for installing Bugzilla can be found at <http://bugzilla.org> as well as in the package you downloaded. Basically all you need is to unzip the package and run checksetup.pl and follow any instructions that come up.

Once you have bugzilla successfully installed you need to do the following to get Testopia up and running:

1. Download the testopia tarball from <ftp://ftp.mozilla.org>
2. Copy the testopia tar ball into your bugzilla's root directory.
3. Untar

```
tar xzvf testopia-<release>.tar.gz
```

4. Run tr_install.pl

```
perl tr_install.pl
```

5. Download Dojo (If installing from CVS)

The easiest way to install dojo is from their subversion repository. Instructions are found in the DOJO_INSTALL_HELP file inside the testopia/dojo directory.

The tr_install script sets up the database tables, patches your code, and sets up the Testers group in Bugzilla. Whenever you update Testopia, you should run tr_install and checksetup again.

You should now be able to See the Testopia links in the Bugzilla footer after logging into Bugzilla. Make sure you update the new Testers group to include any users that are going to need access to test cases.

Upgrading From Testopia 1.0.x

To upgrade an existing release of Testopia, you can simply follow the steps outlined above. Running tr_install will upgrade your database and set any file permissions. You can also upgrade to the latest CVS tip using the cvs_update.sh script located in the testopia/scripts directory.

Upgrading from Testrunner (v0.7 and prior) is not supported.

Configuration

Testopia adds a number of additional parameters to the Bugzilla Parameters page. They are

located under the Testopia tab. If you are installing Testopia with a new installation of Bugzilla, you should be sure to set the **urlbase** parameter. Following is a list of required Testopia parameters and their explanations:

allow-test-deletion

If this option is on, users can delete objects including plans and cases. Delete rights are maintained by the plan access control lists.

testopia-allow-group-member-deletes

If this option is on, members of the Testers group will be allowed to delete test objects. This overrides settings for individuals in the Testers group.

testopia-default-plan-testers-regexp

This is the default regular expression for granting access to new test plans.

default-test-case-status

Default status for newly created test cases.

Using Testopia

Why Test?

The following example illustrates why software testing is so important:

Taken from <http://techiecorner.blogspot.com> .

In March 1992, a man living in Newtown near Boston, Massachusetts, received a bill for his as yet unused credit card stating that he owed \$0.00. He ignored it and threw it away.

In April, he received another and threw that one away too.

The following month, the credit card company sent him a very nasty note stating they were going to cancel his card if he didn't send them \$0.00 by return of post. He called them and talked to them; they said it was a computer error and told him they'd take care of it.

The following month, our hero decided that it was about time that he tried out the troublesome credit card figuring that if there were purchases on his account it would put an end to his ridiculous predicament. However, in the first store that he produced his credit card in payment for his purchases, he found that his card had been canceled.

He called the credit card company who apologized for the computer error once again

and said that they would take care of it. The next day he got a bill for \$0.00 stating that payment was now overdue. Assuming that, having spoken to the credit card company only the previous day, the latest bill was yet another mistake, he ignored it, trusting that the company would be as good as their word and sort the problem out. The next month, he got a bill for \$0.00 stating that he had 10 days to pay his account or the company would have to take steps to recover the debt.

Finally giving in, he thought he would play the company at their own game and mailed them a cheque for \$0.00. The computer duly processed his account and returned a statement to the effect that he now owed the credit card company nothing at all.

A week later, the man's bank called him asking him what he was doing writing a cheque for \$0.00. After a lengthy explanation, the bank replied that the \$0.00 cheque had caused their cheque processing software to fail. The bank could now not process ANY cheques from ANY of their customers that day because the cheque for \$0.00 was causing the bank's computer to crash.

The following month, the man received a letter from the credit card company claiming that his cheque had bounced and that he now owed them \$0.00 and unless he sent a cheque by return of post they would be taking steps to recover the debt.

Though never proven to be true, this rather humorous anecdote explains the impact that untested software can have. In the case of both the credit card company and the bank, this seemingly minor issue may have cost both institutions significant sums of money in lost transactions and down time, to say nothing of the frustration of the customer.

Many times software errors cause only minor inconveniences. Other times they can have serious consequences. Think of the life support systems on the space shuttle or the flight control systems on an airliner. Software has become such an integral part of our lives that many times we don't realize how much we depend on it until something goes wrong.

What Types of Testing can Testopia Help You With?

There are two general classifications of software testing: black box and white box.

Black box testing is a strategy in which testing is based solely on the requirements and specifications. Unlike its complement, white box testing, black box testing requires no knowledge of the internal paths, structure, or implementation of the software under test (Copeland, 2004, Sec I).

