AUTOMAATIOTESTAUS JA ROBOT FRAMEWORK

Asennus, testien kirjoittaminen sekä ylläpidettävyys

Jani Koskela

Opinnäytetyö
Maaliskuu 2012

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) KOSKELA, Jani	Julkaisun laji Opinnäytetyö	Päivämäärä 12.03.2012
	Sivumäärä 107	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
AUTOMAATIOTESTAUS JA ROBOT FRAMEWORK
Asennus, testien kirjoittaminen sekä ylläpidettävyys

Koulutusohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
RANTALA, Maj-Lis
KOTKANSALO, Jouko

Toimeksiantaja(t)
Digia

Tiivistelmä

Opinnäytetyössä tutkittiin Robot Frameworkin soveltuvuutta testauksen automatisointiin.

Opinnäytetyössä Robot Framework asennettiin Windows 7- ja Ubuntu 11.04 -käyttöjärjestelmille. Käyttöjärjestelmille asennettiin Python, Java JRE, Jython, Robot Framework ja Robot Framework SeleniumLibrary -ohjelmat ja molemmille käyttöjärjestelmille kirjoitettiin asennusohjeet, jotka löytyvät liitteet osiosta. Asennuksissa kohdatut ongelmat ja niihin löydetyt ratkaisut on kuvattu opinnäytetyössä.

Asennuksien lisäksi Robot Framework otettiin käyttöön Atlassian Bamboo 3.4.2- ja TeamCity 6.5.6 -jatkuvan integraation ympäristöissä. Ympäristö asetettiin hakemaan testit versionhallinnasta, suorittamaan testit ja lopuksi kertomaan käyttäjälle testien tulokset.

Opinnäytetyössä selvitettiin, kuinka testeistä saisi sellaisia, ettei niitä tarvitsisi muuttaa, tai tarvittava muutos olisi hyvin pieni, vaikka sivuston toteutus muuttuisi taustalla.

Robot Frameworkin testien kirjoittamiseen tarkoitetun RIDE-editorin käytettävyyttä arvioitiin 11 kriteerillä ja löydetyt käytettävyysongelmat arvioitiin 3-tasoisella asteikolla. Käytettävyys arvioitiin opinnäytetyössä saadun materiaalin avulla.

Opinnäytetyössä kartoitettiin testien uudelleenkäytön mahdollisuuksia. Mitä tapoja Robot Framework tarjoaa testien jakamiseen pienempiin osiin ja uudelleen käyttää näitä osia myöhemmin.

Avainsanat (asiasanat)

Robot Framework, automaatiotestaus, Atlassian Bamboo, TeamCity, RIDE

Muut tiedot

Liitteenä asennusohjeita ja käyttöönotto-ohjeita, 56 sivua

Author(s)
KOSKELA, Jani

Type of publication
Bachelor´s Thesis

Date
12.03.2012

Pages
107

Language
Finnish

Confidential
( ) Until

Permission for web publication
(X)

Title
TEST AUTOMATION AND ROBOT FRAMEWORK
Installation, testing and maintenance

Degree Programme
Software Engineering

Tutor(s)
RANTALA, Maj-Lis
KOTKANSALO, Jouko

Assigned by
Digia

Abstract

The objective of this thesis was to study the suitability of Robot Framework for test automation and it was assigned by Digia.

The thesis describes Robot Framework installation to Windows 7 and Ubuntu 11.04 operating systems. Python, Java JRE, Jython, Robot Framework and Robot Framework SeleniumLibrary programs were installed to these systems. For both systems there is an installation guide which can be found as an appendix. The encountered problems and solutions for those problems are described in the thesis.

In addition to the installations Robot Framework was introduced in Atlassian Bamboo 3.4.2 and TeamCity 6.5.6 continuous integration environments. The environments were set to get the tests from the version control, to execute the tests and finally show the user tests results.

This thesis researched how tests could be written in a way, that they do not need to change even though the implementation changes. If a change is necessary, what is the best way to minimize the necessary change?

Robot Framework test writing editor RIDE usability was evaluated using 11 criteria and the usability issues that were found were rated using a 3-step scale. Usability was evaluated based on the material gained during the thesis writing process.

In this thesis opportunities for reuse were studied, in particular in what ways Robot Framework offers to divide tests into smaller components and reuse these components later.

Keywords
Robot Framework, test automation, Atlassian Bamboo, TeamCity, RIDE

Miscellaneous
Appendix contains installation guides and initialization guides for total amount of 56 pages

SISÄLTÖ

KUVIOT

KÄSITTEET

Atlassian Bamboo

Atlassian Bamboo on jatkuvan integraation palvelin. Opinnäytetyössä Robot Framework asennettiin Bamboo-palvelimelle ja Bamboon käynnistämä Robot Framework ajoi testejä testisivustolle.

HTML

Lyhenne sanoista *Hypertext Markup Language*. HTML on kieli, jolla www-sivut usein kirjoitetaan.

Jatkuva integraatio

Jatkuva integraatio on ympäristö, joka hoitaa tietovarastossa (*Repository*) olevan tiedon koostamisen, testien ajamisen sekä tiedon julkaisun palvelimelle automaattisesti.

Java

Opinnäytetyössä käytettiin Java Runtime Environment -ympäristöä suorittamaan Robot Frameworkin testejä Jythonilla.

Jython

Jython on ohjelmointikieli, joka pohjautuu Pythoniin, mutta on täysin Javaa. Jythonia käytettiin Robot Frameworkin testien ajamiseen rinnakkaisena vaihtoehtona Pythonille.

MySql

MySql on vapaan lähdekoodin tietokanta. Opinnäytetyössä tietokantaa käytettiin esimerkkitietokantana, johon Robot Frameworkilla syötettiin testidata ennen testien suorittamista. Suorittamisen jälkeen testidata poistettiin kannasta Robot Frameworkilla.

Python

Python on ohjelmointikieli. Pythonia käytettiin Robot Frameworkin testien suorittamiseen.

RIDE

RIDE on editori, joka on suunniteltu Robot Frameworkin testien kirjoittamiseen.

Robot Framework

Nokia Siemens Networkin kehittämä yleiskäyttöinen testiautomaatiotyökalu hyväksymistestaukseen. Robot Framework on avoimen lähdekoodin sovellus, joka on julkaistu Apache License 2.0 -lisenssillä (ks. http://www.apache.org/licenses/LICENSE-2.0.html). (Robot Framework 2011.)

TeamCity

TeamCity on jatkuvan integraation palvelin. Opinnäytetyössä Robot Framework asennettiin TeamCity-palvelimelle ja TeamCityn käynnistämä Robot Framework ajoi testejä testisivustolle.

Testi

Opinnäytetyössä testillä tarkoitetaan yksittäistä testitapausta, jolla varmistutaan siitä, että järjestelmä toimii oikein.

XML

Metakieli. Lyhenne sanoista *Extensible Markup Language*. Robot Frameworkin tuottama "output.xml"-tiedosto oli XML-formaatissa.

xunit

XML-muodossa oleva tiedosto. Jatkuvan integraation ympäristöt Atlassian Bamboo ja TeamCity pystyivät lukemaan xunit-tiedostosta testien tulokset ja kertomaan ne käyttöliittymässään käyttäjälle.

1. TYÖN LÄHTÖKOHDAT

1.1. Toimeksiantaja

Opinnäytetyön toimeksiantajana oli Digia Oyj, joka on suomalainen ohjelmistoratkaisu- ja palveluyhtiö. Digia jakautuu strategisesti neljään painopistealueeseen, jotka ovat asiakaskohtaiset ratkaisut ja palvelut, toimialakohtaiset monistettavat ohjelmistoratkaisut, kansainvälinen ohjelmistoliiketoiminta ja uudet markkina-alueet. Digian ydintoimialoihin kuuluvat teleoperaattorit, pankki- ja vakuutustoimiala, julkinen hallinto ja kaupan toimiala. Digia hakee lisäkasvua laajentumalla nopeasti kasvaville markkina-alueille kuten Venäjälle. (Digian vuosikertomus 2012.)

1.2. Opinnäytetyön tavoitteet

Opinnäytetyön tarkoituksena oli määrittää, kuinka Robot Framework soveltuu websovellusten testaamiseen. Opinnäytetyö aloitettiin tutkimalla kirjallisuudesta testauksen automatisointia, käytettävyyttä, uudelleenkäytettävyyttä sekä testauksen automatisoinnin työkaluja.

Opinnäytetyön kolmannessa luvussa raportoidaan, kuinka Robot Framework saatiin asennettua Windows- ja Ubuntu-käyttöjärjestelmille. Asennuksessa kohdatut ongelmat ja niihin löytyneet ratkaisut kuvattiin opinnäytetyöhön. Lisäksi asennuksista tehtiin erilliset asennusohjeet kummallekin käyttöjärjestelmälle. Asennusohjeet löytyvät liitteistä 1, 2 ja 3.

Neljännessä luvussa kuvataan Robot Frameworkin käyttöönottoa Atlassian Bamboo ja TeamCity -jatkuvan integraation palvelimilla. Kohdatut ongelmat ja niihin löytyneet ratkaisut kuvattiin jälleen opinnäytetyöhön. Käyttöönotoista tehtiin myös käyttöönotto-ohjeet, joissa kerrottiin, mitä käyttäjän tulisi tehdä, jos hän haluaa käyttää Robot Frameworkia Atlassian Bamboo tai TeamCity-jatkuvan integraation palvelimella. Käyttöönotto-ohjeet löytyvät liitteistä 4 ja 5.

Opinnäytetyön viidennessä luvussa käsitellään testien kirjoittamista. Ensiksi selvitettiin, miten Robot Frameworkilla pystyy lisäämään tietoa suoraan tietokantaan ja

kuinka lisätyt tiedot pystyy poistamaan tietokannasta testien suorittamisen jälkeen. Tämän jälkeen tutkittiin, kuinka testejä pystyy jakamaan suoritettaviin kokonaisuuksiin. Viimeisenä asennettiin Robot Frameworkin testien kirjoittamiseen tarkoitettu RIDE-editori ja tutkittiin testeihin kohdistuvaa muutostarvetta tilanteissa, joissa sivuston toteutus muuttuu.

Robot Frameworkin RIDE-editorin käytettävyyttä arvioidaan kuudennessa luvussa. Käytettävyyttä arvioitiin 11 eri perusteella ja löydetyt käytettävyysongelmat arvioitiin kolmitasoisella asteikolla.

Seitsemännessä luvussa kuvataan opinnäytetyön aikana huomattuja testien uudelleenkäyttömahdollisuuksia. Uudelleenkäyttöä tutkittiin muun muassa avainsanojen (*keyword*) sekä kirjastojen avulla. Luvussa selvitettiin testien jakamista useisiin tiedostoihin ja tiedoston uudelleenkäyttöä.

Kahdeksannessa luvussa koostetaan asioita, jotka liittyvät Robot Frameworkin käyttämiseen testauksen automatisoinnin työkaluna. Koostamisen lähtökohtana oli pohdinta siitä, kannattaako Robot Framework ottaa käyttöön ohjelmistoprojektissa.

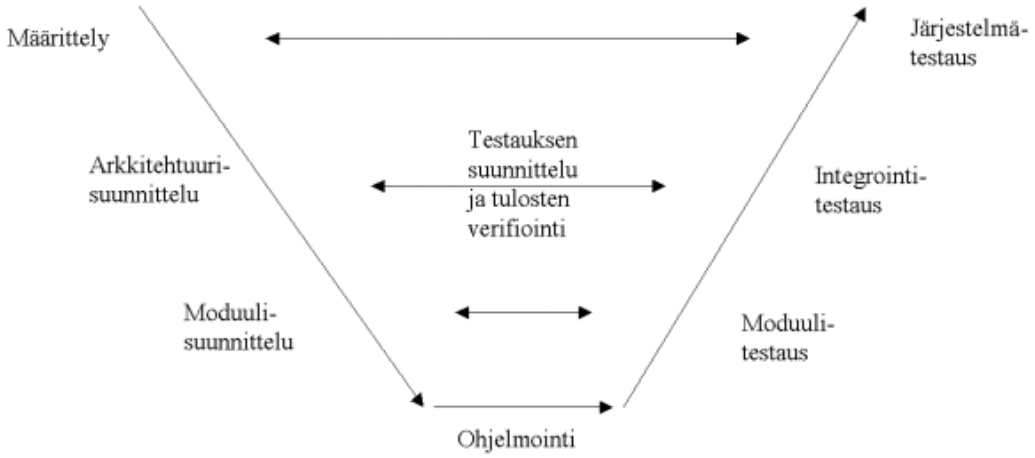
Yhdeksännessä luvussa pohditaan mahdollisia jatkokehityksen kohteita. Kohteita löytyi useita, koska opinnäytetyön rajauksien takia kaikkia osa-alueita ei pystytty tutkimaan riittävällä tasolla.

2. TESTAUKSEN AUTOMATISOINTI

Testaukseen liittyviä työvaiheita on neljä: suunnittelu, testiympäristön luonti, testin suorittaminen ja tulosten tarkastelu (Haikala & Märijärvi 2006, 289-290). Suunnittelussa suunnitellaan, mitä asioita testataan ja kuinka niitä testataan. Testiympäristön luonnissa luodaan testiympäristö, jossa testattavaksi valitut testit pystytään suorittamaan. Seuraavaksi suoritetaan testit testiympäristössä ja lopuksi tarkastellaan, miten testit onnistuivat.

Testaus voidaan jakaa kolmeen eri tasoon: moduuli-, integrointi- ja järjestelmätestaus. Moduulitestaus pitää sisällään alimman tason testauksen, jossa testataan pieniä ohjelmanosia. Moduulitestauksen suorittaa yleensä kehittäjä itse. Integrointitestauksessa testataan, että järjestelmän eri palikat toimivat oikein keskenään. Tämänkin kehittäjä testaa, yleensä, itse. Järjestelmätestaus puolestaan kuuluu testaajalle. Järjestelmätestauksessa järjestelmää testataan kokonaisuutena määrittelydokumentaatiota vasten. (Haikala & Märijärvi 2006, 289-290.)

Haikala ja Märijärvi (2006, 290) kertovat myös, että mitä korkeammalla tasolla V-mallissa (ks. kuvio 1) ollaan, sitä kalliimmaksi virheen korjaaminen tulee. Tällöin korjaaminen voi aiheuttaa ongelmia muualla järjestelmässä ja vaatii uudelleen testauksen. Tätä uudelleen testausta kutsutaan regressiotestaukseksi.



KUVIO 1. Testauksen V-malli (ks. alkuperäinen kuvio: Haikala & Märijärvi 2006, 289.)

Regressiotestaus voidaan suorittaa manuaalisesti, automaattisesti tai näitten yhdistelmällä. Manuaalisessa testauksessa käyttäjä etsii virheitä järjestelmästä ja raportoi löytyneistä virheistä kehittäjälle. Testeissä kokeillaan erilaisten arvojen yhdistelmiä, joita verrataan tulokseen, jonka järjestelmän olisi pitänyt antaa. Kehitysvaiheessa manuaalinen testaaminen tulee suorittaa jokaisen koodimuutoksen tai asetuksen muuttumisen jälkeen. (Why automated testing? 2011.)

