



# Tracking Test Scripts

Article #1 in the 'Introduction to QaTraq Series'

## **Introduction to QaTraq Series**

The 'Introduction to QaTraq Series' is a series of 5 articles looking at how the freely available software testing and test management tool 'QaTraq' can help address some of the common test process issues. In this first article we look at how QaTraq provides you with clear visibility of your test scripts and how you can develop your test script execution strategy against defined builds of the product under test.



## Introduction

It's a seemingly simple task to track the test scripts run for the different builds of a product under test. Yet when you consider that even on a small test project you might have 20 different builds of a product and 30 different test scripts to track that quickly turns into the potential for 600 test script instances.

**20 builds X 30 test scripts = 600 possible test script instances**

Add to that, 5 different testers running the scripts, and different versions of the same test script. Then attempt to pin point exactly which test scripts have been run, which versions of the test scripts were used and which testers ran them. Then, finally attempt to link all of this to the builds of the product! Suddenly this seemingly simple task quickly explodes into a real problem. "Help! Which scripts have been run against which builds?" Not an uncommon cry from someone in a QA team trying to keep track of it all. So how can QaTraq help you gain control in such a scenario?

## In Control

Before we describe how to take control lets just take a quick look at what it's like to be in control. We'll use a small example with 9 different test scripts and 7 builds (or versions) of the product under test. With this small sample we can already see that we have the potential for 63 test script instances (9 test scripts x 7 builds).

Yet with one simple report from QaTraq (see Diagram A) it's easy to see that in fact we've decided to run just 24 instances of our test scripts (24 instances of the 9 different test scripts. E.g. Test Script 1 is run for Build 1, Build 4 and Build 7. This makes up 3 of the 24 total test script instances).

The screenshot shows a report titled "Results" with the subtitle "Test Scripts per Version for 'Product 3' Product." It includes the following information:

- Plans: test plan 1
- Product All
- Versions:

Product Version	Build 1	Build 2	Build 3	Build 4	Build 5	Build 6	Build 7
Test Design							
Test Script							
test design							
Test Script 1	TSC14-0.1			TSC23-0.3			TSC32-1.0
Test Script 2	TSC15-0.1			TSC24-0.2			TSC33-1.0
Test Script 3	TSC16-0.1			TSC25-0.2			TSC34-1.0
Test Script 4		TSC17-0.1			TSC26-0.2		TSC35-1.0
Test Script 5		TSC18-0.1			TSC27-0.3		TSC36-1.0
Test Script 6		TSC19-0.1			TSC37-0.4		TSC38-1.0
Test Script 7			TSC20-0.1			TSC29-0.3	
Test Script 8			TSC21-0.1			TSC30-0.3	
Test Script 9			TSC22-0.1			TSC31-0.2	

Diagram A – 'Scripts per Build' Report

What else can we tell from this report? Well, this report shows many important points about our test script execution strategy. We'll come onto these points in a moment. First though, we need to understand that a Test Script has three important identifiers (It's crucial to understand that these three identifiers are not just common to QaTraq. Any test process should have test scripts with Titles, unique identifiers and clear version control):

- Unique ID:** every instance of a test script has a unique identifier. This unique numerical identifier is in the format of 'TSCnn'. For example 'Test Script 1' assigned to Build 1 is identified as 'TSC14'.
- Version:** every instance of a test script has a version. This version identifier is in the format of 'TSCnn-V.v', where 'V' is the major version number and 'v' is the minor version number. For example Version 0.1 of Test Script 1 (TSC1-0.1) is assigned to Build 1. Version 0.3 of Test Script 1 (TSC23-0.3) is assigned to Build 4.
- Title:** the test script title should describe the area of functionality that the test script and the associated test cases cover. The Test Script title is used as a common identifier for different instances and different versions of a similar test script (e.g. the title 'Test Script 1' refers to the three instances, TSC14, TSC23 and TSC32). The title 'Test Script 1' also encompasses the different versions of the same test script (e.g. v0.1, v0.3 and v1.0 shown for 'Test Script 1' in Diagram A).