Black Box Testing

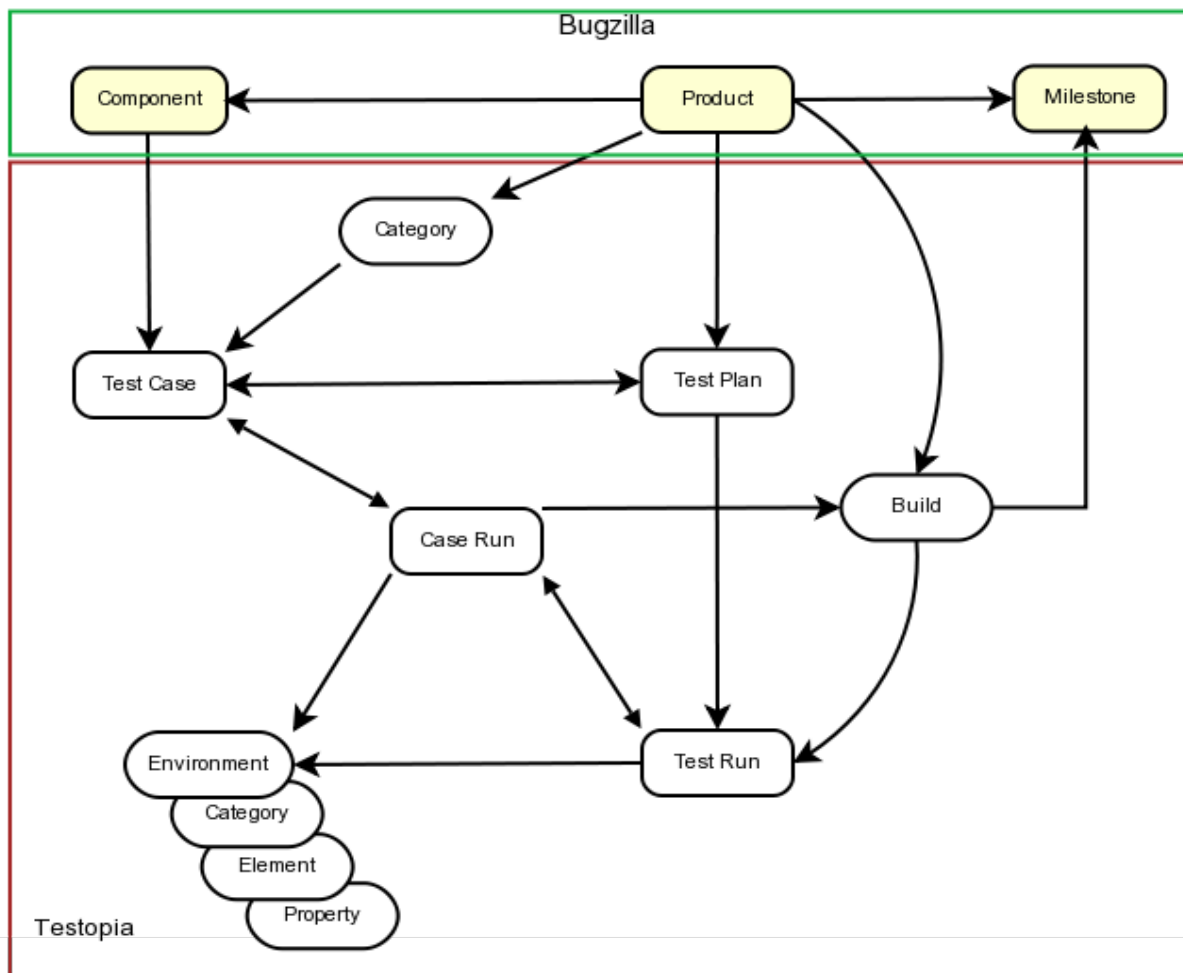
Testopia was designed primarily for this type of testing. Requirements are translated directly into test cases to be applied by the testers to the software being developed. If a requirement is not met, the test case fails; if it is met, it passes.

White box and Automated Testing

Though not specifically designed to handle white box tests (which are often automated), Testopia can still provide a repository for test results. Each test case can have an associated script and test logs can be attached to a test case to show the results of testing.

Getting Started

The Big Picture



Since Testopia is an extension to Bugzilla, it relies on many of the same objects that Bugzilla uses to track bugs. In order to use Testopia you first need to set up products and versions using the Bugzilla product editor. Testopia makes use of Bugzilla's target milestones feature as well so you may want to turn this optional field on in the parameters.

Testopia is comprised of several objects that are interdependent. Together, they make managing the testing procedure possible. Lets look at each of these in turn.

Test Plans

At the top of the Testopia hierarchy are test plans. Before you can do anything else in Testopia, you need a test plan. Test plans are associated with a single product in Bugzilla, though you can have multiple test plans for each product. Your test plan will serve as the storage point for all related test cases and test runs and it will act as the dashboard for your testing. It will also serve to determine who will have access to update test cases.

Test Cases

Test cases are the heart of all testing. Test cases spell out what steps to take while running a test and what outcomes to expect. If a particular run through the steps fails to produce the expected outcome, the test fails. Test cases are semi-independent in Testopia. Each test case can be associated with multiple test plans so care should be taken to ensure that updating a test case does not interfere with testing in another test plan than your own. A list of associated test plans is displayed with each test case though, so this should not be too difficult.

Test cases are divided into categories. You can define as many categories for your product as you like from the test plan page. These should not be confused with components however. Each product in Bugzilla is divided into components and you can apply multiple components to each test case, however each test case can belong to only one category at a time.

Test Runs

Test runs are the focal point of the testing effort. Once you have defined a set of test cases, you are ready to run through those tests in a test run. Each run is associated with a single test plan and can consist of any number of test cases from that plan. Before we can run any test however, we need to have a couple more items first.

Test Run Environments

If test cases are the 'what' of testing, then environments are the 'where'. No test runs in a vacuum. Where you run your test is as important as how you run it. Software often is designed to run on specific hardware under specific conditions. These conditions are captured in the test environment. Environments are applied to test runs directly, but can be applied to test cases indirectly as we will see.

Builds

Software development is usually an iterative affair. Developers write code which is then compiled and included in a system. As bugs and enhancement requests come in, the developer rewrites portions over again to fix or enhance a product. In Testopia, each iteration is called a build. Builds are often associated with milestones of the project. This is reflected in the build's relationship to Bugzilla's target milestones object. Regardless of whether you use target milestones in Bugzilla, you need to define at least one build for your product before you can begin a test run.

Test Case-Runs

A test case-run is the record of how a particular test case fared in a particular run for a given build in a given environment. When you create a test run, records for each test case in that run are created. By default these take on the build and environment of the test run. Once a run has started and a new build comes along, you can add a test case with this new build to the run. This is in addition to any other instances of the test case that might already exist in the run. The same can be done for new environments. This is desirable in situations where most test cases are fairly generic in environmental scope but single test cases might require specific conditions. It also aids in iterative testing as each test case then be updated in a single run rather than having to create new runs to test the failed cases.

Testing In Testopia

Start With a Plan

All testing should start with a test plan. In most cases this will take the form of a document that details the purpose, structure, and methods that will be employed in testing. The IEEE has put together a standard document template for test plans (IEEE Standard 829-1998 for Software Test Documentation). Whether you decide to use a formal template such as this or not, having a rough outline of what your purpose for testing is will help those who test this in the future know what you were looking for in your testing.

To create a new test plan in Testopia:

1. Click the '[New Plan](#)' link in the Bugzilla footer
2. Enter a name for this test plan.
3. Select a product from the product drop-down list
4. Select a type for this plan (descriptions of types below)
5. Select a product version. This will be the default version for new runs.
6. Type or paste your plan document into the Plan Document Editor
7. Click the **Add** button

Once you have done this, you will be taken to your newly created test plan.

Viewing A Test Plan

At the top of your new plan, you will see an overview section detailing much of the information you entered on the new plan form. In addition to this however, you will see such things as the plan author, when it was created, and what version of the document you are viewing.

Below the overview section is the place for viewing categories and builds associated with this plan's product. You also have an area where you can add tags and view existing tags on this plan. We will discuss tags at greater length later.

From here you may wish to create some categories for your test cases and add some builds before you begin creating test cases.

Attaching Files

Once you have created your test plan, you can upload attachments to it. To create an attachment, simply click the browse button and locate the file you wish to attach, type a description in the Description field and click the attach button.