Automatisoinnissa manuaalinen testaaminen suoritetaan koneellisesti jonkin testausohjelman avulla (Pohjolainen 2003, 23). Viimeinen vaihtoehto on näiden edellä mainittujen yhdistelmä, jossa osa testeistä suoritetaan automaattisesti ja osa manuaalisesti. Esimerkiksi pitkän testin muuttumaton alkuosa voidaan automatisoida ja loppuosa testataan manuaalisesti.

2.1. Miksi testejä automatisoidaan?

Testien automatisoinnilla saadaan lisättyä tehokkuutta ja testien kattavuutta (Why automated testing? 2011). Jokainen ohjelmistokehitysryhmä testaa koodinsa, mutta silti toimitetussa ohjelmassa on aina virheitä. Testaajilla on tapana löytää osa virheistä, ennen kuin ohjelma julkaistaan, mutta silti virheillä on tapana ilmestyä uudelleen manuaalisessa testausprosessissa.

Automatisoinnilla parannetaan myös tarkkuutta. Kaikkein tunnollisinkin testaaja tekee joskus virheen toistuvassa manuaalisessa testauksessa. (Why automated testing? 2011). Automatisoimalla vältytään tältä. Automatisointi suorittaa joka kerta testit samalla tavalla ja näin ollen parantaa tarkkuutta.

Testien automatisoinnilla voidaan säästää aikaa ja rahaa. Testit tulisi toistaa joka kerta, kun järjestelmään tulee muutoksia (Why automated testing? 2011). Kuvitellaan tilanne, jossa ohjelman A tulee toimia käyttöjärjestelmillä C ja D, asetuksilla E ja F. Tässä tilanteessa ohjelmaa A koskevat 20 testiä tulisi suorittaa neljä kertaa, kaksi kertaa käyttöjärjestelmälle C asetuksilla E ja F ja kaksi kertaa käyttöjärjestelmälle D asetuksilla E ja F. Automatisoimalla testit säästetään tässä tapauksessa paljon aikaa.

Automatisoinnilla voidaan myös tehdä asioita, jotka ovat manuaalisesti hyvin vaikeita toteuttaa. Hyvänä esimerkkinä on kuormitustestaus, jossa varmistutaan, että järjestelmä kestää ennalta sovitun määrän yhtäaikaisia käyttäjiä. Esimerkiksi 1 000 yhtäaikaisen käyttäjän mallintaminen käy hyvin hankalaksi yritykselle, jossa työskentelee alle 100 henkilöä.

Automatisoinnilla on myös yksi suuri hyöty manuaaliseen nähden. Automatisointiin erikoistunut ohjelma kokoaa testien tulokset yhteen ja kertoo, mikä on järjestelmän tilanne. Tietenkin testaaja voi tehdä sen manuaalisesti testien suorittamisen jälkeen, mutta tämä vie aikaa, joka on pois testaamisesta.

2.2. Automatisoinnin ongelmat

Automatisointi ei aina ole kannattavaa. Automatisointi vie yleensä 3-10 kertaa enemmän aikaa kuin manuaalisen testin kertasuoritus (Pohjolainen 2003, 7). Pohjolainen (2003) jatkaa edelleen, että säästö saadaan testauksen toistoista. Jos oletetaan, että testaus suoritetaan vain kerran, testauksen automatisoinnin tarve on hyvin pieni. Automatisoinnin hyötyä ei saada käytettyä, koska toistokertoja on sen verran vähän. Jos testaus puolestaan suoritetaan esimerkiksi 20 kertaa, testauksen automatisoinnilla voidaan saada säästöjä aikaan.

Pohjolainen (2003, 39) kertoo, että automatisointiin liittyy myös paljon ongelmia, esimerkiksi automatisointi ei löydä paljon uusia virheitä. Suurin osa virheistä löydetään ensimmäisellä testauskerralla, ja sen jälkeen löytyy enimmäkseen virheitä, jotka aiheutuvat virheiden korjaamisesta (mts. 39).

Pohjolaisen (2003) mukaan automatisoitujen testien ylläpito on lopettanut monta testauksen automatisointia heti alkuunsa. Uudet ohjelmistoversiot tarvitsevat uusia testejä ja useasti muuttavat olemassa olevia testejä, koska toiminnot muuttuvat. Uusia testejä tarvitaan, kun järjestelmään on lisätty jokin uusi toiminnallisuus. Testejä voidaan joutua poistamaan turhina, kun jotain toiminnallisuutta poistetaan järjestelmästä. Testejä voidaan joutua myös muokkaamaan, kun jokin järjestelmän toiminnallisuus muuttuu. Tämä vaikuttaa ylläpitokustannuksiin, ja tämän takia automatisoidun testauksen ylläpitokustannukset ovat suuremmat kuin manuaalisen. (Mts. 2003, 39-41.)

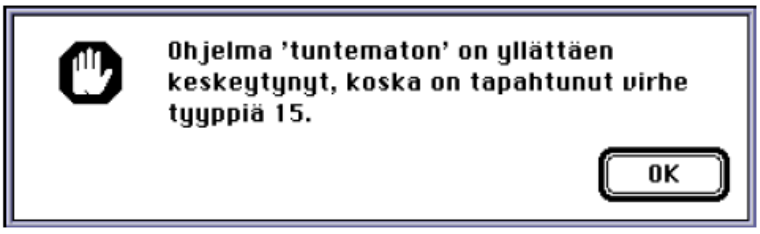
2.3. Käytettävyyden arviointi

Opinnäytetyössä tutkittiin Robot Frameworkin käytettävyyttä automatisoinnin työkaluna. Käytettävyys on mittari, joka mittaa vuorovaikutteisen käyttöliittymän kuten

nettisivun tai sovelluksen käyttäjäkokemusta. Käyttäjäystävällinen käyttöliittymä on helppo oppia, auttaa käyttäjää hänen tehtävissään ja tavoitteissaan tehokkaasti. Käyttäjäystävällinen käyttöliittymä on myös miellyttävä käyttää sekä opettava. (UsabilityFirst 2012.)

Käyttöliittymän käytettävyyttä voidaan arvioida kutsumalla järjestelmän käyttäjiä käytettävyyden arviointitilaisuuteen. Tilaisuudessa käyttäjälle annetaan tehtäviä, jotka käyttäjän tulee tehdä ilman tutkijan apua. Tutkija tarkkailee käyttäjän käyttäytymistä, tunteen purkauksia ja käyttäjän tehokkuutta, kun hän suorittaa annettuja tehtäviä. Useilta käyttäjiltä saatu tieto kertoo tutkijoille, mitä ja mistä järjestelmää tulee parantaa, jotta se on käytettävämpi. (UsabilityFirst 2012.)

Käyttöliittymää voidaan arvioida myös erilaisten käytettävyysperiaatteiden eli heuristiikkojen avulla. Arvioinnissa järjestelmää verrataan esimerkiksi Nielsen ja Molichin kymmenen säännön listaan (Usabilitybok 2012). Esimerkiksi järjestelmän tulee antaa käyttäjälle virhetilanteista mahdollisimman selkeä ja tarkka ilmoitus käyttäjän ymmärtämällä kielellä ja sanastolla (ks. kuvio 2). Virheilmoitus ei saa myöskään syyttää käyttäjää virheestä, vaan sen tulee ohjata käyttäjä korjaamaan virheensä. (Riihiaho 2012, 11.)



KUVIO 2. Esimerkki huonosta virheilmoituksesta (ks. alkuperäinen kuvio: Riihiaho 2012, 11)

Opinnäytetyössä arvioitiin Robot Frameworkin käytettävyyttä kuviossa 3 esiintyvillä arvoilla. Arvioinnin kohteena oli Robot Frameworkin tarjoama RIDE-editori, joka on suunniteltu Robot Frameworkin testien kirjoittamiseen (RIDE 2012).

KUVIO 3. Käytettävyyden arviointi (ks. alkuperäinen kuvio: Käytettävyyden perusteet 2012, 22)

Löydetyt käytettävyysongelmat arvioitiin kolmi-portaisella asteikolla:

1. Pieni kosmeettinen ongelma tai ei käyttöä haittaava ongelma
2. Ongelma, joka haittaa käyttöä
3. Ongelma, joka estää ohjelman käytön

2.4. Uudelleenkäytettävyys

Opinnäytetyössä tutkittiin myös kirjoitettujen testien uudelleenkäytettävyyttä. Erosen (2010, 7) mukaan uudelleenkäytössä uusi ohjelmisto pohjautuu jollain muotoa olemassa olevaan ohjelmistoon. Sama pätee myös testeihin. Uusissa testeissä pyritään käyttämään mahdollisimman paljon jo tehtyjä testejä. Eronen (mts. 7) jatkaa edelleen, että uudelleenkäytöllä saadaan pienennettyä ohjelmiston toteuttamiseen tarvittavaa aikaa. Sama toteutuu myös testeissä. Jos esimerkiksi sisään kirjautuminen on jo toteutettu, käytetään sitä muissa testeissä. Jos sisään kirjautumisen testi on toteutettu hyvin, voidaan sitä käyttää jopa muissa projekteissa.

Erosen (2010, 8) mukaan nykyhetken merkittävimpiä uudelleenkäyttötapoja ohjelmistoissa ovat komponenttipohjainen uudelleenkäyttö, arkkitehtuuriratkaisujen uudelleenkäyttö ja ohjelmistotuotelinja. Lähestymistavat eivät ole toisiaan poissulkevia.

Testien uudelleenkäyttö voidaan toteuttaa esimerkiksi komponenttien uudelleenkäytöllä ja arkkitehtuuriratkaisulla. (Mts. 2010, 8.)

Komponenttipohjaisessa uudelleenkäytössä järjestelmä pyritään koostamaan jo olemassa olevista ohjelmistokomponenteista. Käytettävät komponentit on suunniteltu ja toteutettu varta vasten uudelleenkäyttöön. Arkkitehtuuripohjaisessa uudelleenkäytössä pyritään hyödyntämään suurempia osakokonaisuuksia kuin komponenttipohjaisessa uudelleenkäytössä. Tässäkin tapauksessa osakokonaisuudet on suunniteltu ja toteutettu varta vasten uudelleenkäyttöön. (Eronen 2010, 8.)

Ohjelmistotuotelinjaratkaisussa toteutetaan tietylle sovellusalueelle erilaisia ohjelmistotuotteita, jotka ovat ominaisuuksiltaan erilaisia. Tuotteiden arkkitehtuuri ja toteutus pohjaavat kuitenkin yhteen ja samaan tuotealustaan. (Eronen 2010, 8.)

Opinnäytetyössä keskityttiin tutkimaan vain komponenttipohjaista ratkaisua. Ennen opinnäytetyön aloittamista tiedettiin, että Robot Frameworkissa pystyy kirjoittamaan avainsanoja. Avainsanoilla pystytään määrittelemään, mitä testejä Robot Framework suorittaa, kun se käynnistetään tietyillä parametreilla.

2.5. Testauksen automatisoinnin työkaluja

Testauksen automatisointiin on olemassa monenlaisia työkaluja. Haikala ja Märijärvi (2006, 297) kertovat, että automatisointiin käytettäviä työkaluja ovat muun muassa testipetigeneraattorit, testitapausgeneraattorit, vertailijat ja testikattavuustyökalut. Robot Framework on yhdistelmä testipetigeneraattoria ja vertailijaa. Muita samantyyppisiä automatisointi työkaluja ovat muun muassa Selenium (ks. http://seleniumhq.org/) ja Watir (ks. http://watir.com/).

Testipetigeneraattoreilla testattavalle ohjelmalle luodaan testipeti, jossa halutulla testikuvauskielellä kuvattu testi suoritetaan. Testiin kuvataan testin vaiheet sekä haluttu tulos, jolloin tarkastelu on automatisoitavissa. Järjestelmätestauksessa testitapaukset voidaan nauhoittaa ja käyttää automatisoinnissa. (Haikala & Märijärvi 2006, 297.)

Testien generointi on Haikalan ja Märijärven (2006, 297) mukaan useimmissa tapauksissa mahdotonta. Testit generoidaan dokumentaatiosta ja jos dokumentaatiota ei ole kerrottu tarpeeksi tarkasti, ei generaattori osaa generoida oikeantyyppisiä testejä.

Vertailuohjelmilla verrataan testin tulostetta aikaisempiin tulosteisiin. Vertailuohjelmien ongelmana on muun muassa muuttuva päivämäärä. (Haikala & Märijärvi 2006, 297.) Jotta vertailuohjelma toimii, tulee testi rakentaa siten, että päivämäärän vaihtelut ja muut muuttuvat asiat eivät kuulu tarkastuksen piiriin.

Testikattavuusanalysaattorit ovat työkaluja, jotka mittaavat testin kattavuutta. Analysaattorit ovat toimintaperiaatteeltaan esiprosessoreita, jotka instrumentoivat moduulin koodia. Analysaattoreilta voi usein myös mitata suorituskertojen lukumäärän ja prosessoriajan käyttöä. (Haikala & Märijärvi 2006, 297.)

3. ROBOT FRAMEWORKIN ASENTAMINEN

Opinnäytetyössä Robot Framework asennettiin Windows- ja Ubuntu-käyttöjärjestelmille. Asennuksille asetettiin hyväksymiskriteereiksi seuraavat:

- Testiympäristössä saaadaan ajettua yksi testi onnistuneesti ja yksi testi epäonnistuneesti.
- Epäonnistuneesta testistä otetaan automaattisesti kuvaruutukaappaus.
- Käyttäjä pystyy lukemaan koosteen, jossa kerrotaan, montako testiä ajettiin ja montako meni läpi.

Asentamisen dokumentointi jaettiin kolmeen eri dokumenttiin. Liitteessä 1 kuvataan asentaminen Windows-käyttöjärjestelmälle ja liitteessä 3 kuvataan asentaminen Ubuntu-käyttöjärjestelmälle. Liitteessä 2 kuvataan, kuinka varmennettiin, että järjestelmä toimi oikein kaiken asentamisen jälkeen. Opinnäytetyön dokumentaatiossa kerrottiin kohdatuista ongelmista ja niihin löydetyistä ratkaisuista.

3.1. Windows 7 -käyttöjärjestelmälle

Opinnäytetyössä asennettiin Robot Framework Windows 7 -käyttöjärjestelmälle. Windows 7 -käyttöjärjestelmästä valittiin Windows 7 Professional 32-bit -versio (ks. http://windows.microsoft.com/en-US/windows7/products/home). Valmistelutoimina opinnäytetyölle asennettiin puhdas Windows 7 -käyttöjärjestelmä, jonka jälkeen aloitettiin Robot Frameworkin asentaminen.

3.1.1. Python 2.7.2

Robot Frameworkista valittiin asennettavaksi versio 2.6.3, joka vaati toimiakseen joko Pythonin tai Jythonin (User Guide 2011). Opinnäytetyössä päädyttiin asentamaan kuitenkin molemmat. Robot Frameworkin oppaassa (User Guide 2011) kerrottiin Pythonin käytöstä kolme sääntöä: Robot Framework ei tue Pythonin 3:sta, Robot Framework 2.5:stä lähtien Pythonista tulee käyttää vähintään versiota 2.5 ja aikaisemmat Robot Frameworkin versiot tukevat Python 2.3:sta. Näistä säännöistä johtuen valittiin asennettavaksi uusin Python 2 eli versio 2.7.2.