Product Version	Build 1	Build 2	Build 3	Build 4
Test Design				
Test Script				
test design				
Test Script 1	<a href="#">TSC14-0.1</a>			<a href="#">TSC23-0.3</a>
Test Script 2	<a href="#">TSC15-0.1</a>			<a href="#">TSC24-0.2</a>

Diagram B

In Diagram B we can see that Test Script 1 has evolved from version 0.1 (which was run on Build 1), to version 0.3 (which was run on Build 4). It is possible that this

change in versions is because we've added test cases to Test Script 1 prior to the script being run against Build 4 (you could see exactly what the changes are by clicking on the two links for TSC14-0.1 and TSC23-0.3).

Build 6	Build 7
	<a href="#">TSC32-1.0</a>
	<a href="#">TSC33-1.0</a>
	<a href="#">TSC34-1.0</a>
	<a href="#">TSC35-1.0</a>
	<a href="#">TSC36-1.0</a>
	<a href="#">TSC38-1.0</a>
<a href="#">TSC29-0.3</a>	
<a href="#">TSC30-0.3</a>	
<a href="#">TSC31-0.2</a>	

Diagram C

In Diagram C we can see that we didn't run Test Scripts 7, 8 or 9 for the final Build (Build 7). Perhaps functionality tested in scripts 7, 8 and 9 hadn't changed between Build 6 and Build 7. Perhaps though, there were big changes in functionality between Build 6 and Build 7 and we've just identified a hole in our testing.

In Diagram C we can also see that Test Scripts 7, 8 and 9 are all still in draft (i.e. version numbers are at 0.3, 0.3 and 0.2 respectively). Whereas Test Script 6 (TSC38-1.0) which was run on the final build, Build 7, was at version 1.0. If we were running a formal review and sign off process for our test scripts it could cause us some concern that Test Scripts 7, 8, and 9 were not formally reviewed and moved to version 1.0 for the final test script runs. This supposition is largely based on the assumption that you have implemented a review and sign-off process (we'll come on to implementing a review process using

QaTraq in one of our later articles in this series).

So, we've seen how QaTraq can give us control of our Test Scripts and how QaTraq can help pin point where we are with running those scripts against different builds. So how do we build up our test script repository to produce the kind of information seen above?

## Taking Control

There are three key steps (and 1 optional step) when it comes to building up and taking control of your test scripts with QaTraq:

1. Define the product and the builds to be tested
2. Create a set of 'placeholder' test scripts (optional step)
3. Develop the test scripts
4. Copy, delete, modify and create new test scripts

*We're assuming here that you know how to create, modify, delete and copy test scripts in QaTraq. If you need detailed instructions on how to create test scripts please refer to the User Guide (links to the User Guide can be found at the end of this article).*

### 1. Define the product and the builds to be tested

Before you start assigning test scripts to product builds you need to define the product and the builds associated with the product. QaTraq always uses the term 'Version' to specify a release of a product but there's nothing to stop you using this field to specify 'Builds' if you so wish (see the Definitions section of a description of Versions and Builds). With qatraq\_6\_2 you can also specify dates for builds, which helps to order the versions in the reports correctly.

## 2. Create a set of 'Placeholder' Test Scripts (optional step)

You can define a number of test scripts before the testing even begins (or even after the testing starts) just to identify areas that need to be tested. These placeholder test scripts will not be run and will never have any test results entered against them. However, these placeholders, identifying areas for testing, can be used as templates for creating other test scripts as the project progresses.

So to start with define an arbitrary test build, say 'Build 0'. Then create a number of test scripts (these test scripts don't even need any test cases for now). Each test script will identify an area for testing (using a descriptive test script title).

Product Version	Build 0	Build 1	Build 2
<i>Test Design</i> Test Script			
<i>test design</i>			
Test Script 1	TSC53-0.2		
Test Script 2	TSC54-0.2		
Test Script 3	TSC55-0.2		

Diagram D

From Diagram D you can see that we've defined 3 test scripts. These test scripts identify 3 different areas for testing. At the beginning of the project none of these scripts contain any test cases, although we will be adding test cases as the project progresses.

## 3. Develop the Test Scripts

As we begin to learn more about the product under test we can start to add test cases to our test scripts. To begin with we may only be adding test cases to the 'placeholder' test scripts. When the first build of software is delivered to the test team we'll copy or create new test scripts and assign them to the new build. You will see from Diagram E that we've assigned 'Test Script 1' and 'Test Script 2' to Build 1.