Attachments are treated very much the same as they are in Bugzilla. The size limit will be determined by the max-attachment-size parameter the same as it is for Bugzilla

To edit an attachment click the edit link. This takes you to a page where you can set the description, filename, and mime-type for the attachment as well as view it if it is a viewable type or download it otherwise.

Editing Plan Fields

With the possible exception of product, you can edit any of the values that you supplied when you created your test plan by scrolling down to the Attributes section or by clicking the '[Edit Plan Attributes](#)' link in the overview section. You are able to edit the product field as long as there are no test runs associated with this plan yet. Once you have created a test run you will not be able to update the product attribute unless you delete the run first.

Viewing Plan History

Any changes made to plan values are captured and a history is kept so that as test circumstances change over the course of a development cycle, you can know what applied for a given set of tests in time. This also serves to thwart unintentional changes to your test plan since each change captures when it happened and who performed it. To view this history click the **History** button.

Changes to the plan document are handled a little differently. A full version is kept for each change and it is possible to view the differences between versions by selecting which versions you want to compare and clicking the **Compare** button on the plan history page.

Adding Categories and Builds

Categories are used to classify your test cases. Each product has a default category and it is up to you want to use categories to divide up your test cases. Since you can also apply product components to a test case, you may find that this suffices. Before you can create a test run however, you have to specify at least one build. To add a build or category you can click the [add](#) link found below each of these fields from the test plan page. The fields are fairly self explanatory. In the case of builds however, you have the option of hiding the build by unchecking the **Active** box.. This will hide it from the build list used when creating a run.

To edit categories and builds, click the links above their respective fields on the test plan. From here you can edit individual builds and categories or remove them.

Creating Test Cases

Once you have a plan to store them in, you can start entering your test cases. Click the [Create a New Test Case](#) link on the plan page or click the [New Case](#) link in the footer to begin.

Shortcut – Creating a Test Case

1. Click [Create a New Test Case](#) on the plan page or [New Case](#) from the footer
2. Enter a short description of your test case in the **Summary** field
3. Select a **Category**
4. Add a default tester or select a component to assign the test case to the Bugzilla QA Contact for that component.
5. List the steps for testing in the **Action** field
6. Provide the expected outcomes in the **Expected Results** field
7. Click the **Add** button.

Because it is possible to have a single test case linked to multiple test plans, you can select which plans you wish to link your new case to. For now, we only have one so we will only see a single check box.

First you need to write a short summary of your test case in the summary field. Notice the options for the other fields here as well. Descriptions of each can be found at the end of this document. The only required field here is the summary, however you may want to apply a default tester and a requirement at this time. Selecting a component at the right will automatically set the default tester based on the Bugzilla QA contact of that component.

Though not required at this point, you should supply your list of testing steps in the **Action** field and the list of expected outcomes in the **Expected Results** field. To do this, click the [Edit Document](#) link under the appropriate field. This will display a WYSIWYG editing area where you can type or paste your values. For tips on writing good test cases refer to D.L Runnels' [How to Write Better Test Cases](#)

From here you are taken to your test case. You will notice a link at the top giving you the option of creating another test case for this plan at this time.

Adding and Removing Components and Tags

Once again, you are offered an overview detailing the information for this test case at a glance. Below that you have the option of adding additional components to this test case. Because some testing is cross product, you have the option to add components from any product in Bugzilla to your test case. You can also remove a component by selecting it from the list and clicking the **Remove** button.

As with plans you also can apply tags to your test cases. This will be described later.

Viewing Test Case-Run Results

Below this section will display a history of this test case in all runs and their status as to whether they passed or failed in those runs. You also will see a graph representing the percentage of pass/fail across all runs.

Attaching Files

Just like test plans, you can attach files to your test cases. This works the same as for test plans.

Attaching Bugs

Unlike test plans, you have the option of attaching Bugzilla bugs to your test cases. Each test case can have multiple bugs attached. To attach a bug, enter the bug number in the field and click the **Commit** button. A table of attached bugs is displayed.

Editing Case Fields

As with test plans, you can update any of the values you entered while creating the plan. A history of these changes is also kept which you can view by clicking the **History** button near the top of the test case.

Test Case Dependencies

Test cases have the concept of dependencies. Often when testing a set of test cases, the order in which you test is determined by what tests came before. It is also often the case that if a given test case fails, it prevents other test cases from being run successfully at all. You can represent those relationships here using the **blocks** and **depends on** fields. If this test potentially blocks the execution of another test case, you would enter that other test case's id in the **blocks** field. If this test case requires some other test case be run first you can enter that other test case's id in the **depends on** field. If a test case that blocks another test case fails, and both of them are in a single run, the blocked test case will automatically receive a status of **BLOCKED**.

Creating Environments

As mentioned above, environments are the *where* in testing. In software testing in particular, this might include such things as which operating system and what hardware platform a test was conducted on.. Environments can be as broad or narrow as you define them. The most basic environment consists of an OS and platform chosen from Bugzilla's lists of these objects. However it can be much more complicated such as a suite of applications and other products. It could be a browser or other client package.

Creating an environment in Testopia requires two steps. The first involves defining a set of variables to be used in your environment. The second is to create the environment from the set of possible elements.

Environment Administration

When you first install Testopia, you must first define the set of environment variables that will be used to construct your environments. To access these, click the [Environment Variables](#) link in the footer. The environment variables are arranged in a hierarchy of objects that is represented as a tree. There are four major levels: Category, Element, Property, and Property Values.

Categories

Environment Categories are similar to test case categories in that they provide a sorting mechanism for your environment elements. Each category is associated with a single product or in the special bucket labeled *–ALL–* . The *–All–* denotes all products, meaning it holds categories of elements that are not specific to any product.

When you first install Testopia you will see that the *–ALL–* bucket contains two Categories, OS and Platform. Expanding these you will see that there are elements representing each of the OS and Platform values defined in Bugzilla. This list is generated at the time you first install Testopia and is maintained separately from the Bugzilla lists thereafter.

To create a category, right click on the product or *–ALL–* and choose *Add Category*. You can then click on the newly create category in the tree which will pop up a form that allows you to edit the category name or change the product it is associated with.

Elements

Once you have a category defined for your product or the *–ALL–* bucket, you can add elements to that category. Elements are the crux of what makes up your environment. To create an element, right click on the category you wish to add it to and choose *create element*.

This will create an element labeled “New Element” which you can edit by clicking on it in the tree, or right clicking and choosing *Edit*.

Elements can be nested inside other elements. To create a sub element, right click on the element and choose *Add Element*. You can edit this child element in the same manner as its parent. You

can create as many levels of elements as you need to represent the complexity of your environment.