Pythonin asennus aloitettiin tarkistamalla, oliko asennusympäristöön esiasennettu Pythonia. Sitä ei ollut esiasennettu, joten ympäristöön asennettiin Python 2.7.2.

Asennus oli hyvin suoraviivainen, kuten liitteestä 1 käy ilmi. Ensiksi valittiin, kenelle Python haluttiin asentaa. Seuraavaksi valittiin, mihin kansioon Python asennettiin ja viimeiseksi valittiin, mitä ominaisuuksia Pythonista haluttiin asentaa.

Pythonin asennuksen jälkeen törmättiin ongelmiin. Komentokehotteessa ajettava komento ”python --version” ei näyttänyt, että Python olisi asennettu. Ratkaisu löytyi Robot Frameworkin käyttöohjeesta, jossa sanottiin, että Pythonin asennnuskansio tuli lisätä ympäristömuuttujaan "Path" (User Guide 2011). Kun tämä lisäys oli tehty, ”python --version” löysi ja tulosti Pythonin version.

3.1.2. Java Runtime Environment 1.6.0 päivitys 29

Robot Framework oppaassa (User Guide 2011) kerrottiin Jythonin käytöstä kaksi sääntöä: Robot Framework versiosta 2.5 lähtien Jythonista vaaditaan versio 2.5 ja

aikaisemmat Robot Framework -versiot ovat yhteensopivia Jythonin 2.2 kanssa. Näistä säännöistä johtuen valittiin asennettavaksi uusin Jython eli versio 2.5.2.

Jython 2.5.2 puolestaan vaati Java Runtime Environmentin version 1.5.0 tai uudemman (Jython 2011). Asennusympäristöön päädyttiin asentamaan uusin Java Runtime Environment eli 1.6.0 päivitys 29.

Kuten liitteessä 1 kuvataan, asennuksessa ei tarvinnut muuttaa kuin Javan asennuskansion osoite. Asennuksessa ei esiintynyt ongelmia.

3.1.3. Jython 2.5.2

Robot Framework 2.6.3 tarvitsi toimiakseen Jythonista version 2.5 tai uudemman. Säännöstä johtuen valittiin asennettavaksi Jython 2.5.2.

Heti aluksi törmättiin ongelmiin, kuinka saadaan Jythonin asennus käyntiin pääkäyttäjän oikeuksilla. Koska kyseessä oli jar-päätteinen tiedosto, asennuksessa jouduttiin käynnistämään komentorivi pääkäyttäjän oikeuksilla ja tämän jälkeen käynnistämään Jythonin asennusohjelma (ks. liite 1).

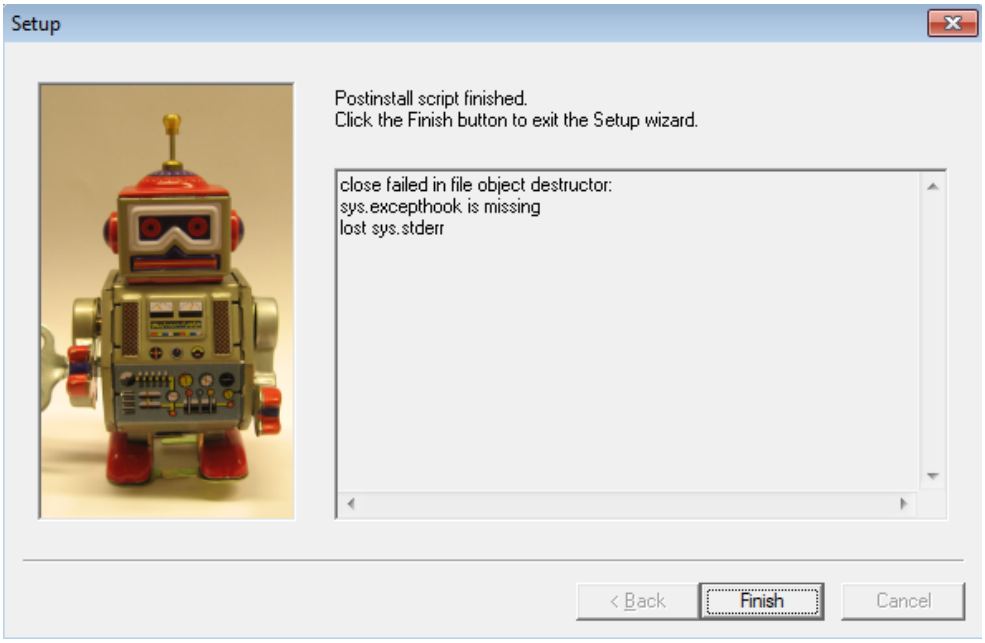
Jythonin asennuksessa valittiin ensimmäiseksi kieli, jolla Jython haluttiin asentaa. Tämän jälkeen törmättiin jälleen ongelmiin, mikä tyyppi Jythonista tarvitaan? Jythonista pystyi asentamaan kaiken kattavan, perus- tai minimiasennuksen. Opinnäytetyössä päädyttiin asentamaan perusasennus, koska asennusohjelma ehdotti sitä. Viimeisenä asennus tiedusteli Java-hakemistoa. Tällöin tarkastettiin vain, että Java-hakemisto, johon viitattiin, oli asennetun Javan kotihakemisto. (Ks. liite 1.)

Asennuksen jälkeen törmättiin jälleen ongelmiin. Komentoriviltä ajettavaa komentoa ”jython --version” ei löytynyt. Ratkaisu ongelmaan löydettiin kokeilemalla. Ensimmäiseksi kokeiltiin samaa ratkaisua kuin Pythonin asennuksessa, eli lisätä Jythonin kansio ympäristömuuttujaan "*Path*". Ongelma ratkesi kyseisellä korjauksella.

3.1.4. Robot Framework 2.6.3

Kun tarvittavat komponentit oli asennuttu, päästiin asentamaan Robot Frameworkia. Robot Frameworkin asentaminen oli hyvin helppo, koska siinä ei ollut mitään toimintoja, joita olisi pitänyt muistaa laittaa päälle. Asennuksessa vain tarkastettiin, että Robot Framework löysi asennetun Pythonin. (Ks. liite 1).

Kun asennus oli valmis, Robot Frameworkin asennus antoi virheilmoituksen (ks. kuvio 4). SourceForgen mukaan ongelma johtui Pythonissa olevasta virheestä (SourgeForge 2011). Virheellä ei ollut vaikutusta asennuksen onnistumiseen, vaan asennus saatiin suoritettua onnistuneesti läpi.



KUVIO 4. Robot Frameworkin virheilmoitus

Virheilmoituksen jälkeen törmättiin heti uuteen ongelmaan: järjestelmä ei löytänyt juuri asennettua Robot Frameworkia. Komentokehotteessa ajettavat komennot ”pybot --version” ja ”jybot --version” eivät tulostaneet Robot Frameworkin versiota. Robot Frameworkin käyttöohjeesta (User Guide 2011) löytyi ratkaisu tähän ongelmaan. Pythonin asennuskansion sisällä oleva script-kansio tuli lisätä myös ympäristömuuttujaan "*Path*" (User Guide 2011). Kun lisäys oli tehty, komennot ”pybot --version” ja ”jybot --version” -komennot löysivät asennetun Robot Frameworkin.

3.1.5. Robot Framework SeleniumLibrary 2.8

Viimeisenä asennusympäristöön asennettiin Robot Framework SeleniumLibrary 2.8. Kyseistä kirjastoa käytettiin opinnäytetyössä ottamaan kuvaruutukaappauksia epäonnistuneista testeistä.

Selenium-kirjaston asentaminen oli samankaltainen kuin Robot Frameworkin asentaminen eli tarkastettiin, että asennusohjelma löysi Pythonin. Asennuksen varmentaminen puolestaan osoittautui erittäin hankalaksi ja hyväksymiskriteerien toteuttaminen vielä hankalammaksi. Selenium-kirjasto käyttää selaimen käynnistämiseen Selenium Server -työkalua, jota ei aluksi löytynyt mistään. Lopulta työkalu löytyy, mutta heti törmättiin uuteen ongelmaan, kuinka se saadaan käyntiin siten, että virheilmoitukset nähdään, jos niitä tulee. Ratkaisuksi löytyi komentokehote. Komentokehotteella mentiin hakemistoon, jossa työkalu sijaitsi ja käynnistettiin se komennolla ”java -jar selenium-server.jar”.

Kun Selenium Server saatiin käyntiin, päästiin kirjoittamaan testejä, jotka testasivat, että asennetut komponentit toimivat oikein. Testien kirjoittaminen on selitetty liitteessä 2. Testien kirjoittamisen jälkeen annettiin Robot Frameworkille komento ”pybot my_test.txt”. Tämä komento ajoi kirjoitetut testit ja muodosti niistä kolme tiedostoa: *Output.*xml, Log.*html* ja *Report.html*. *Output.xml*-tiedosto sisälsi xml-formaatissa kaiken mitä Robot Framework teki testiajon aikana. Tästä tiedostosta Robot Framework muodosti *Log.html*-tiedoston, joka sisälsi kuvauksen avainsana kerrallaan, mitä Robot Framework teki. *Output.xml*-tiedostosta muodostettiin myös *Result.html*-tiedosto, joka koosti testiajon tulokset yhteen tiedostoon. Kuvat tiedostoista löytyvät liitteestä 2.

Viimeinen hyväksymiskriteeri oli, että epäonnistuneesta testistä otetaan kuvaruutukaappaus. Selenium-kirjastossa oli siihen tarkoitettu komento ”Capture Screenshot”. Kyseinen komento lisättiin avainsanan ”TearDown” jälkeen, joka suoritetaan aina, kun testi on suoritettu onnistuneesti tai epäonnistuneesti. Ongelmana oli vain se, että kuvaruutukaappaus tuli ottaa ainoastaan epäonnistuneesta testistä. Tähän löytyi ratkaisu Selenium-kirjaston avainsanoista. ”Run Keyword If Test Failed” suorittaa

perässä olevan komennon ainoastaan silloin, kun testi epäonnistuu. Koko komento on nähtävissä liitteessä 2.

3.1.6. Yhteenveto

Robot Frameworkin asennus Windows-käyttöjärjestelmälle onnistui, koska se täytti kaikki kolme asetettua ehtoa: Asennetulla Robot Frameworkilla pystyi suorittamaan sekä onnistuneen, että epäonnistuneen testin. Asennettu Robot Framework otti kuvaruutukaappauksen epäonnistuneesta testistä. Robot Framework loi myös tiedoston, josta kävi ilmi onnistuneiden ja epäonnistuneiden testien määrät.

3.2. Ubuntu 11.04 -käyttöjärjestelmälle

Opinnäytetyössä asennettiin Robot Framework myös Linux-käyttöjärjestelmälle. Linux käyttöjärjestelmistä valittiin Ubuntu 11.04 -versio (katso http://www.ubuntu-fi.org/). Valmistelutoimina opinnäytetyölle asennettiin puhdas Ubuntu 11.04 -käyttöjärjestelmä, jonka jälkeen aloitettiin Robot Frameworkin asentaminen.

3.2.1. Python 2.7.2

Pythonin asennus aloitettiin tarkastamalla oliko asennusympäristöön jo asennettu Python. Robot Frameworkin käyttöohje (User Guide 2011) kertoi, että Robot Framework 2.6.3 vaati Pythonista version 2.5. Terminaalissa ajettava "python --version" paljasti, että asennusympäristöön oli asennettu Python 2.7.1+, jolloin Pythonia ei tarvinnut opinnäytetyössä asentaa.

3.2.2. Java Runtime Environment 1.6.0 päivitys 29

Robot Framework oppaassa (User Guide 2011) kerrottiin Jythonin käytöstä kaksi sääntöä: Robot Framework versiosta 2.5 lähtien Jythonista vaaditaan versio 2.5 ja aikaisemmat Robot Framework versiot ovat yhteensopivia Jythonin 2.2 kanssa. Näistä säännöistä johtuen valittiin asennettavaksi uusin Jython eli versio 2.5.2.

Jython 2.5.2 puolestaan vaati Java Runtime Environmentin version 1.5.0 tai uudemman (Jython 2011). Asennusympäristöön päädyttiin asentamaan uusin Java Runtime Environment eli versio 1.6.0 päivitys 29.

Javan asentaminen osoittautui erittäin haastavaksi. Ensimmäiseksi ladattiin java-paketti. Tämän jälkeen paketti siirrettiin kansioon, johon se haluttiin asentaa. Siirtämisen jälkeen törmättiin ensimmäiseen ongelmaan, kuinka paketin pystyi asentamaan. Vähän ajan kuluttua huomattiin, että tiedostolta puuttuivat ajo-oikeudet. Ajo-oikeudet lisättiin "chmod"-komennolla.

Ajo-oikeuksien lisäämisen jälkeen "java -version"-komento ei kuitenkaan löytänyt asennettua Javaa. Ratkaisu tähän ongelmaan oli kertoa järjestelmälle, että tänne kansioon on asennettu tälläinen ohjelma ja sitä voi kutsua tällä komennolla (CodeGhar 2011). Komennon " sudo update-alternatives --install" suorittaminen oikeilla parametreilla on kuvattu liitteessä 3. Komennon suorittamisen jälkeen "java -version" -komento löysi asennetun Javan.

3.2.3. Jython 2.5.2

Robot Framework 2.6.3 tarvitsi toimiakseen Jythonista version 2.5 tai uudemman. Säännöstä johtuen valittiin asennettavaksi Jython 2.5.2.

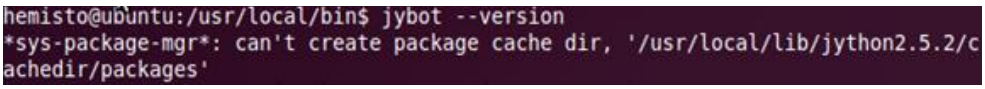
Jythonin asennuksessa ensiksi valittiin kieli, jolla Jython haluttiin asentaa. Tämän jälkeen valittiin asennettavaksi perusasennus. Viimeisenä asennus tiedusteli Java hakemistoa, tällöin tarkastettiin vain, että Java hakemisto, johon viitattiin, oli asennetun Javan kotihakemisto. (Liite 3).

Asennuksen jälkeen törmättiin ongelmaan: terminaalissa ajettava "jython --version" ei kertonut Jythonin versiota. Ensimmäiseksi kokeiltiin saman tyylistä ratkaisua kuin Javan asentamisessa, jossa järjestelmälle ilmoitettiin, mihin java on asennettu. Komennon ajamisen jälkeen "jython -version" näytti Jythonin version. Komento on kokonaisuudessaan nähtävissä liitteessä 3.

3.2.4. Robot Framework 2.6.3

Kun kaikki tarvittavat ohjelmat oli asennettu, päästiin asentamaan Robot Frameworkia. Asennus aloitettiin lataamalla Robot Frameworkin koodit Robot Frameworkin kotisivuilta.

Asennus oli todella yksinkertainen. Asennuksen aluksi Robot Frameworkin sivuilta saatu paketti purettiin. Asennuksen jälkeen kirjoitettiin komento "sudo python setup.py install" terminaaliin. Asennusta varmennettaessa törmättiin kuitenkin ongelmaan: terminaalissa ajettava komento "jybot --version" antoi virheilmoituksen (ks. kuvio 5). Ongelmaan löytyi yksinkertainen ratkaisu. Opinnäytetyötä tehdessä tiedettiin jo, että Windowsin asennuksessa Jython prosessoi Java-tiedostoja ensimmäisellä ajokerralla. Kohteeseen, johon Robot Framework asennettiin, normaalikäyttäjällä ei ollut oikeuksia asentaa mitään. Komento tuli suorittaa pääkäyttäjän oikeuksin eli eteen tuli kirjoittaa "sudo". Tämän jälkeen komento loi tarvittavat tiedostot ja "jybot --version"-komento näytti Robot Frameworkin version.