You will also notice from Diagram E that 'Test Script 3' hasn't been assigned against any builds of the software. A clear indication that we still have some work to do in this area (and a good reason why it's useful to create these placeholder test scripts).

You can probably see that it's very easy just to assign a copied test script (along with the associated test cases) to a new build. Thus creating a test script instance for a particular build. What you may find even more useful is that you can create these test script

Product Version	Build 0	Build 1	Build 2	Build 3
<i>Test Design</i> Test Script				
<i>test design 1</i>				
Test Script 1	TSC53-0.2	TSC64-0.1	TSC66-0.1	
Test Script 2	TSC54-0.2	TSC65-0.1		
Test Script 3	TSC55-0.2			

Diagram E

instances before you even start adding test cases to the scripts, and before you even receive the first build for testing. In this way you can clearly map out your testing at a very early stage in the project.

## 4. Copy, Delete, Modify and create New Test Scripts

As builds of software get delivered you will start to juggle and assign your resources to complete the testing. You can create as many or as few test script instances as you need to give you the best coverage for the resources you have at your disposal. Just use the Copy, Delete, Modify and New test script functionality in QaTraQ to produce the test script execution strategy you require.

In Diagram E above we might have added numerous test cases to the placeholder test script TSC53-0.2. As we received build 2 we could copy this placeholder test script to create TSC66-0.1, assigning it to Build 2. You can see that the action of copying the original placeholder script easily creates a test script instance so that testing can start on Build 2 (as shown by the dashed line in Diagram E above).

## Summary

We have taken a quick look at how QaTraq can give you control over many test scripts, with different test script versions, run against different builds of a product. With QaTraq tracking the test script Unique Ids, Versions and Titles we can quickly see our test script execution strategy using the 'Scripts per Build' report.

We have also looked at how we can develop our test script execution strategy by completing four steps; 1) Define the product and the builds which are to be tested 2) Create a set of 'placeholder' test scripts 3) Develop the test scripts 4) Copy, delete, modify and create new test scripts.

With one QaTraq report and 4 steps to develop your test script execution strategy this seemingly simple task really is simple!

## Top Tips

### Top Tip #1

Say you have the same test script assigned to a number of different builds. Then you decide you need to modify a test case, which is included in all of these test scripts. When you modify the test cases use the QaTraq 'Update existing Test Scripts' function to automatically update all of the test scripts that contain this test case. See the 'Test Scripts - Modifying' section of the User Guide for more information (links to the User Guide can be found at the end of this article).

### Top Tip #2

Use the 'Scripts per Build' report as the main view to create all the test script instances you need. Each time you need to create an instance of a test script for a particular build *hold shift down on your keyboard and left click on the test script (TSCnn-V.v) link* in the 'Scripts per Build' report. This will open up an existing test script instance in a new browser window. Copy this test script, selecting the new build that you'd like to run the script on (make sure you keep the test script title the same). Save and then close the test script browser window. Press F5 to refresh the 'Scripts per Build' report view and you'll see the new test script instance in the table.

## Definitions Used in this Article

**Test script:** a document that describes in detail how a test is to be conducted. A test script will contain information that outlines the test, defines the pre-requisites and contains a number of test cases. The user will enter the outline and pre-requisite text and then include a number of test cases, which need to be included within the test script.

**Test Script Instance:** an instance of a test script that is assigned against a product build or version. A test script instance is identified by the test script title, a unique identifier and the version number.

**Build and Version:** the terms build and version are used interchangeably in this document. In QaTraq the term Product Version is always used. However, how you use the Version identifier in QaTraq is largely dependent on your development process. If you tend to deal with builds of software in your test and development cycle it may be more logical to specify build titles in the QaTraq Version field.

**Build:** could be a specific compile of source code, with a unique identifier, that will be presented to the test team for testing.

**Version:** could be a specific release of the software, with a unique identifier, that is presented to the test team for testing (we think of versions as a super set of a build i.e. a version might contain a build of the software plus additional components like user guides, install scripts, etc)

## Contact Us

If you would like to provide us with any feedback or comments on this article please contact us at [feedback@traqsoftware.co.uk](mailto:feedback@traqsoftware.co.uk).