Properties

Properties describe your element. You can add properties to your elements by right clicking the element and choosing *Add Property*. You can add as many properties to your elements as you need. Properties cannot be nested however.

To edit your property, click on it in the tree or right click on it and choose *Edit*.

Property Values

Once you have defined a property for your element, you will need to provide a list of values from which to select for your environment. Right click on your property and choose *Add Value* to create a value for your property. You can edit property values by right clicking and choosing *Edit*, or by clicking on one of the values under your property.

You can change the name or reorder the list of values from the form provided. You must hit *Save Changes* in order for your changes to be committed.

Creating Your Environment

Once you have set up the elements that will be used in your environment, you can now create environments with those elements.

Click the [New Environment](#) link in the footer.

You are prompted to name your environment and choose a product for it. The product is used only for classification. It does not limit your choices of which elements can be placed in your environment.

Clicking **Create** will take you to the environment editor. Here you will see two trees, one representing your new environment and the other containing the list of variables from which to choose. Your environment will consist of the elements you defined earlier. To add an element, find it in the list and simply drag it onto your environment tree. The order does not matter. You can grab child elements at any level, but dragging an element with children will bring the children as well.

Once you have selected the elements for your environment, you can now select which of the property values apply to your environment. Expand the element and property and simply click the value you wish to use. It will have a star placed next to it to show your selection.

To remove an element, right click on it and choose *Remove*.

All changes to your tree are saved immediately.

Creating A Test Run

Shortcut – Creating a Test Run

1. Click the [Create a New Test Run](#) link on your test plan or the [New Run](#) link in the footer.
2. Select which test cases to include. Use the filter to narrow your scope and the paging links to view more or fewer test cases.
3. Enter a summary for this test run.
4. Select a build from the **Build** list or type the name for a new one.
5. Select an environment from the **Environment** list.
6. Click **Add**.

Once you have an environment to test in and some test cases to test, you are now ready to begin testing. You do this by creating a test run. The easiest way to do this is from within your test plan. Click the [Create a New Test Run](#) link in the Test Runs section of your plan. You can also create a run by clicking the [New Run](#) link in the footer and entering your plan number.

You will be presented with a list of CONFIRMED test cases from your test plan. You can select only the ones you want included or all of them at once using the select all link. If you want to limit the list to only include test cases that meet certain criteria, you can use the filter to narrow down the list. If you have a lot of test cases, you can use the paging features in the table of test cases to view more than the default 25 at a time. This is important as the only test cases that will be included are those that are visible on the screen and have a check in the box next to the case ID. If you do not know which test cases to apply, or do not wish to do so at this time, you can always add test cases later.

Next you need to provide a summary and select a build. If you forgot to create the build before, you can simply type the name of a new build in the **New** box and it will be added to the product. You can then edit the build later to add a milestone. Finally you need to select an environment. You do this by typing the name of your environment in the **Environment** box. As you type, a list of environments from your product that match will appear. You can also click the drop down arrow to view a list of all environments in your product.

Once you have completed the form, click the **Add** button to be taken to your new run.

Viewing Your Test Run

As with the other objects, you will notice an overview section at the top of your run. This contains similar information as found in the test case and test plan with one notable exception: the progress bar. This will show you a percentage of completed test cases with the colors representing statuses.

Below this section you will see the Test Case Run Logs. This table represents the test cases you will be testing each row represents a single case-run. You will notice the expander arrows next to

each row in the table. Click on this to expand the case-run form. Take note of the **Filter** and **Report** expanders as well. You can use the filter much like you did when choosing the test cases to include in your run, except this provides more options for narrowing the list of case runs visible at a time.

Adding Cases

There are two ways to add test cases to your test run. The first deals with adding test cases that are not already part of this run but are in the test plan the run belongs to. You do this by clicking the **Add Cases** button below the case-run table. This will take you back to the list of test cases you saw when creating your test run excluding any test cases in your plan that are already in this run.

The second method deals with adding a second (or third or more) instance of a test case with a different build and/or environment combination. To do this, enter the test case ID's separated by commas in the box above the table but below the filter. You then select a build and an environment from the available drop-down lists and click the New Case-Run button. You also have the option of adding them from a list, similar to above, except that test cases that are currently part of the run will be included. Do this by clicking the Add several at once link.

Editing Test Run Fields

As with test cases and test plans, you can update any of the values you applied when you created the test run in the form at the bottom of the page. Note that changing the build and environment will not affect the case-runs already in the run but it will apply to any test cases added after the change. Changes to these fields are tracked and a history is kept much the same way as with test plans and test cases. Clicking the **History** button will display this history.

Running Your Tests

As a tester, you will likely spend most of your time running tests and recording the results from the test run page. When you first open a run the test cases are represented in a table as case-runs. Clicking the expander arrow next to each test case will expand the test case-run form. Your run may included only those test cases that are required for a specific testing priority, or for a given tester. Or, your test run may include a larger set of test cases that represents an entire test cycle. On a given day, you may only want to test the highest priority test cases and skip lower priorities for later in the process. Or you may just want to run regression tests on the set of previously failed test cases. You can do this using the filters.

Filtering Your Test Cases in a Run

Expand the **Filter** by clicking the expander triangle next to the filter. You will notice a number of options for filtering your test cases including:

- Status

- Category
- Build
- Environments
- Priorities
- Components
- Assignee
- Tags
- Case summary.


Testopia remember your filters so that the next time you view this run it will apply the last filter you used. To clear the filter and return to the full list click the **Clear** button.


Sorting Test Cases

You can sort the list of test cases by clicking on any of the table column headings. You can also sort on the summary by clicking any of the **Summary** field headers. If you wish to create a custom sort order for your test cases, type a number to index on in the **Index** fields and click **Change**. Sorting on this field will then display your cases in ascending order of the indexes you supplied.

Currently Testopia only supports ascending sorts. Support for descending sorts are planned for future releases.

Passing and Failing Test Cases

Once you are ready to begin testing, expand the first test case in your list and read the **Action** and **Expected Results** for this case. You can then perform the test. If the expected outcome is achieved you can pass the test by clicking the green check or **PASSED** button: .

If the results were not expected or an error occurred, you can fail the test by clicking the red X or **FAILED** button: .

Explanations of the other statuses appear at the end of this document.

Adding Notes

Updating the status will add a line to the notes field with the time of the change and who made it. You can add additional notes by typing them in the **Add Note** field and clicking **Append Note**.