KUVIO 5. Jybotin virheilmoitus

3.2.5. Robot Framework SeleniumLibrary 2.8

Viimeisenä asennusympäristöön asennettiin Robot Framework SeleniumLibrary 2.8. Kyseistä kirjastoa käytettiin opinnäytetyössä ottamaan kuvaruutukaappauksia epäonnistuneista testeistä.

Asennus oli hyvin samankaltainen kuin Robot Frameworkin asentaminen. Ensin ladattiin paketti. Sen jälkeen se purettiin ja asennettiin. (Liite 3). Ympäristö testattiin liitteessä 2 mainitulla tavalla. Ongelmia ei esiintynyt, koska testit ajettiin samalla aineistolla kuin Windows-käyttöjärjestelmän tapauksessa.

3.2.6. Yhteenveto

Robot Frameworkin asennus Ubuntu-käyttöjärjestelmälle onnistui, koska se täytti kaikki kolme asetettua ehtoa: asennetulla Robot Frameworkilla pystyi suorittamaan sekä onnistuneen, että epäonnistuneen testin ja asennettu Robot Framework otti kuvaruutukaappauksen epäonnistuneesta testistä. Robot Framework loi myös tiedoston, josta kävi ilmi onnistuneiden ja epäonnistuneiden testien määrät.

Suurimmat Ubuntun asentamiseen liittyvät ongelmat johtuivat siitä, että Ubuntua ei osattu käyttää. Käytänteitä ei tiedetty, minkä seurauksena asennuksissa oli vaikeuksia. Tarvittavia komentoja ei tiedetty, jolloin jouduttiin etsimään, miten jokin asia tehdään.

4. ROBOT FRAMEWORKIN KÄYTTÖÖNOTTO

Opinnäytetyössä Robot Framework otettiin käyttöön sekä Atlassian Bamboo, että TeamCity-jatkuvan integraation palvelimilla, jotka oli esiasennettu toimimaan Ubuntu 11.04 -käyttöjärjestelmälle. Käyttöönotolle asetettiin samat hyväksymiskriteerit kuin käyttöjärjestelmien tapauksessa. Kriteerit olivat seuraavat:

- Testiympäristössä saaadaan ajettua yksi testi onnistuneesti ja yksi testi epäonnistuneesti.
- Epäonnistuneesta testistä otetaan automaattisesti kuvaruutukaappaus.
- Käyttäjä pystyy lukemaan koosteen, jossa kerrotaan, montako testiä ajettiin ja montako meni läpi.

Käyttöönoton dokumentointi jaettiin kahteen eri dokumenttiin. Liitteessä 4 kuvattiin käyttöönotto Atlassian Bamboo -palvelimella ja liitteessä 5 TeamCity-palvelimella. Opinnäytetyön dokumentaatiossa kerrottiin kohdatuista ongelmista ja niihin löydetyistä ratkaisuista.

4.1. Atlassian Bamboo

Opinnäytetyössä Robot Framework 2.6.3 asennettiin Atlassian Bamboo 3.4.2 käännösversiolle (*build*) 2810 Ubuntu 11.04 ympäristöön. Opinnäytetyö aloitettiin tilanteesta, jossa Atlassian Bamboo oli asennettu ja se toimi oikein.

4.1.1. Suunnitelman (plan) tekeminen

Robot Frameworkin asennus aloitettiin tekemällä Atlassian Bamboo -palvelimelle uusi suunnitelma (*plan*). Uuden suunnitelman valitsemisen jälkeen täydennettiin

tarvittavat tiedot, muun muassa suunnitelman nimi ja versionhallinnan tyyppi ja osoite.

Koska Bamboon suunnitelma (*plan*) on julkinen, päädyttiin tekemään Bamboolle oma käyttäjätunnus versionhallintaan. Tunnuksen luomisen jälkeen törmättiin ongelmiin, millä varmennus (*authentication*) valinnalla versionhallinnan sisäänkirjautuminen tulisi suorittaa. Ensimmäisenä yritettiin salasanalla kirjautumista, mikä ei onnistunut. Ongelman syyksi paljastui lopulta, että kirjautumista oltiin yritetty väärällä käyttäjätunnuksella. Käyttäjätunnuksen vaihtamisen jälkeen Bamboo sai yhteyden versionhallintaan.

Viimeisenä piti valita milloin suunnitelma (*plan*) suoritetaan. Opinnäytetyössä valittiin, että suunnitelma suoritetaan ainoastaan käyttäjän komennosta. Tämä paljastui hyväksi valinnaksi, koska muut valinnat eivät sopineet epäsäännölliseen suunnitelman suorittaminen ja testien muuttumattomuuteen.

4.1.2. Testien ajaminen

Testien ajamisessa pyrittiin hyödyntämään mahdollismman paljon jo olemassa olevaa kokemusta Robot Frameworkin asentamisesta Ubuntu-käyttöjärjestelmälle. Atlassian Bamboon eri tehtävätyyppiä tutkittaessa huomattiin, että käsikirjoitus (*script*) vastasi parhaiten Ubuntun tapausta.

Seuraavaksi pyrittiin selvittämään, pystyikö Atlassian Bamboossa säilyttämään joitain komentoja. Tämä oli tärkeä tieto, koska aina suunnitelman (*plan*) tekijä ei tiedä, mihin kansioon ohjelma on asennettu tai millä komennolla ohjelmaa saa kutsuttua. Vastaus löydettiin muuttujasta (*variable*). Muuttujia pystyi tekemään suunnitelmalle useampia ja niitä pystyi käyttämään käsikirjoituksissa (*script*). Lisää muuttujien käytöstä liitteessä 4.

Robot Framework pystyi nyt suorittamaan testit, mutta se ei osannut raportoida tuloksia. Ensimmäiseksi alettiin tutkimaan Robot Frameworkin tuottaman *Output.xml*-tiedoston sisältöä. Tarkoitus oli käyttää sovellusta, joka osaisi etsiä tiedostosta tiestien tulokset. Kyseisen tiedoston lukemiseen ei löytynyt ohjelmaa,

joten vaihtoehtona oli tehdä tämmöinen ohjelma itse. Tämä idea kuitenkin hylättiin, koska se olisi ollut liian suuritöinen urakka.

Viimein ratkaisu löydettiin Robot Frameworkin käynnistysparametrin "help" tarjoamasta listasta. Listalta löytyy komento "--xunit", jolloin Robot Framework koosti testien tulokset myös xunit-tiedostoon. Seuraavaksi tutkittiin, osaisiko Bamboo lukea xunit-tiedostoa. Bamboosta löytyi tehtävä (*task*), joka osasi lukea xunit-tiedostoa. Tehtävän (*task*) lisäämisen jälkeen, Bamboo osasi tulkita Robot Framework testien tulokset ja kertoa, moniko testitapaus suoritettiin onnistuneesti ja moniko epäonnistui.

4.1.3. Testituloksien tarkastelu

Robot Frameworkin generoimia tiedostoja Bamboo ei vielä osannut näyttää suunnitelmasivulla (*plan*). Ratkaisuksi löydettiin Bamboossa oleva artefakti (*artifact*). Artefaktin pystyi jakamaan ja tällöin se näkyi suunnitelmasivulla.

Pelkkä artefaktin (*artifact*) jakaminen ei kuitenkaan riittänyt, koska Robot Frameworkin luomassa tulossivussa on alisivuja ja alisivuilla on kuvia. Opinnäytetyössä päädyttiin tekemään kansio, johon Robot Framework tulostaa kaiken dokumentaation. Tämän jälkeen artefaktia muutettiin siten, että se koskee kaikkia juuri tässä kansiossa olevia tiedostoja. Seuraavaksi Robot Frameworkin luoman raportin pääsivun nimi muutettiin, jolloin artefaktia painamalla käyttäjälle avattiin tulossivu. Kuvattu tarkemmin liitteessä 4.

4.1.4. Yhteenveto

Robot Frameworkin asentaminen Atlassian Bamboolle oli yllättävän helppo toimenpide. Suurimmat ongelmat olivat: löytää miten Atlassian Bamboo käsitteli testituloksia ja kuinka Robot Framework pystyi tuottamaan ne. Ennen Robot Frameworkin xunit-ominaisuuden löytymistä, oltiin jo ehditty aloittamaan oman ohjelman tekeminen. Tämän ohjelman tarkoitus olisi ollut tulkita Robot Frameworkin testituloksia.

4.2. TeamCity

Opinnäytetyössä Robot Framework 2.6.3 asennettiin TeamCity Professional 6.5.6 käännösversiolle (*build*) 18130 Ubuntu 11.04 ympäristöön. Opinnäytetyö aloitettiin tilanteesta jossa TeamCity oli asennettu ja se toimi oikein. Agenttina (*Agent*), joka huolehti testien suorittamisesta, oli sama ympäristö, johon TeamCity asennettiin.

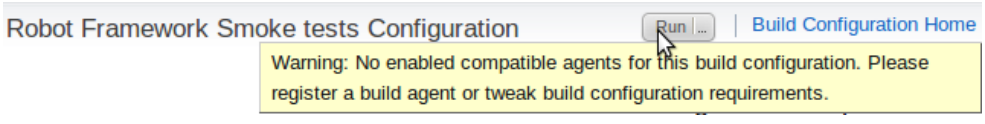
4.2.1. Projektin tekeminen

Projektin (*project*) tekeminen oli hyvin suoraviivainen. Aluksi täytettiin yleisiä tietoja, kuten projektin nimi ja kuvaus. Tämän jälkeen projektille luotiin käännösversio (*build*). Ensimmäiset asiat, jotka käännösversiossa tuli täydentää, olivat versionhallintaa koskevat asetukset.

Seuraavaksi käännösversiolle (*build*) luotiin askel (*step*), joka suoritti versionhallinnasta haetut Robot Frameworkin testit. Tämä vaihe sujui ongelmitta.

4.2.2. Testien ajaminen

Ensimmäisen ajon käynnistämisessä törmättiin ongelmiin. Järjestelmä antoi kuvion 6 mukaisen varoituksen. Varoitus johtui siitä, että TeamCityyn ei ollut liitetty yhtään mahdollista agenttia, joka olisi pystynyt suorittamaan käännösversion (*build*) ajamisen. Ongelma saatiin korjattua käynnistämällä TeamCity uudelleen komennolla "./runAll.sh start", joka käynnisti myös oletusagentin.



KUVIO 6. TeamCityn antama varoitus

Agentin käynnistämisen jälkeen TeamCity ei kuitenkaan osannut lukea testien tuloksia, jotka Robot Framework tulosti. Ongelmaan löydettiin vastaus projektin askeleelta (*step*), johon pystyi lisäämään ominaisuuden (*feature*). Ominaisuudeksi pystyi valitsemaan junit-tiedoston lukemisen. Seuraavaksi vain muutettiin Robot Frameworkin

käynnistyskomentoa siten, että se muodosti testitulokset xunit-tiedostona. Lisää komennosta liitteessä 5.

4.2.3. Testituloksien tarkastelu

Robot Frameworkin generoimien tiedostojen linkittämisessä törmättiin ongelmiin. Ensimmäiseksi päätettiin tutkia, olisiko TeamCityssä samanlainen artefakti (*artifact*) -toiminto, kuin Atlassian Bamboossa. Pienen tutkimisen jälkeen, artefakti-toiminto löytyi projektin (*project*) asetuksista. Toiminnon lisäämisen jälkeen artefaktit näytettiin erillisellä välilehdellä.

Seuraavaksi havaittiin, että TeamCity näytti turhia artefakteja (artifact) välilehdellä. TeamCityn antaman ohjeen mukaan artefakteja pystyi määrittelemään rivinvaihdolla eroteltuna. Näytettäviksi artefakteiksi valittiin Robot Frameworkin luomat "log.html", "report.html" sekä kaikki kuvat, jotka liittyivät epäonnistuneisiin testeihin. Tällöin artefakti sivulla ei näytetty turhia tiedostoja.

4.2.4. Yhteenveto

TeamCityn asennuksessa suurimmat ongelmat koskivat tarvittavan tiedon määrää. TeamCityssä oli paljon erilaisia toimintoja, joista vain murto-osaa käytettiin. Käyttämättömien toimintojen listalla oli muun muassa toiminto, jolla pystyi määrittämään, milloin testit ajettiin. Syy käyttämättömyyteen oli se, että TeamCityn asetuksien muuttaminen ei ollut tarpeen, koska hyväksymiskriteerit olivat jo täyttyneet.

Muita isoja ongelmia ei asennuksen aikana kohdattu, lukuun ottamatta agentin (agent) käynnistymiseen liittyvää ongelmaa. Ongelmien puuttuminen johtui suurimmaksi osaksi siitä, että asennuksessa pystyttiin hyödyntämään Atlassian Bamboon asennuksesta opittuja asioita. Opittuihin asioihin kuului muun muassa Robot Frameworkin xunit-tiedoston muodostaminen testituloksista.

5. Testien kirjoittaminen

Opinnäytetyössä tutkittiin, kuinka Robot Frameworkilla kirjoitettiin hyviä testejä. Hyviksi testeiksi luokiteltiin testi, jossa testissä tarvittava tieto lisättiin tietokantaan käyttäen tietokantakirjastoa. Tämän jälkeen halutut testit suoritettiin käyttäen SeleniumLibrary-kirjastoa. Testien suorittamisen jälkeen, tietokantakirjastolla poistettiin testissä lisätyt tiedot tietokannasta.

Tietokantatestien jälkeen asennettiin RIDE-editori. Asennus oli hyvin yksinkertainen, vain neljä eri komentoa ja RIDE oli asennettuna. RIDEä käytettiin tutkimaan testien muuttumattomuutta, kun ohjelmiston toteutus muuttui.

Seuraavaksi selvitettiin, kuinka testejä pystyttiin lajittelemaan kokonaisuuksiksi. Opinnäytetyössä tutkittiin Robot Frameworkin tarjoamia vaihtoehtoja jakaa testejä suoritettaviin kokonaisuuksiin.

Opinnäytetyössä ei varsinaisesti keskitytty kirjoittamaan testejä vaan tutkittiin, kuinka testejä tuli muuttaa, jos sivuston toteutus muuttuu. Tutkinnan kohteena olivat tapaukset: elementin nimi muuttui, sivuston rakenne vaihtui ja sivuston siirtäminen palvelimelta toiselle.