Attaching Bugs

If a test case has failed, or for any other reason, you may want to attach a bug to this test. You can attach existing bugs or create new ones. To attach an existing bug, type the bug number in the box and click **Attach Bug**. The bug will be displayed in the **Bugs Detected** field. If you wish

to log a new bug, click the **New** button. You will be taken to the enter bug page with information about the test case already provided in the form. You can now provide any additional details and submit the bug. This is done in a new window so that you do not lose your place in testing. To return to your run, close the new window.

Reassigning Tests

The assignee field is used to help testers track their own test cases. If you wish to change the assignee of a particular test case, you can do so by entering their Bugzilla user name in the assignee field and clicking **Assign**. Updating test cases is not limited by assignee. Anyone with rights to edit the case-run can pass or fail the test. If a test case is closed (PASSED, FAILED, or BLOCKED), the name of the person who performed the action is captured in the **Tested By** field.

Changing Build or Environment on a Test Case

Testopia is designed to be flexible. Testing organizations vary greatly on modes and methodologies. For this purpose, test runs were designed to allow the greatest flexibility possible as to how tests are run. Some groups will create a single test run for each build and environment combination they encounter. Others will want a monolithic test run that represents an entire product release. In this case you can update the build and environments on individual test cases in your run. Each time you do, the appropriate case-run record is returned, if one exists, or is created if it does not. A note is appended with the time of the change as well as who made the change. If a new record is created, the status is set to IDLE and you are ready to test with your new build or environment selection. For any given run, the case-runs displayed are the last ones that were updated. In essence however, the case-run table represents a two dimensional view of a three dimensional object. Changing the build or environment bring the corresponding record to the forefront. This should be kept in mind when running reports on case runs as what you see in a run is only the surface of what data may be included.

Classic View

The test run page uses Javascript and AJAX technology to provide a more seamless user experience while running your tests. If you have trouble with the forms or would like a simpler view of your case runs, you may use the **classic interface**. This provides a view of a single test case-run at a time. From here you can update any of the fields mentioned above and also attach files to the test case.

Deleting Case-Runs

Sometimes you might mistakenly add a test case to a run that does not belong or you may select an invalid build and environment combination. In these cases it is easiest to remove them by deleting the case-run all together. You must have the proper permissions to do so and the administrator needs to allow this in the parameters. To delete a single case run, click the Delete link on the standard form, or click the **Delete** button on the classic form. You will be prompted to

confirm the action before the record is removed.

Updating Multiple Cases at Once

You can update a group of case-runs at once by clicking the [Update Multiple](#) link below the table of cases. From here you can change the status, attach bugs, update build or environment attributes, and even delete a whole list at once.

Wrapping Up

Once all tests in a run are complete, you should set the status on the run to STOPPED. This will prevent further updates to case-runs in the run.

Getting Around Testopia

Navigating Testopia is done using the links in the Bugzilla footer. You can use the QuickSearch box to quickly jump to any case, run, plan or environment. To do so simply type the object's ID number, or part of the name or summary, prefixed by a type identifier. Following is the list of identifiers:

| <i>Object</i> | <i>Prefix</i> |
|----------------------|--|
| TEST CASE | (Optional for cases) <ul style="list-style-type: none">• case• TC• c |
| TEST PLAN | <ul style="list-style-type: none">• plan• TP• p |
| TEST RUN | <ul style="list-style-type: none">• run• TR• r |
| TEST RUN ENVIRONMENT | <ul style="list-style-type: none">• env• TE• e |
| TEST CASE-RUN | <ul style="list-style-type: none">• caserun• TCR |

| <i>Object</i> | <i>Prefix</i> |
|---------------|---|
| | <ul style="list-style-type: none"> • cr |
| TAGS | <ul style="list-style-type: none"> • tag |

So for example, to find test run 45 you could type *r 45* or *tr 45* or *run 45*.

To find environments with 'Linux' in the name you could type *e:linux* or *TE Linux*.

Searching

Bugzilla has very powerful searching capabilities. Testopia has applied the Bugzilla search engine to searching for test cases, plans, runs, environments, and case-runs. Each of these has its own set of search parameters that can be used to narrow the scope of a search. To perform a search, click the [Search](#) link in the footer and then click the tab representing the object you wish to search from.. From here you select the parameters you wish to query on and click the **Submit** button. This will take you to the corresponding list page for that object.

Sorting Search Results

Search results are paged and sortable. To sort, click the column header of the corresponding field you wish to sort on. This will return the list in ascending order. Descending sorts are not currently supported but are planned.

Paging

By default, Testopia will display 25 records at a time. You can use the links at the bottom of the list to navigate from one page of search results to another or use the **Jump To Page** button to jump directly to a page of results. You change the number of records you view at a time by selecting an option from the page size drop down. Lastly, if there are not too many records, you can opt to view all of the records at once using the [View All](#) link.

Batch Processing

From the search results page of cases, runs, plans, and case-runs, you can update a batch of objects at once. Do this by selecting which objects to update by checking its box and then updating the fields at the bottom of the form. For example, with test cases, you can apply changes to virtually any fields as well as add these cases to test runs or link them to a set of plans. You can also export test cases as CSV or XML. More on this later.

Saved Searches

Like Bugzilla, Testopia offers the user the opportunity to save a search for later retrieval. You do this by typing a name in the *save search as* field and clicking the **Save Search** button. Saved searches appear below the other Testopia links in the footer. You can remove a saved search by running it and clicking the Delete saved query link at the bottom of the search results page.

Reporting

As with searching, Testopia has modified the Bugzilla reporting engine to provide basic tabular reports based on a set of search criteria. These reports can be run on test cases, runs, plans, and case-runs. To run a report, click the Reports link in the footer and click the tab corresponding to the object you wish to report on. Select the fields for the horizontal and vertical axes of the report and then select your search criteria. The default report is in the form of tabular data. You can then switch between bar and line graph modes or export the data as a CSV (comma separated values) file which can be opened in a spreadsheet.

In addition to these generic reports, certain objects have other reports associated with them. These are accessed from the object pages directly.

Plan Reports

Build Coverage Report

This Report is designed to show builds that are not being covered in testing. It breaks down each run, displaying the status of the latest case-run for a given build. If there are builds that are not being covered, no status will display for any runs for that build. This report is not very practical for plans with very large numbers of test cases.

Top Bugs

This report displays the bugs found in test cases within the test plan ordered by the bugs attached to the most test cases. In other words, which bugs are failing the most test cases.

Bugs Found In This Plan

This generates a bug list of all the bugs attached to any case in this plan.

Printable Percentage Report

This report displays the pass/fail percentages of the plan by run, build, and environment.