5.1. Tietokantakirjaston käyttöönotto

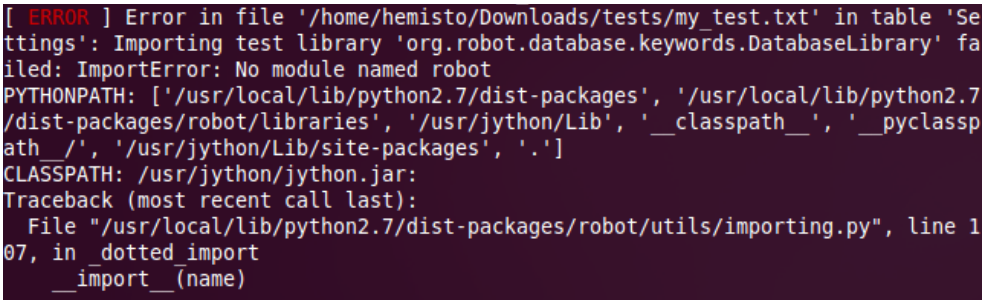
Tietokantakirjaston käyttöönotto suoritettiin samalla tyylillä kuin Robot Frameworkin käyttöönotto jatkuvan integraation palvelimilla. Asentamien on kuvattu liitteessä 6 ja opinnäytetyössä kerrotaan kohdatut ongelmat ja niihin löydetyt ratkaisut.

Robot Framework tarjosi kaksi erilaista tietokantakirjastoa: toinen näistä oli toteutettu Javalla ja toinen Pythonilla (Robot Framework 2012). Opinnäytetyössä päädyttiin käyttämään Java-pohjaista tietokantakirjastoa, koska Java-pohjaisen tietokantakirjaston ei uskottu vaativan ohjelmien asentamista testiympäristöön.

Opinnäytetyössä valittiin käytettäväksi MySQL-tietokantaa, koska se on hyvin yleinen ja ilmainen tietokanta. Tietokannan suurimmat käyttäjät ovat Wikipedia sekä Face-

book (MySQL Customers 2012). Sen lisäksi, että tietokanta on yleinen ja ilmainen, Robot Frameworkin tietokantakirjasto toimii todistetusti MySQL-tietokannalla (Jaspers 2012). Opinnäytetyön tekeminen aloitettiin tilanteesta, jossa MySQL-tietokantaan oli lisättynä testeissä käytettävä taulu. Lisää tietokannan taulusta liitteessä 6.

Tietokantakirjaston käyttöönotto aloitettiin lataamalla Robot Framework Dblibrary. Lataamisen jälkeen kirjoitettiin lyhyt testi, joka otti yhteyden tietokantaan ja varmisti, että siellä oleva "users"-taulu on olemassa. Tämän jälkeen testi sulki avatun tietokantayhteyden. Seuraavaksi testi ajettiin, mutta Robot Framework antoi virheilmoituksen (ks. kuvio 7).



KUVIO 7. Robot Frameworkin antama virheilmoitus

Ongelman kuviteltiin johtuvan siitä, että oli unohdettu ottaa käyttöön MySQL-tietokannan ajurit, jotka Jaspers ohjeisti lataamaan (Jaspers 2012). Ajurien lataaminen ei kuitenkaan auttanut, vaan edelleen tuli sama virhe. Jaspersin (2012) ohjeista löytyi maininta, että jar-tiedostojen kansio tuli lisätä Javan classpath-muuttujaan. Muuttujaan lisättiin kansio, jossa molemmat jar-tiedostot olivat, mutta edelleen tuli sama virheilmoitus.

Lopuksi päädyttiin kokeilemaan erilaista ratkaisua, jossa classpath-muuttujaan laitettiin molemmat jar-tiedostot erikseen. Yllätykseksi huomattiin, että virheilmoitus oli erilainen (ks. kuvio 8). Virheilmoitus johtui käyttäjäoikeuksista. Käyttäjällä, jolla komento ajettiin, ei ollut oikeuksia prosessoida uusia jar-tiedostoja.

```
*sys-package-mgr*: processing new jar, '/home/hemisto/Downloads/java_home/dblibr
ary-1.0.jar'
*sys-package-mgr*: can't write cache file for '/home/hemisto/Downloads/java_home
/dblibrary-1.0.jar'
*sys-package-mgr*: processing new jar, '/home/hemisto/Downloads/java_home/mysql-
connector-java-5.1.18-bin.jar'
*sys-package-mgr*: can't write cache file for '/home/hemisto/Downloads/java_home
/mysql-connector-java-5.1.18-bin.jar'
*sys-package-mgr*: can't write index file
```

KUVIO 8. Robot Frameworkin antama uudentyyppinen virheilmoitus

Seuraavaksi, ennen asentamisen jatkamista, tarkistettiin mitä Jaspers (Jaspers 2012) mainitsi jar-tiedostoista. Tarkoitus oli lähettää Jaspersille sähköpostia virheestä ja pyytää häntä korjaamaan ohjeissa oleva virhe. Tekstiä lukemalla paljastui kuitenkin, että Jaspers (mts. 2012) olikin tarkoittanut yksittäisiä jar-tiedostoja eikä jar-tiedostojen kansiota, kuten ymmärrettiin.

Kuviossa 8 mainittu ongelma saatiin korjattua kirjautumalla Terminaaliin "super user" -käyttäjän oikeuksilla, jolloin käyttäjällä riitti oikeudet prosessoida jar-tiedostot. Jar-tiedostojen prosessoinnin jälkeen tietokantakirjasto sai yhteyden tietokantaan ja testi saatiin suoritettua onnistuneesti. Lisää kirjautumisesta liitteessä 6.

5.2. Tietokantakirjaston käyttäminen

Tässä kappaleessa tarkastellaan tietokantakirjaston tärkeimpiä avainsanoja (*keyword*). Avainsanojen esimerkit, parametrit ja parametrien selitykset ovat nähtävissä liitteessä 6.

Opinnäytetyössä avainsanoihin (*keyword*) tutustuttiin tutkimalla, kuinka tietokantaan pystyi lisäämään tietoa, kuinka tieto pystyttiin hakemaan ja kuinka tiedon pystyy poistamaan.

Tietokantakirjaston käyttöön liittyi tiettyjä toimintoja, jotka oli suoritettava ennen kuin testejä pystyi kirjoittamaan. Ensimmäiseksi piti ottaa käyttöön Jaspersin tekemä tietokantakirjasto. Tietokantakirjaston sai käyttöön avainsanalla (*keyword*) "Library". Tämän jälkeen avattiin yhteys tietokantaan avainsanalla "Connect To Database".

Tietokantakirjastossa oli vain yksi avainsana (*keyword*), jolla tietokantaan pystyi lisäämään tietoa, "Execute SQL" (Jaspers 2012). "Execute SQL"-avainsanalla oli tarkoitus lisätä tieto tietokantaan ja myöhemmin poistaa se käyttäen tiedon saamaa teknistä tunnistetta (*id*). Koska "Execute SQL"-avainsana ei palauttanut mitään, piti keksiä jokin toinen keino saada tiedon tekninen numero selville. Opinnäytetyössä päädyttiin seuraavanlaiseen ratkaisuun: Ensimmäiseksi kysyttiin käyttöjärjestelmältä kellon aika. Tämän jälkeen tietokantaan lisättiin haluttu tieto, johon oli lisätty käyttöjärjestelmältä kysytty kellonaika. Seuraavaksi tietokannalta kysyttiin, mikä rivin tekninen numero on, jolta löytyy kysytty kellonaika. Lopuksi kellonaika muutettiin alkuperäiseksi tekstiksi. (Liite 6.)

Tiedon hakemiseen tietokannasta oli käytettävissä ainoastaan yksi avainsana (*keyword*), "Read Single Value From Table". (Jaspers 2012). Tällä avainsanalla saatiin haettua haluttu arvo tietokannan taulusta tietystä kentästä. Tietokantakirjasto tarjosi paljon muita avainsanoja, joilla pystyttiin varmistamaan, että tieto on tietokannassa. Esimerkiksi "Table Must Be Empty"-avainsanalla pystyttiin varmistamaan, oliko tietokannan taulu tyhjä.

Tietokantakirjastossa oli vain yksi avainsana (*keyword*), jolla pystyi tietoa poistamaan tietokannasta, "Delete All Rows From Table". Kyseinen avainsana tyhjäsi tietokannan taulun, mikä ei ollut toivottua. Opinnäytetyössä päädyttiin ratkaisuun, jossa yksittäinen tieto poistettiin tietokannasta käyttämällä "Execute SQL"-avainsanaa, sekä teknistä numeroa (*id*). (Liite 6).

Testien suorittamisen jälkeen tietokantayhteys suljettiin avainsanalla (*keyword*) "Disconnect From Database".

5.3. Testien jakaminen ajettaviin kokonaisuuksiin

Tavallisesti testien jakaminen ajettaviin kokonaisuuksiin suoritetaan jakamalla ajettavat testit eri tiedostoihin. Tällöin testien ylläpidettävyys kärsii, koska sama testi pitää kopioida useaan eri tiedostoon, jos halutaan, että sama testi suoritetaan

useissa ajettavissa kokonaisuuksissa. Tämän lisäksi, jos halutaan luoda uusi ajettava kokonaisuus, pitää luoda uusi tiedosto, johon kopioidaan kaikki testit.

Robot Framework tarjoaa hieman erilaisen lähestymistavan tavalliseen nähden. Robot Frameworkin tavassa testeille annetaan lippuja (*tags*). Lippujen avulla Robot Frameworkin käynnistämisen yhteydessä pystytään valitsemaan, mitä testejä halutaan suorittaa. (User Guide 2011.)

Robot Frameworkin käyttöohje (User Guide 2011) listaa lipuille (*tags*) neljä hyödyllistä käyttöä:

- Liput näkyvät sekä Log*.html,* että *Report.html* -tiedostoissa, antaen arvokasta lisätietoa testeistä.
- Tilastotietoa testeistä. Koska tiedot lajitellaan lipuittain, antaa lipun käyttäminen tilastotietoa, montako testiä kohdistui lipulle ja kuinka moni näistä testeistä onnistui ja epäonnistui.
- Käynnistettäessä Robot Frameworkia, käyttäjä pystyy määrittelemään parametrien ”include” ja ”exclude” avulla, mitä testejä ajossa suoritetaan.
- Lippujen avulla käyttäjä pystyy määrittelemään testien tasot, esimerkiksi mikä testi on kriittinen.

Robot Frameworkin tarjoama lippu (*tag*) toiminallisuus kuulostaa paljon paremmalta, kuin tavallinen tiedostoihin jakaminen. Liputuksessa testi on kirjoitettuna vain yhteen paikkaan. Tämän lisäksi liputuksessa testille on hyvin helppo lisätä uusia lippuja ja näin ollen jakamaan testejä paremmin ajettaviin kokonaisuuksiin.

Robot Frameworkin käyttöohjeessa (User Guide 2011) käyttäjää ohjeistetaan, että yhdessä testitiedostossa tulisi olla alle 10 testiä. Ratkaisuksi tähän Robot Frameworkin käyttöohje kertoo, että testejä voidaan jakaa kokonaisuuksiin luomalla testisarjoja (*test suite*). Testisarjaan puolestaan linkitetään testitiedostoja, joissa testit ovat. Kyseisessä puurakenteessa Robot Frameworkin tarvitsee kutsua vain testisarjaa suorittaakseen kaikki testit.

Testitiedostoja voidaan jakaa myös kansioihin (User Guide 2011). Tällöin Robot Framework suorittaa kaikki kansiosta löytyvät testitiedostot ajonaikana. Robot Framework luo kansiosta testisarjan (*test suite*), johon se liittää kaikki kansiossa olevat testitiedostot. Käyttöohjeessa (User Guide 2011) kerrotaan kuitenkin kolme rajoitetta kansioden käyttämiselle:

- Tiedostoja ja kansiota, jotka alkavat pisteellä tai alaviivalla, ei suoriteta
- Kansiot, joiden nimi on ”CVS”, ei suoriteta
- Tiedostot, joilla ei ole jokin seuraavista tiedostopäätteistä, ei suoriteta: html, xhtml, htm, tsv, txt, rst tai rest

Kansiolla voi olla erillinen alustustiedosto (*initialization file*). Alustustiedoston nimen tulee olla ”__init__.ext”, jossa ”ext” tilalle tulee jokin hieman ylempänä manituista tiedostopäätteistä. (User Guide 2011.) Alustustiedoston syntaksi on sama kuin testisarjalla (*test suite*), mutta alustustiedostossa ei voi olla suoritettavia testejä. Alustustiedostossa voidaan kuitenkin suorittaa joitain avainsanoja (*keyword*), kuten ”Suite Setup”, jossa voidaan avata tietokantayhteys, jota käytetään kansiossa olevissa testitiedostoissa.

5.4. RIDE

Seuraavaksi opinnäytetyössä asennettiin RIDE-editori, joka on suunniteltu Robot Frameworkin testien kirjoittamiseen ja suorittamiseen. RIDEstä asennettiin versio 0.42.1 Ubuntu testiympäristöön. RIDEn asennusohjeiden mukaisesti ensimmäiseksi asennettiin wxPython. WxPythonin asennusohjeessa (wxPython 2012) kerrottiin, että wxPythonin saa asennettua komennolla:

sudo apt-get install python-wxgtk2.8 python-wxtools wx2.8-i18n

Kun asennus oli valmis, ladattiin RIDEn asennuspaketti. Ensimmäiseksi asennus paketti purettiin ja sen jälkeen asennettiin. Komennot olivat:

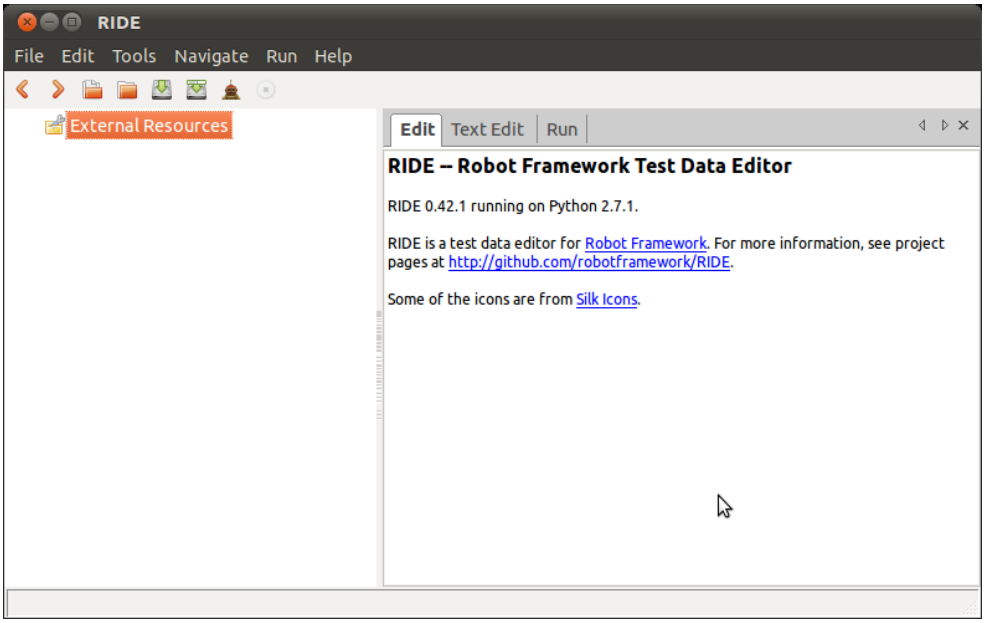
tar –zxvf robotframework-ride-0.42.1.tar.gz
sudo python setup.py install

Asennuksen jälkeen RIDEn sai käyntiin komennolla ”*ride.py*”. Kun RIDE käynnistyi, näytettiin käyttäjälle kuvion 9 mukainen näyttö.

Näytön yläriviltä löytyivät valikot: *File, Edit, Tools, Navigate, Run* ja *Help*. Näiden valikoiden alta löytyi nimeä kuvaavia toiminnallisuuksia, kuten *File*-valikon alta löytyy muun muassa toiminnallisuudet luoda uusi projekti sekä avata olemassa oleva projekti.

Tämän alapuolella oli palkki, johon oli sijoitettu joitain valikoista löytyviä ominaisuuksia. Palkissa oli seuraavat toiminnallisuudet lueteltuna vasemmalta oikealle:

- *Go Back* -komennolla käyttäjä pääsi takaisin edelliseen kohtaan. Toiminta samanlainen kuin selaimen *Back*-painikeella.
- *Go Forward* -komennolla käyttäjä siirryttiin seuraavaan kohtaan. Toiminta samanlainen kuin selaimen *Forward*-painikeella.
- *Open*-komennolla käyttäjä pystyi avaamaan testitiedoston.
- *Open Directory* -komennolla käyttäjä pystyi avaamaan kansion, jolloin kaikki kansiossa olevat testitiedostot avattiin.
- *Save*-komennolla tallennettiin avattu tiedosto.
- *Save All* -komennolla tallennettiin kaikki avatut tiedostot.
- *Run Test Suite* -komennolla suoritettiin testien ajaminen.
- *Stop Running* -komennolla pystyttiin keskeyttämään suorituksessa oleva testi.

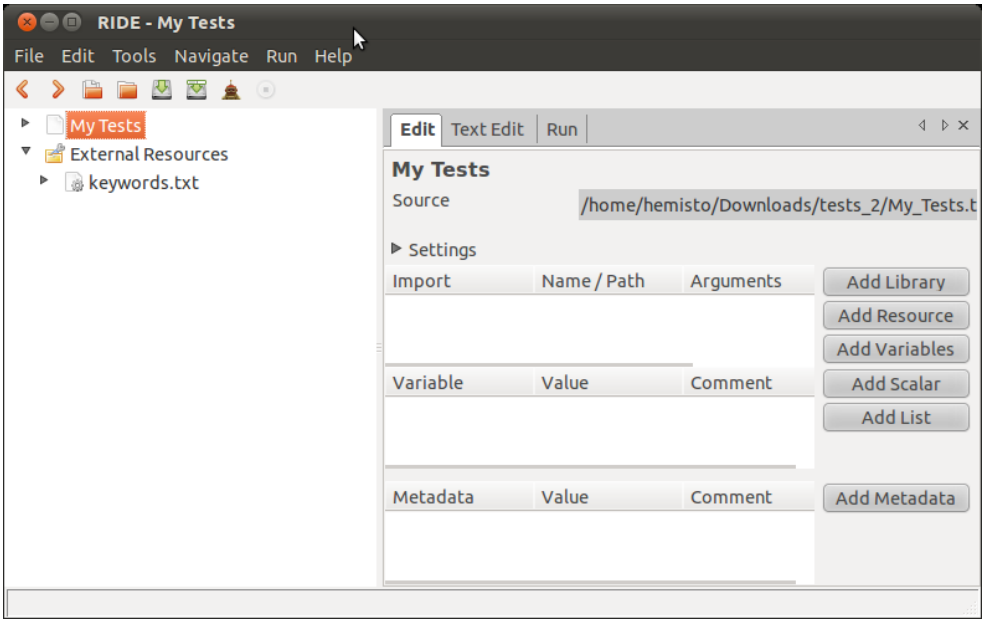


KUVIO 9. RIDEn ensimmäinen käynnistys

Uuden projektin luomisen jälkeen avautui näyttö, jossa näytettiin testitiedosto ja avainsanatiedosto (ks. kuvio 10). *Edit*-välilehdellä näytettiin toiminnallisuudet kirjaston, resurssitiedoston ja muuttujatiedoston lisäämiseen. Näiden alapuolella näytettiin toiminnallisuudet muuttujien lisäämiseen. Muuttuja lisättiin testitiedostoon eikä erilliseen muuttujatiedostoon. Viimeisenä testitiedostolle pystyttiin antamaan metatietoja. Metatietoa lisättäessä annettiin nimi, arvo ja kommentti.

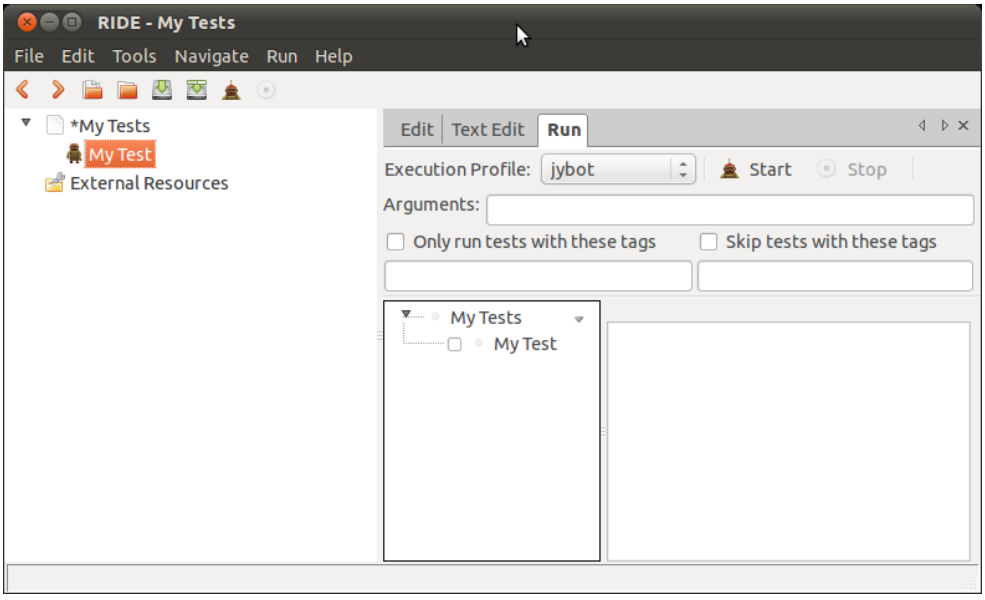
Käyttäjän painaessa *Settings*-tekstiä, avattiin valikko, johon pystyttiin antamaan lisää testitiedostoa koskevia arvoja (ks. kuvio 10). Valikosta pystyi muun muassa kommentoimaan testitiedostoa ja määrittelemään *Suite Setup* ja *Suite Teardown* -avainsanat.

RIDEssä oli tämän lisäksi kolme välilehteä: *Edit, Text Edit* ja *Run*. *Edit*-välilehti oli normaali näkymä, jossa käyttäjä pystyi valikoiden avulla luomaan testin. *Text Edit* -välilehti puolestaan näytti testitiedoston tekstimuodossa. Tässä näkymässä testiä pystyi kirjoittamaan samalla lailla, kuin normaalissa tekstieditorissa.



KUVIO 10. Uusi projekti luotu

Run-välilehdellä oli toiminnallisuudet testien suorittamiseksi (ks. kuvio 11). Välilehdeltä pystyi valitsemaan ajettiinko testien käyttäen *jybot*- vai *pybot*-komentoa. Tämän lisäksi käyttäjä pystyi antamaan Robot Frameworkille käynnistysparametreja ja listaamaan liput (*tags*), jotka suoritettiin tai jätettiin suorittamatta.



KUVIO 11. Testien suorittaminen

5.5. Testien muuttamisen tarve

RIDen asentamisen jälkeen opinnäytetyössä tutkittiin, miten testejä tulee muuttaa, jos sivuston toteutus muuttuu. Sivustoksi, jossa testit suoritettiin, valittiin opinnäytetyön aikana tehty testisivusto. Tällöin sivuston toteutusta pystyttiin helposti muuttamaan ja käytännössä testaamaan, kuinka testejä tuli muokata.

Selenium-kirjaston avainsanat käyttävät elementteihin viittaamiseen paikanninta (*locator*). Paikantimia oli 10: *id*, *name*, *value*, *href*, *src*, *alt*, css, link *text*, *dom* ja *xpath*. (SeleniumLibrary documentation 2012.) Opinnäytetyössä tutkittiin testien muuttamisen tarvetta, kun käytössä oli *id*-paikannin.

Id-paikannin valittiin, koska elementtiin pystyi viittaamaan ainakin *id*-paikantimen avulla. Muilla viittauksilla oli joitain rajoitteita. Esimerkiksi avainsanalle *Checkbox Should Be Selected* viittausmahdollisuuksiksi listattiin *Id* ja *Name*, kun taas *Click Button* -avainsanalle mahdollisuudet olivat *Id, Name* ja *Value*. (Mts. 2012.)

5.5.1. Elementtien uudelleen nimeäminen

Elementtien uudelleen nimeämisellä tarkoitetaan elementin *id*-arvon vaihtamista. Välillä sivuston kehityksessä voidaan joutua tilanteeseen, jossa elementtien nimiä joudutaan vaihtamaan kuvaavammiksi. *Id*-arvon vaihtamisella toteutuksessa oli katastrofaaliset vaikutukset testeihin. Kaikki testit epäonnistuivat, koska elementtiä ei löytynyt.

Vaihtoehtoina olivat seuraavat: viitata jollain muulla paikantimella tai minimoida testien muokkauksen määrä. *Xpath* tai *dom*-paikantimet viittaavat elementtiin sivuston rakenteen avulla. Rakenteella viittaaminen ei ota kantaa elementtien nimiin, mutta on hyvin tarkka sivuston rakenteen muutokselle. Rakenteen muutosherkkyyden takia opinnäytetyössä päädyttiin tutkimaan ratkaisua, jossa minimoidaan tarvittavien muutoksien tarve kun käytetään *id*-paikannusta.

Opinnäytetyössä keksittiin ratkaisu, jossa html-elementtien viittauksen kohteet laitettiin erilliseen resurssi tiedostoon, josta niitä kutsuttiin *${<resurssin_nimi>}* -

komennolla. Tämä kaikki joudutaan tekemään käsin, koska RIDE ei tarjoa tähän mitään aputoiminnallisuutta. Tämä vie hiukan enemmän aikaa, mutta jos samaa elementtiä on käytetty useassa kymmenessä testissä, tarvitsee muutos tehdä vain yhteen tiedostoon usean kymmenen sijasta.

5.5.2. Rakenteen muuttaminen

Rakenteen muuttumisella tarkoitetaan elementtien siirtämistä sivustolla paikasta toiseen. Tällä haluttiin testata pitääkö testejä muuttaa, jos sivuston rakennetta muutetaan, mutta sisältö pysyy samana. Koska *id*-arvo on elementit erottava yksilöllinen arvo, ei sivuston ulkoasun muuttumisella ollut mitään vaikutusta testien suoritettavuuteen.

5.5.3. Sivuston uudelleen sijoittaminen

Sivuston uudelleen sijoittamisella tarkoitetaan tapausta, jossa koko sivusto siirretään palvelimelta toiselle. Uudelleen sijoittamisessa ei ollut ongelmaa, jos linkitys oli tehty *id*-paikantimella. Ongelma esiintyi tilanteessa, jossa linkkiin viitattiin linkin koko osoitteella, esimerkiksi ”*href=http://domain.fi/news*”.

Ratkaisuja kyseiseen ongelmaan oli kaksi. Ensimmäinen oli käyttää jotain muuta paikanninta esimerkiksi *id*-paikanninta. Aina kuitenkaan *id*-arvoa ei ole linkille saatavilla, jolloin helpointa on käyttää *href*-viittausta. Toinen vaihtoehto oli muokata *Href*-paikanninta ylläpidettävämpään muotoon käyttäen muuttujaa. Muuttujaan tallennetaan palvelimen osoite ja tähän viitataan jokaisen linkin osoitteessa, esimerkiksi ”*href=${domain}/news*”. Tällöin, jos palvelinta vaihdetaan, joudutaan palvelimen osoite muuttamaan vain yhteen paikkaan.

6. Käytettävyys

Käytettävyydessä tutkittiin Robot Frameworkin käytettävyyttä RIDE-editorilla, joka on suunniteltu Robot Frameworkin testien kirjoittamiseen ja suorittamiseen (RIDE 2012). Käytettävyyttä arvioitiin seuraavilla 11 kriteerillä: tehokkuus, opittavuus, muistettavuus, muistettavien asioiden määrä, hallittavuus, opastus, miellyttävyys,

virheiden sieto, virheettömyys, tehtävään sopivuus ja johdonmukaisuus. Käytettävyydessä arvioitiin Robot Frameworkin testien kirjoittamista ja suorittamista RIDE-editorilla. Löydetyt käytettävyys ongelmat arvioitiin kolmi-portaisella asteikolla:

1. Pieni kosmeettinen ongelma tai ei käyttöä haittaava ongelma
2. Ongelma, joka haittaa käyttöä
3. Ongelma, joka estää ohjelman käytön

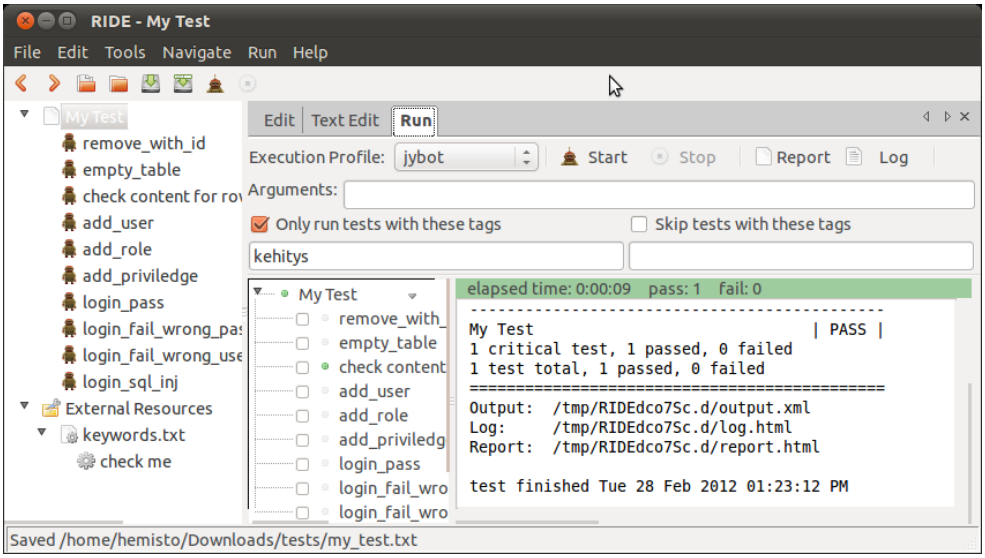
6.1. Tehokkuus

Robot Frameworkin käyttö RIDEllä oli todella tehokasta. Aluksi testien kirjoittaminen kangerteli, koska ei tiedetty kuinka RIDE toimii ja kuinka sitä tulisi käyttää. Kun RIDEn toiminnallisuudet alkoivat hahmottua, nopeutui testien kirjoittaminen huomattavasti.

Testien suorittaminen oli varsin nopeaa, mutta kun testejä oli 10, alkoi testien ajamiseen kuluva aika olla hidaste. Ongelmaan oli kaksi ratkaisua: ensimmäinen vaihtoehto oli käyttää Robot Frameworkin tarjoamaa lippu (*tag*) toiminallisuutta. Toinen vaihtoehto oli käyttää RIDEssä olevaa toiminnallisuutta, jossa ajettavat testit valittiin valintalistasta.