Test Case Reports

Estimated vs Actual Time

Test cases can have a time estimate associated with them. This can then be used in conjunction with the case-runs for that case to determine the actual running time. To do this, test case-runs must be placed in the RUNNING state when a test is started. This then starts the stopwatch so to speak for that case in that run. When a case is passed or failed the difference of the two times is calculated. The average of all case-runs in all runs for this case is then used to calculate this report.

Historic Status Breakdown

This pie chart represents the percentage of all case-runs for this case divided by status.

Cloning

Often it is desirable to copy test information for use in a new development cycle. Testopia allows you to clone test cases, test run, and whole test plans, allowing you more efficiently move from one product release to another. Cloning is performed by clicking the **Clone** button on the object you wish to clone.

Cloning Test Plans

To clone a test plan:

1. Navigate to the plan you wish to clone and click the **Clone** button.
2. Enter a name for the new plan or accept the default.
3. Choose a product and product version to clone to.
4. Select the clone options
5. Click **Clone**.

When cloning a test plan, you have the option to copy attachments, the plan document, any tags, and the plans access control list. You have the option of either copying the test cases, or creating links to the existing test cases. Copying the test cases will produce a new set of test cases with new Ids whereas linking will allow you to make modifications to your test cases and have then reflected everywhere.

If you choose to copy the test cases, you are also given the option to apply yourself as the author of the new cases.

Cloning Test Cases

To clone a test case:

1. Navigate to the test case you wish to clone and click the **Clone** button.
2. Select a clone method.
3. If copying, check the box to copy within the plans already associated
 1. Enter the plan ids of any additional test plans you wish to copy to
 2. Select clone options
4. If linking, enter the ids of any plans you wish to link this case to.

As with cloning plans, you have the option of making yourself the author of the copy. You can choose to copy the attachments, tags, components, and document.

Cloning a Run

To clone a test run:

1. Navigate to the test run you wish to clone and click the **Clone** button.
2. Enter a summary for the copy.
3. Choose a plan within the product to clone to.
4. Choose a product version and build.
5. Select clone options.
6. Click the **Clone** button.

In addition to the options to copy the tags and making yourself the manager of the cloned run, you have the option of copying over the test cases. You can limit the clone to only include test cases of a certain status. For instance you may want to only include cases that failed in the cloned run.

Deleting

It is possible to delete test cases, plans, runs and case-runs. To delete, you must have the allow-test-deletion parameter turned on and have the proper rights. Bugzilla admins (members of the admin group) can delete regardless of the parameter settings and have rights to delete anything in the Testopia system.

To delete an object, click the Delete button on that objects page. You will be taken to a confirmation screen detailing what other objects will be deleted, if any, as a result of deleting this object. To confirm, click the **Delete This Test** button. Deleting in Testopia is irreversible. If you wish to simply hide an object instead you can do so using the methods outlined below:

Test Cases

To hide a test case from search results, place it in the DISABLED status.

Test Plans

To hide test plans from search results, you can archive them by clicking the **Archive** button on the plan page.

Test Runs

To hide test runs, place them in the STOPPED state.

Importing and Exporting

Test cases can be exported as either XML or in Comma Separated Value formats. To export a test cases, click the appropriate format button under the **Export** section found at the bottom of the test plan page, the test case page, or the case search results page.

Exporting in CSV allows you to open you test cases in a spreadsheet where you can manipulate the values and generate custom reports.

Exporting as XML allows you to import your test cases into other Testopia installations or into custom database applications.

Importing Test Cases

You can import test cases from other Testopia installations or from other test case management systems using the tr_importxml.pl script. In order to be eligible for import, your exported test cases must first comply with the Testopia document type definition found in the testopia.dtd file. This can be done by using XML stylesheets to transform the XML to match the DTD. This is outside the scope of this manual, but information on this subject is readily available via the web.

To import test cases, run the tr_importxml.pl script with the path to your test case XML file. More details can be found on the project wiki at

<http://wiki.mozilla.org/Testopia:Documentation:XMLImport>

Testopia Security

Like Bugzilla, Testopia provides several methods for restricting access to test data. Because test data is often more forward looking than bug reports, Testopia requires that users log in to see any test objects. There are several layers of security that a user must pass through to view those objects however. These are determined in part by Bugzilla group membership, but mostly by the test plan access control list.

The Testers Group

When you first install Testopia, it will create a Bugzilla group called 'Testers'. Members of this group have access to view and update all test plans and their associated objects such as cases and runs. Membership in this group is required in order to create new test plans, clone test plans, and

administer environments. If the **testopia-allow-group-member-deletes** parameter is on, members of this group will also have rights to delete any object in Testopia. Membership in this group is checked first and supersedes the access control lists for individual plans.

Test Plan Access Control Lists

In addition to the Testers group, each test plan maintains it's own access control list which can be used to allow or deny access to test plans based on email domain or explicit inclusion. Each test plan has its own access list. For a user that is not in the Testers group to access a test plan or any associated cases, runs, or case runs, he or she must be included on the list either by matching a regular expression, or explicit inclusion. To edit the access control list for a plan, navigate to the test plan and click the [Edit Access Controls](#) link in the over view section.

User Regular Expression

Users with login names (email addresses) matching a supplied regular expression can be given rights to a particular test plan. The regular expression should be crafted with care to prevent unintentional access to the test plan by outsiders. For example, to grant access to your test plan by all users at acme.com you would supply the following regular expression:

```
^.*@acme\.com$
```

To provide access to all users at acme.com and foo.org you would use:

```
^.*@(acme\.com|foo\.org)$
```

To provide public access (all users) you would use:

```
.*
```

An empty regular expression does not match anything meaning leaving this field blank will mean the test plan will rely solely on explicit membership.

Once you have supplied the regular expression, you must select the access level.

Explicit Inclusion

If you do not wish to grant access to a whole group at once, you can add individual users by entering their Bugzilla login in the field provided and clicking the **Add User** button. This allows the most fine grained control as to who can do what within your test plan. However, if you add a user that matches the regular expression they will have the greater of the two rights.

Access Rights

Users on the test plan access control lists can be granted rights to read, write, delete, and admin test plans and their associated objects.

Read

Allows viewing rights to the plan and all test cases, test runs, and test case-runs associated with it. Test cases linked to more than one plan will be visible to users in both plans.

Write

Implies Read. Allows rights to modify the plan and associated cases, runs, and case-runs. Test cases linked to more than one plan will not be writable unless the user has write rights in all plans.

Delete

Implies Read and Write. Allows rights to delete the plan and associated cases, runs, and case-runs. Test cases linked to more than one plan will not be deletable unless the user has delete rights in all plans.

Admin

Implies Read, Write, and Delete. Allows rights to modify the plan's access controls.