Liputuksessa testille annettiin kehityksen aikana jokin lippu (*tag*). Kun testejä ajettiin, ajettiin vain kaikki kyseisen lipun testit (ks. kuvio 12). Kun testi saatiin kirjoitettua valmiiksi, lippu poistettiin.

RIDEssä oli myös toiminnallisuus, jolla pystyi valitsemaan ”Run”-välilehdeltä testit, jotka suoritettiin. Testi valittiin suoritukseen valitsemalla neliö testin nimen vasemmalta puolelta (ks. Kuvio 12). Vaihtoehdon heikkoutena oli se, että asetukset eivät näkyneet muille ohjelmille. Liputuksessa puolestaan testin pystyi antamaan esimerkiksi Atlassian Bamboolle suoritettavaksi. Atlassian Bamboossa muokattiin Robot Frameworkin käynnistysparametreja siten, että se suoritti vain ”kehitys”-lipun omistavat testit.



KUVIO 12. Kehitteillä olevan testin ajaminen ja tuloksen tarkastelu

6.2. Opittavuus

Robot Framework opetti käyttäjää RIDEn käytössä todella hyvin. Hyvin nopeasti testejä kirjoitettaessa opittiin hahmottamaan testin muoto, testisarjan (*test suite*) muoto ja avainsanojen käyttäminen. Hetken testejä kirjoitettua avainsanojen nimen pystyi jo melkein arvaamaan, jos tiesi sanan käyttökohteen. Myös pikanäppäimet olivat yksinkertaisia ja helppo oppia, kuten F8-painikkeella sai ajettua testit.

6.3. Muistettavuus ja muistettavien asioiden määrä

RIDE auttoi käyttäjää vähentämään tarvittavien asioiden muistamisessa. Testejä kirjoitettaessa, käyttäjän apuna oli avainsanojen täyttäminen (*keyword completion*). Kun testiin oli kirjoitettu avainsanan alkua ja painettiin kontrollia ja välilyöntiä yhtä aikaa, järjestelmä näytti käyttäjälle alkuun sopivat avainsanat. Robot Framework olisi saanut arvosanaksi kiitettävän, mutta avainsanojen täyttäminen ei toiminut jarpäätteisellä tietokantakirjastolla. Tästä johtuen avainsanoja jouduttiin opettelemaan ulkoa jonkin verran. Ongelma luokiteltiin tasoon kaksi.

6.4. Hallittavuus

Hallittavuus oli myös todella hyvällä tasolla Robot Frameworkissa. Robot Frameworkissa oli liputus (*tag*) toiminnallisuus, joilla testejä pystyttiin jakamaan ajettaviin ko-

konaisuuksiin hyvinkin helposti ja vaivattomasti. Myös ajettavien kokonaisuuksien valitseminen suoritettavaksi oli helppoa. RIDEssä valittiin, että suoritetaan kaikki testit, joilla on ”kehitys”-lippu (ks. kuvio 12).

RIDE osasi myös erotella yhdessä tiedostossa olevat testit erillisiksi osioiksi. Tämä erottelu lisäsi entisestään hallittavuutta, koska haluttu testi oli paljon helpompi löytää valikosta, kuin tiedostoa selaamalla. Kuviossa 12 näkyy, kuinka testit erottuvat avatusta tiedostosta, jossa on 10 testiä.

6.5. Opastus

Opastus kangerteli hiukan Robot Frameworkissa. Testejä suoritettaessa Robot Framework kertoi kyllä käyttäjälle askel askeleelta mitä suoritettiin ja testin epäonnistuttua antoi Robot Framework kuvaavan virheilmoituksen. Osassa tapauksista kuitenkin virheilmoitus ei kuvannut mikä meni vikaan ja ongelmaa etsittiin suhteellisen kauan. Esimerkiksi Selenium-kirjastolla avattaessa yhteyttä Selenium Grid -palvelimelle saatiin virheilmoitukseksi *Error 500 java.lang.NullPointerException* ja tämän jälkeen tulostui terminaaliin html-elementtejä. Virheilmoituksen kuvauksen täsmällisyys ei estänyt RIDEn käyttöä ja siksi se oli 1-tason käytettävyysongelma.

Testejä kirjoitettaessa opastus toimi huomattavasti paremmin. Avainsanan (*keyword*) kirjoittamisen jälkeen RIDE näytti mitkä parametrit olivat pakollisia. Kun hiiren vei parametrien päälle, näytti RIDE parametrin selityksen.

6.6. Miellyttävyys

RIDEä oli todella miellyttävä käyttää neutraalien värien johdosta. Vaalean harmaa oli hyvä taustaväri, koska ei rasittanut käyttäjän silmiä. Myös kaikki ikonit olivat selkeitä ja erottuivat selvästi taustasta. Ikonit eivät kuitenkaan erottuneet liikaa, jolloin käyttäjälle olisi jäänyt epämiellyttävä olo (ks. kuvio 12).

Kun Robot Framework suoritti testejä, käyttäjä pystyi miellyttävästi seuraamaan testien suoritusta. Robot Framework tulosti askel askeleelta mitä se suoritti ja mikä oli suorituksen tulos.

RIDE työkalusta puuttui kokonaan vieritys (*scroll*) toiminnallisuus. Jos käyttäjän RIDE ikkuna on pieni, eivät kaikki testitiedoston *Settings*-valikon toiminnallisuudet mahdu samalle näytölle. Tällöin vieritystoiminnallisuus olisi välttämätön, jotta käyttäjä pääsee vaihtamaan ihan viimeisenä olevia arvoja. Koska käyttäjä ei pysty muuttamaan viimeisimpiä asetuksia, oli ongelma kolmannen tason käytettävyysongelma.

6.7. Virheiden sieto ja virheettömyys

RIDE sieti virheitä hyvin. Opinnäytetyön aikana ei RIDEä saatu tilanteeseen, jossa ohjelman suoritus olisi jouduttu lopettamaan kesken ja aloittamaan alusta.

Avainsanojen (*keyword*) kohdalla virheilmoituksia saatiin helposti aikaan. Testejä kirjoitettaessa RIDE ei osannut tarkistaa oliko avainsanan parametrit oikeassa muodossa. Kun ajettiin parametrin ollessa väärässä muodossa, testi epäonnistui ja RIDE ilmoitti virheestä. Parametrien oikeellisuuden tarkistamisen puuttuminen testiä kirjoitettaessa luokiteltiin ykköstason käytettävyysongelmaksi.

6.8. Tehtävään sopivuus

Robot Framework soveltuu hyvin automaatiotestaukseen. Opinnäytetyössä huomattiin, että virheilmoituksien ja tuloksien laatu on hyvin tärkeää. Virheilmoitukset auttavat käyttäjää huomaamaan mikä aiheutti virheen ja korjaamaan testissä olevan virheen tai kertomaan kehittäjälle sivustolla olevasta virheestä.

Tuloksista nähdään, mitä järjestelmä teki ennen virhettä ja mistä virhe aiheutui. Nämä auttavat käyttäjää hahmottamaan, miksi järjestelmä käyttäytyi juuri tässä tilanteessa näin. Tämän lisäksi Selenium-kirjasto tarjosi ominaisuuden, jolla pystyi ottamaan kuvaruutukaappauksen virheestä. Tämä auttoi entisestään virheen hahmottamisessa.

6.9. Johdonmukaisuus

Robot Framework oli hyvin johdonmukainen. Erityisesti avainsanojen (*keyword*) nimet olivat hyvin arvattavissa. Esimerkiksi *Should be Equal* todellakin tarkasti oliko kaksi seuraavaa arvoa samat.

6.10. Yhteenveto

Robot Frameworkin käytettävyyden arvioinnissa järjestelmästä löytyi viisi ongelmaa: kolme 1-tason ongelmaa, yksi 2-tason ongelma ja yksi 3-tason ongelma. 1-tason ongelmat eivät haitanneet käyttöä.

2-tason ongelma puolestaan haittasi käyttöä jo merkittävästi. Ongelma tuli esille, kun RIDEssä yritettiin ottaa käyttöön jar-tiedostona olevaa tietokantakirjastoa. Kun testejä kirjoitettiin, RIDE ei näyttänyt tietokantakirjastossa olevia avainsanoja ollenkaan, vaan avainsanat jouduttiin katsomaan tietokantakirjaston dokumentaatiosta. Tietokantakirjasto toimi kyllä testejä ajettaessa.

3-tason ongelma esti ohjelman käyttämisen. Ilman vieritys (*scroll*) toimintoa, kun RIDEn ikkuna oli normaalikokoinen, eivät kaikki testitiedoston *Settings*-osion asetukset näkyneet ruudulla. Ongelma saatiin korjattu kasvattamalla ikkunan kokoa, mutta kaikilla käyttäjällä tämä ei ole mahdollista ja sen takia ongelma tulisi korjata nopeasti.

7. Uudelleen käytettävyys

Opinnäytetyössä tutkittiin Robot Frameworkilla kirjoitettujen testien uudelleenkäytön mahdollisuutta.

7.1. Avainsanat

Robot Framework tarjosi aivan uudenlaisen uudelleenkäytön avainsanojen (*keyword*) avulla (User Guide 2011). Avainsana kirjoitettiin sille kuuluvaan kohtaan testitiedostoa ja tämän jälkeen sitä pystyttiin kutsumaan koko tiedostosta. Avainsanan sisällä pystyttiin kutsumaan muita avainsanoja tai kirjastoissa olevia avainsanoja.

Robot Frameworkissa pystyi myös kirjoittamaan avainsanatiedoston, johon kirjoitettiin kaikki yleisessä käytössä olevat avainsanat. Tämän jälkeen avainsanatiedosto otettiin mukaan testitiedostoon ”Settings”-osiossa. Tiedosto otettiin käyttöön avainsanalla ”Resource” ja tämän jälkeen kirjoitettiin tiedoston nimi.

Resurssitiedoston (*resource file*) muoto oli samanlainen kuin testitiedoston, mutta resurssitiedostossa ei saanut olla ”Test Case” osiota. Resurssitiedoston ”Settings”-osiossa voitiin tuoda (*import*) toisia kirjastoja, resurssitiedostoja tai muuttujatiedostoja (*variable file*), mutta esimerkiksi sarjan alustusta (*suite setup*) ei voitu suorittaa. (User Guide 2011.)

7.2. Kirjastot

Robot Framework tarjosi avainsanojen lisäksi mahdollisuuden kirjoittaa kirjaston (User Guide 2011). Kirjastoon pystyttiin kirjoittamaan avainsanojen (*keyword*) avulla testejä, joihin vietiin parametrien avulla testissä käytettävät tiedot. Esimerkiksi web-sivulla olevaan sisään kirjautumiseen voitiin tehdä kirjastossa oleva testi. Testille vietiin parametreilla käyttäjätunnus ja salasana. Testi käytti suorittamiseen Selenium-kirjastoa ja lopuksi kirjasto palautti testin tulokset kutsujalle.

Testikirjasto voidaan kirjoittaa Pythonilla tai Javalla (User Guide 2011). Pythonilla kirjoitettu kirjasto voidaan suorittaa sekä Pythonilla, että Jythonilla, mutta Javalla kirjoitettu kirjasto voidaan suorittaa ainoastaan Jythonilla.

Robot Frameworkilla on kolme eri ohjelmointirajapintaa (*application programming interface*), staattinen-, dynaaminen- sekä hybridi ohjelmointirajapinta. Staattinen ohjelmointirajapinta on helpoin lähestymistapa moduulin tai luokan tekemiseksi, koska metodin (*method*) nimi vastaa suoraan avainsanan (*keyword*) nimeä. Myös avainsanalla on samat parametri, kuin metodilla. (User Guide 2011.)

Dynaamiset kirjastot ovat luokkia, jotka toteuttavat metodin (*method*), joka hakee käytettävän avainsanan (*keyword*) nimen. Tämän lisäksi dynaamisella kirjastolla on metodi, joka suorittaa kyseisen avainsanan annetuilla parametreilla. Avainsanan nimi, sekä tapa jolla avainsana suoritetaan, voi vaihdella ajonaikana. Tilan raportointi, loki sekä palautus arvot toimivat kuten staattisessa ohjelmointirajapinnassa. (User Guide 2011.)

Hybridi mallissa kirjastot ovat luokkia, jossa metodit (*method*) kertovat minkä avainsanan (*keyword*) se toteuttaa. Avainsanan on oltava kuitenkin suoraan saatavilla. Muu toiminnallisuus on kuten staattisessa ohjelmointirajapinnassa. (User guide 2011.)

7.3. Yhteenveto

Uudelleenkäyttöä tutkittiin vain hiukan ja siitä löytyi kaksi hienoa ominaisuutta, avainsanat (*keyword*) ja kirjastot. Luotuja avainsanoja pystyttiin kutsumaan useista eri testeistä testitiedostosta. Kirjastoilla pystyttiin luomaan suurempia kokonaisuuksia, joita voitiin käyttää useista testiprojekteista.

8. Johtopäätökset

Robot Frameworkin käyttöönotto testauksen automatisoinnin työkaluna sujui suhteellisen vaivattomasti. Ensimmäiseksi käyttäjän tarvitsi luoda ympäristöt testien kehittämiselle ja testien suorittamiselle. Näissä vaiheissa apuna toimivat opinnäytetyössä luodut asennusohjeet Windows- ja Ubuntu -käyttöjärjestelmille sekä käyttöönotto-ohjeet Atlassian Bamboo- ja TeamCity -jatkuvan integraation ympäristöihin. Ympäristöjen pystyttäminen onnistuu yhdessä päivässä, vaikka ongelmia esiintyisikin hiukan asennuksen aikana.

Testien kirjoittamiseen tarkoitettu RIDE-editori helpotti testien kirjoittamista. Yhdessä tiedostossa olevien testien jakaminen testeittäin auttaa käyttäjää hahmottamaan mitä testejä tiedostossa on. Jos RIDEä ei käytetä, joutuu käyttäjä selailemaan tekstieditorilla tiedostoa ylös ja alas hahmottaakseen, mitä testejä tiedostossa on. Tämän lisäksi RIDEssä on avainsanojen listaamiseen tarkoitettu toiminnallisuus, joka tulee esille, kun käyttäjä painaa *ctrl*-painiketta ja välilyöntiä. Molemmat toiminnallisuudet nopeuttavat ja helpottavat käyttäjän työskentelyä Robot Frameworkin parissa.

RIDE-editorista löydettiin muutama käyttöä haittaava ongelma. Vakavimpana ja ohjelman käytön estävänä ongelmana oli vierityksen (*scroll*) puuttuminen. Kun välilehdellä olevan valikko avattiin, kasvoi välilehden pituus, mutta oikeaan reunaan ei tul-

lut vierityspalkkia, jolloin osa valikoista ei tullut näkyviin. Muut ongelmat eivät haitanneet RIDEn käyttöä merkittävästi.

Testien uudelleenkäytön pystyi toteuttamaan Robot Frameworkissa avainsanojen ja kirjastojen avulla. Käyttäjä voi halutessaan luoda testistä avainsanan, jolloin avainsanaa voidaan kutsua muista testeistä. Tällöin myöhemmin tehtävä muutos tarvitsee tehdä vain luotuun avainsanaan.