Tags

Testopia utilizes a tagging mechanism to help organize test objects. Unlike the traditional method of placing things into virtual folders, tags allow a many-to-many relationship by allowing multiple objects to have the same tag while allowing each object to have multiple tags.

Tags are a novel approach to categorizing objects. They have become very popular with numerous web popular web tools and have met with much success. Tags are similar to keywords in Bugzilla but do not require an administrator to create. The act of tagging an object creates the tag which can then be used anywhere.

The idea behind tags is that each user can categorize each item to his or her own liking without destroying other users' categorizations. Test cases, plans, and runs can all have tags associated with them. Though tags are not directly associated with a product, Testopia uses a smart typing approach to recommend tags based on relationships to other objects within the same product.

Tags have a knowledge of who added them to a particular object as well, allowing users to manage the tags that they have created.

Adding Tags to an Object

To add a tag to a test case, run, or plan, begin typing in the **Tag** field. If what you type matches any other tags that others have applied within the product associated with this object, it will appear up in the drop down list. You can then either select a tag that matches or type your own new tag. Clicking the Add button will then attach your tag to the object.

Viewing Tags

To see a list of your tags click the [Tags](#) link in the header above the tags section in any object. From this screen you have the option to look up tags based on product or entered by another user.

Using Testopia With Automated Test Scripts

Testopia provides a web service XML RPC that utilizes the SOAP protocol to interact with Testopia objects via an automated script. Documentation for this feature is available on the project wiki at <http://wiki.mozilla.org/Testopia:Documentation:XMLRPC>

Field Descriptions

Test Plans

Name

Short descriptor for a test plan. Does not need to be unique.

Product

Bugzilla product that this plan is associated with.

Product Version

The Bugzilla product version. Used as the default for new runs.

Type

The testing type for this test plan. Possible values include system, integration, unit, functional and acceptance. This list is modifiable by the Bugzilla admins from the Testopai Admin link.

Archive

If this bit is set to true, the plan will be archived and not display in searches by default. To archive a plan click the **Archive** button. To unarchive click **Unarchive**.

Plan Document

This is document that spells out the type of testing and testing methods used for this plan.

Tags

See tags description above.

Test Cases

Summary

A short description of the test case.

Default Tester

The person who will be assigned to this test case when a new run is created in which this case is included.

Alias

A unique string that can be used to identify this test case. This can be used in place of the test case ID but must be globally unique to the database.

Requirement

The requirement number or URL to a document containing the requirement this test case is designed to test.

Status

Test case status determines whether this test case can be included in new test runs.

- **PROPOSED** – This test case is a work in progress and has not been reviewed for accuracy. It is therefore not ready to be included in test runs.
- **CONFIRMED** – Test case has passed review and is ready to be included in future test runs. Only test cases with this status can be included in new test runs.
- **DISABLED** – This test case is no longer applicable to current testing.

Priority

This denotes the level of testing. Higher priority test cases should be run first and more often than lower priorities.

Category

The product category that this test case belongs to. Each product has a default category. Additional categories can be added which can be used to further classify your test cases. Because this is a product attribute, this list will be the same for all plans in a given product.

Estimated Time

The estimated time in HH:MM:SS format that this test case should take to complete.

Add To Runs

Entering a run ID here will include this test case in a test run if it is not already included. You can also enter a comma separated list of run numbers to add to multiple runs at once.

Automatic

Test cases can be either Automatic or Manual. Automatic test cases are run by a script while manual test cases are performed by a tester.

Script

If this is an automatic test case, you can enter the name of the script that runs this test case. TESTOPIA DOES NOT RUN YOUR SCRIPT FOR YOU. You must run your script and capture the results to send to Testopia. You can use the XMLRPC interface to have your script update the test case-run results automatically.

Arguments

If there are specific arguments sent to the script that apply to this test case, they can be stored here. TESTOPIA DOES NOT RUN YOUR SCRIPTS FOR YOU. This field is only used to store the information. The XMLRPC can be used by your script to capture this from the test case.

Attach Bugs

You can attach bugs to your test cases. This field accepts a comma separated list of bug numbers.

Depends on

If this test case requires other test cases be run before this one, their case numbers should appear here. Enter a list of comma separated case numbers to add dependencies.

Blocks

If this test case prevents others from being run their ID numbers will appear here. This field accepts a comma separated list of case numbers.

Set Up

This field details the steps that are necessary to prepare to run a test

Break Down

This field details the steps required to reset the test case to a known state in preparation for the next run.

Action

This field lists the steps of testing.

Expected Results

This details the expected outcomes of a test.

Component

You can associate Bugzilla components to your test cases. Because testing might be across multiple components and products, you can add multiple components from any product. Selecting the product from the drop down list displays the components available. Click Add to associate the selected component with the test case.

Tags

See above for a description.

Test Runs

Product Version

This is the Bugzilla product version that this test run is testing.

Plan Version

This is the version of the plan document that this test run is using.

Manager

The user in charge of this test run

Build

The default product build. This is applied to test cases added to the run.

Status

Determines if this test run is active.

- **RUNNING** – This test run is still active. Test cases in this run can be updated.
- **STOPPED** – Test cases in a stopped run cannot be modified.

Environment

This is the environment in use for this test run. Test cases added to this run will receive this environment. Environments must be created using the Environment editor before a test run can be created.

Summary

A short description of this test run

Notes

This field is a place to make notes and observations about this run.

Test Case-Runs

Status





The status of a case-run determined by whether it passed or failed or is in some other state.



- **IDLE**: This is the default status. Case has not yet been examined.



- **PASSED**: This test case met the requirement or ran as expected.

-  - **FAILED:** This test case did not run as expected or produced an unhandled exception.
-  - **RUNNING:** This test case is currently being examined.
-  - **PAUSED:** This status is used to denote a problem with the test case itself that prevents the test from being completed.
-  - **BLOCKED:** This test case has a dependency that failed.

Index

A user defined sort index. This can be used to place your test cases in a specific order.

Add Notes

This is a place to append notes to the case-run.

Notes

This field combines the notes from all case-runs records for this case in this run as well as status change history.

Update bug status

With this box checked, bugs that are attached to this test case in the RESOLVED FIXED state will be automatically placed in the REOPENED state if the test case fails or into the VERIFIED FIXED state if it passes.

Attach Bugs

This field allows you to attach bugs to this test case. Enter a comma separated list of bug numbers to attach more than one bug at a time.

Assignee

This is the person assigned to run this test case.

Tested By

This is the person that actually ran the test. This is updated when a test case is placed into the PASSED, FAILED, or BLOCKED status.

Close Date

The time this test case was closed (PASSED, FAILED, or BLOCKED)

Build

The product build used for this test case-run.

Environment

The product environment used for this test case run.

Glossary

| <i>Term</i> | <i>Objects</i> | <i>Description</i> |
|--------------------|-----------------------|---|
| Action | Test Case | The list of steps that a test case must complete. |
| Alias | Test Case | A globally unique string that identifies a test case in conjunction with the test case ID. |
| Archive | Test Plan | Test plans may be archived and hidden from regular searches. |
| Arguments | Test Case | A list of parameters to send to an automatic test script . |
| Assignee | | The person responsible for applying a status to a test case-run |
| Blocks | | A test case that blocks another test case. |
| BLOCKED | | A status of a test case-run indicating the prerequisite test case failed. |
| Build | | In software testing, a string denoting the compiled results of a period of development. |
| Category | | A property of a product that is used to classify test cases. |
| Clone | | An exact replica of data between two objects. In Testopia you can clone plans, runs, and cases. |
| Component | | A Bugzilla component. An attribute of a product. |
| CONFIRMED | | A status of a case. Confirmed test cases have been approved for use in test runs. |
| Default Tester | | The default person responsible for applying a status to the test case-run for a given test case. |
| Dependency | | Test cases can be dependent on other test cases. There are two types of relationships, depends on and blocks. A test case that is blocked by another should not be examined before the prerequisite test case as success is determined in part by the outcome of the predecessor. |
| Depends on | | Sets up a dependency between test cases. Depends on lists the test cases that a particular test case requires to be completed before this case. |
| DISABLED | | A status of a test case denoting it is no longer used for active testing. Similar to archival for a plan. |
| Expected Results | | The expected results upon completing the action of a test case. |

| | | |
|---------------|--|--|
| Environment | | A list of the surrounding conditions that a test run is performed in. |
| FAILED | | A status of a test case-run. Denotes the test case failed in the given run. |
| IDLE | | A status of a test case-run. Denotes the test case has not been examined in the given run. |
| Manager | | The person in charge of a given run. |
| Milestone | | A Bugzilla object. A property of a product that implies when a given bug or feature will be fixed or included. Testopia builds are associated with milestones. |
| PASSED | | A status of a test case-run. Denotes the test case has met the conditions of success detailed in the Expected Results of the test case in the given run. |
| PAUSED | | A status of a test case-run. Denotes the test case has been under examination in the given run and is on hold. Used primarily for performance tests that may span long time periods. |
| Plan Document | | The information of a test plan detailing what the test plan will cover for what by when. Depending on the level of scrutiny required it can be very verbose. |
| Plan Version | | The version of the plan document used for a particular run. |
| Priority | | The Bugzilla priority. Test cases can be assigned a priority similar to bugs. |
| PROPOSED | | A status of a test case that denotes it has not yet been approved for use in test runs. |
| Requirement | | A field of a test case provided to capture information about a requirement. Typically an ID of a requirement in a separate requirement tracking system. |
| RUNNING | | A status of a test case-run. Denotes the test case is in the process of being examined in the given run. |
| Running | | A status of a test run. Running test runs can have case-runs updated and implies that there is further testing to be done in the run. |
| Script | | A path to an external automated test script for a given test case. Testopia does not run this script, the field is only provided as a way of informing the user where to find it. |
| Stopped | | A status of a test run. Stopped test runs can not have case- |

| | | |
|---------------|--|--|
| | | runs updated. This status denotes the run is complete. |
| Tag | | A user defined string used to classify test plans, cases, and runs. |
| Test Case | | A list of conditions and expected results for success for a particular feature or object under scrutiny. Test cases are associated with one or more test plans and with zero or more test runs. |
| Test Case-run | | The union of a test case and a test run. Each time a test case is included in a new test run, an entry is made for it in the test case-runs table. This captures whether the test case passed or failed in the given run. Each case-run should be associated with only one build for a given status. |
| Test Plan | | The defining object in Testopia. Organizes the other objects in Testopia. |
| Test Run | | The instance of performance in Testopia. Each run is associated with a single plan and environment. It contains a list of test cases to be examined and stores the results in the case-runs table. |
| Tested By | | The person who examined and applied a status to a given case-run |
| Type | | The plan type. Plan types might include System, Unit, Integration etc. Each plan can be of only one type. |

Getting Help

There are a number of resources for getting help in Testopia. You should first check out the FAQ on the wiki for additional pointers not included in this manual. If your problem is not there, asking your question on the mailing lists or in the chat room are your next best options. Please do not email the developers directly with support questions. They are busy people like you who have other commitments and cannot expect to be available at all times. Besides, asking your questions in a public forum allow more people the chance to respond and might get you a solution more quickly.

Mailing Lists

- support-webtools@lists.mozilla.org
- dev-apps-webtools@lists.mozilla.org

IRC Chatroom

<irc://irc.mozilla.org/testopia>

Wiki

<http://wiki.mozilla.org/Testopia>

Project Home Page

<http://mozilla.org/projects/testopia>

Reporting Bugs

Testopia is still a work in progress. As such, it still has a lot of bugs to be worked out. If you encounter a bug or find some way to make Testopia better, please log it at

http://bugzilla.mozilla.org/enter_bug.cgi?product=Testopia

Be sure to include steps to reproduce the problem and what browser you are using to access Testopia.

Bibliography

Copeland, L. (2004). A Practitioner's Guide to Software Test Design
Norwood, MA: Artech House

IEEE Std 829-1998 IEEE Standard for Software Test Documentation (n.d).
Retrieved February 28, 2007 from
http://standards.ieee.org/reading/ieee/std_public/description/se/829-1998_desc.html

Lohmeyer, J. (2004, April 27). Open-Source Bug Tracking with Bugzilla. Linux Journal.
Retrieved February 27, 2007, from <http://www.linuxjournal.com/article/7216>

Meyers, G.J. (2004). The Art of Software Testing, Second Edition
Hoboken, NJ: John Wiley & Sons.

Runnels, D.L. (1999). How to Write Better Test Cases
Retrieved September, 2006 from
<http://www.stickyminds.com/getfile.asp?ot=XML&id=2136&fn=XDD2136filelistfilename1%2Epdf>

Sherry, R.(2006, August).Testopia
Retrieved January 23, 2007 from
<http://www.rosiesherry.com/softwaretesting/show/Testopia>

Why Software Testing is Important ??? - Good one! (June 6, 2006).
Retrieved January 23, 2007 from
<http://techiecorner.blogspot.com/2006/06/why-softwaretesting-is-important-good.html>