9. Jatkotutkimus

Opinnäytetyön rajauksista johtuen kaikkia asioita ei voitu tutkia tarvittavan tarkalla tasolla. Tähän kappaleeseen on kerätty näistä tärkeimmät alueet, joita kannattaisi tutkia tarkemmin

9.1. Testien kirjoittaminen RIDE-työkalulla

Opinnäytetyössä käytettiin RIDEä testien kirjoittamiseen, mutta RIDEä kannattaisi ehdottomasti tutkia lisää. Tutkinnassa luotaisiin laajempi testiympäristö, jota tutkinnassa kirjoitetut testit testaisivat. Tutkinnasta luotaisiin aloittajan paketti (*starter kit*), jossa olisi testiympäristö, iso määrä esimerkkitestejä ja ohjeet testien kirjoittamiseen.

9.2. Selenium Grid

Opinnäytetyössä käytettiin Selenium Grid ympäristöä suorittamaan Atlassian Bamboon ja TeamCityn web-sivujen testejä, mutta asiaan ei perehdytty kunnolla.

Selenium Grid on ympäristö, jossa yksi Selenium Server toimii käskyttäjänä x-määrälle muita Selenium Servereitä, jotka suorittavat testejä. Serverille voidaan sanoa mitä selaimia, miten monta ja missä ympäristössä selaimet ovat. Vastaavasti, kun testejä suoritetaan, voidaan määritellä missä ympäristössä ja millä selaimella testit halutaan suorittaa. Lisää osoitteessa
http://code.google.com/p/selenium/wiki/Grid2

9.3. Atlassian Bamboo ja TeamCity

Opinnäytetyössä Robot Framework otettiin käyttöön Atlassian Bamboo ja TeamCity -jatkuvan integraation ympäristöissä, mutta ympäristöjen toiminnallisuutta ei tutkittu. Tutkinnassa perehdyttäisiin jatkuvan integraation ympäristön toiminnallisuuteen ja mahdollisuuksiin.

9.4. Kolmannen osapuolen kirjastot

Opinnäytetyössä käytettiin kolmannen osapuolen tietokanta kirjastoa, mutta Robot Frameworkiin on saatavilla myös muita kirjastoja. Tutkinnassa tarkasteltaisiin muita tehtyjä kirjastoja ja mitä kirjastoja ollaan tekemässä. Tämän lisäksi kirjoitetaan ohjeet oman kirjaston luomiseen.

LÄHTEET

Atlassian Bamboo. n.d. Try Bamboo. Viitattu 12.1.2012.
http://www.atlassian.com/software/bamboo/try/

CodeGhar. n.d. Update Alternatives in Debian. Viitattu 18.12.2011.
http://codeghar.wordpress.com/2009/01/27/update-alternatives-in-debian/

Digian vuosikertomus. n.d. Digian vuoden 2011 vuosikertomus. Viitattu 12.03.2012.
http://vuosikertomus2011.digia.com/digia-nyt-ja-tulevaisuudessa

Eronen, H. 2010. Uudelleenkäytettävyys eräässä ohjelmisto projektissa. Viitattu 5.1.2012. http://www.doria.fi/bitstream/handle/10024/69179/nbnfi-fe201103181363.pdf?sequence=3

Haikala, I. & Märijärvi, J. 2006. Ohjelmistotuotanto. Helsinki: Talentum.

Jaspers, T. 2012. Robot Framework dblibrary. Viitattu 27.01.2012.
https://github.com/ThomasJaspers/robotframework-dblibrary/wiki

Jython. n.d. DownloadInstruction. 23.03.2011.
http://wiki.python.org/jython/DownloadInstructions

Käytettävyyden perusteet. 2007. Tampereen teknillinen yliopisto. Viitattu 26.1.2012.
http://www.cs.tut.fi/~kaper/syksy07/luennot/S07L1.pdf

MySQL Customers. n.d. Viitattu 12.02.2012. http://www.mysql.com/customers/

Pohjolainen, P. 2003. Ohjelmiston testauksen automatisointi. Viitattu 29.12.2011.
http://cs.uef.fi/uku/tutkimus/Teho/PenttiPohjolainen_Gradu.pdf

RIDE. 2012. Robot Framework test data editor. Viitattu 25.2.2012.
https://github.com/robotframework/RIDE/wiki

Riihiaho, S. n.d. Käytettävyyden arviointi ilman käyttäjiä. Viitattu 6.1.2012.
http://www.soberit.hut.fi/T-121/T-121.600/asiantuntija-arviot.pdf

Robot Framework. n.d. A generic test automation framework. Viitattu 22.10.2011.
http://code.google.com/p/robotframework/

SeleniumLibrary documentation. n.d. Selenium kirjaston dokumentaatio. Viitattu 24.2.2012. http://robotframework-seleniumlibrary.googlecode.com/hg/doc/SeleniumLibrary.html?r=2.8

SourgeForge. n.d. Python for Windows extensions. Viitattu 17.12.2011. http://sourceforge.net/tracker/index.php?func=detail&aid=3402824&group_id=78018&atid=551954

Usabilitybok. n.d. Heuristic Evaluation. Viitattu 1.1.2012. http://www.usabilitybok.org/methods/p275?section=how-to

UsabilityFirst. n.d. About usability. Viitattu 1.1.2012. http://www.usabilityfirst.com/about-usability/introduction-to-user-centered-design/

User Guide. n.d. Robot Framework user guide. Viitattu 6.11.2011. http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html?r=2.6.2

Why automated testing? n.d. TestComplete technical papers. Viitattu 25.10.2011. http://support.smartbear.com/articles/testcomplete/manager-overview/

wxPython. n.d. Installing wxWidgets and wxPython on Ubuntu or Debian. Viitattu 2.3.2012. http://wiki.wxpython.org/InstallingOnUbuntuOrDebian

LIITTEET

Liite 1. Install Robot Framework. 32-bit Windows 7

Install Robot Framework

32-bit Windows 7

Contents

About this guide

In this guide you will install Python 2.7.2, Java Runtime Environment 1.6.0 update 29, Jython 2.5.2, Robot Framework 2.6.3 and Robot Framework Selenium library 2.8 to Windows 7 32-bit operating system. Robot Framework needs Python and Jython to work and Jython needs Java Runtime Environment to work. Selenium library is extension library to Robot Framework that can be used to test web applications.

Robot Framework runs on Python and Jython, you will need to have at least one of them installed. Robot Framework recommends that you will always install Python. Starting from Robot Framework 2.5 supports Python 2.5 or newer. Earlier versions are compatible with Python 2.3 or newer. Robot Framework is not compatible with Python 3.x. (Preconditions.)

You will have to install Python and Robot Framework to get started. You will have to install Java and Jython if you want to use 3rd party libraries that are written using Java. You will also have to install Selenium library if you want to test web applications.

After you have completed this tutorial you have an environment where you can create and run Robot Framework tests. Installation is complete when you are able to:

- Run simple test, where 1 will fail and 1 will pass
- Robot Framework takes a screenshot from failed test case
- Read a report that tells you how many tests succeeded and how many failed

It is highly recommended that you will run all installations as Administrator. If you don't run it as Administrator, installation might not be able to finish configuration and program might not work.

Install Python

Robot Framework 2.6.3 requires at least Python 2.5, but not Python 3. First you will have to check if your system already has Python installed. Start Command Prompt and type

```
python --version
```

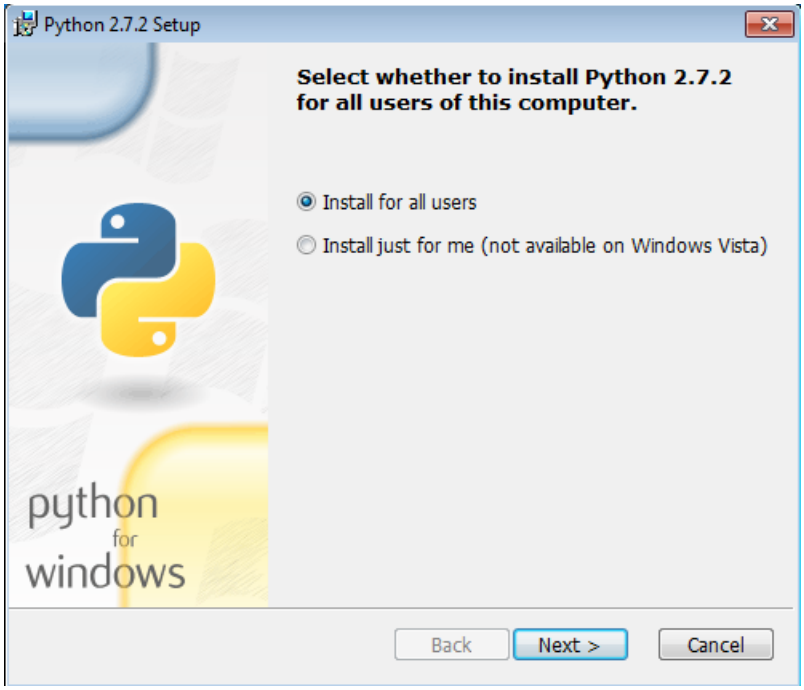
If you have Python installed, your system will print something like

```
C:\Users\hemisto>python --version
Python 2.7.2
```

If you have Python installed and its version is at least 2.5 but not 3, you can skip Python installation and move on to installing Java Runtime Environment.

Python installation is divided into 3 steps: download, install and verify. Python 2.7.2 can be downloaded from Python homepage (http://www.python.org/). When you have downloaded Windows installation package double click it to start installation:

1. Robot Framework user guide suggest that you will install Python to all users. Select "install for all users" and click Next.



2. Choose directory where you want to install Python and click Next. I highly recommend that you won't use whitespaces in folder names, those can lead you into trouble. Use "C:\python2.7" for example.
3. Next you can customize your Python. You can just click Next because default values are okay for you.

4. Click Finish.

5. When installation is done, append system Environmental variable "*Path*"

```
;<path_to_python_folder>
```

6. Verify installation. Start Command Prompt and type

```
python --version
```

System should print Python version like shown below.

```
C:\Users\hemisto>python --version
Python 2.7.2
```

Install Java Runtime Environment

Robot Framework 2.6.3 requires at least Jython 2.5. Jython 2.5.2 you will install next requires at least Java Runtime Environment 1.5. First you will have to check if you already have Java installed. Start Command Prompt and type

```
java -version
```

If you have Java Runtime Environment installed, your system will print something like

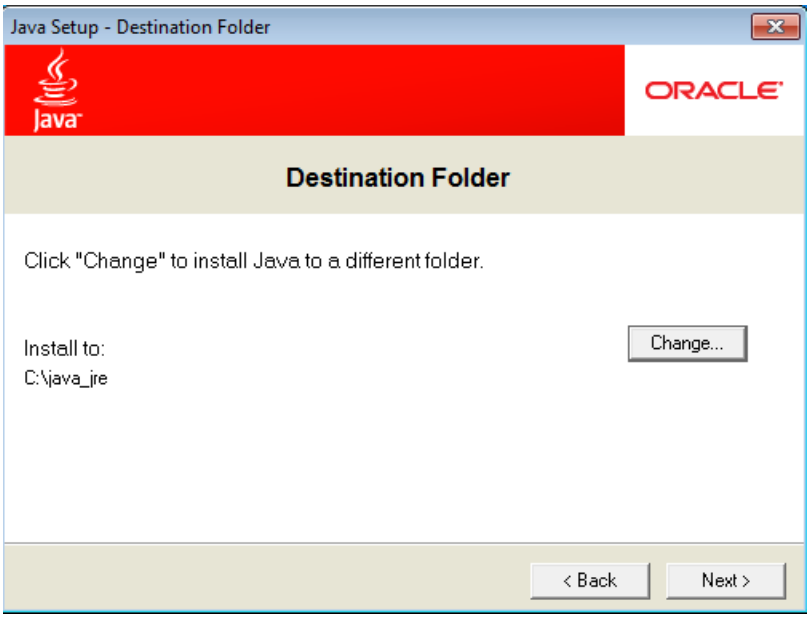
```
C:\Windows\system32>java -version
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) Client VM (build 20.4-b02, mixed mode, sharing)
```

If you have Java Runtime Environment installed and its version is at least 1.5 you can skip Java Runtime Environment installation and move on to installing Jython.

Java installation is divided into 3 steps: download, install and verify. Java 1.6 update 29 can be downloaded from Java homepage (http://www.java.com/en/download/index.jsp). When you have downloaded Windows installation package double click it to start:

1. Checkbox from bottom left corner and click Install.

2. Select location where you want to install Java and click Next



3. Next installation asks you do you want to install toolbars. Those are extra options and you don't need them. Click Next.
4. Verify installation. Start Command Prompt and type

```
java -version
```

Your system should print your Java version like shown below.

```
C:\Windows\system32>java -version
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) Client VM (build 20.4-b02, mixed mode, sharing)
```

Install Jython

Robot Framework 2.6.3 requires at least Jython 2.5. First you will have to check if your system already has Jython installed. Start Command Prompt and type:

```
jython --version
```

If you have Jython installed, your system will print something like

```
C:\Users\hemisto>jython --version
Jython 2.5.2
```

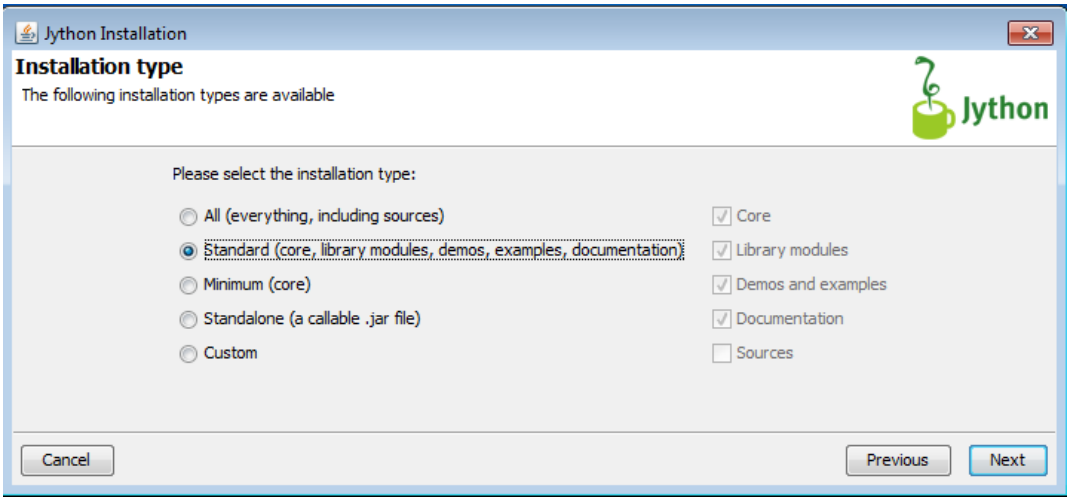
If you have Jython installed and its version is at least 2.5 you can skip Jython installation and move on to installing Robot Framework.

Jython installation is divided into 3 steps: download, install and verify. Jython 2.5.2 can be downloaded from Jython homepage (http://wiki.python.org/jython/DownloadInstructions). When you have downloaded jar -file you can double click it to start installation.

If you want to run installation as Administrator, start your Command Prompt as Administrator and navigate to folder where you downloaded Jython jar file. Then type

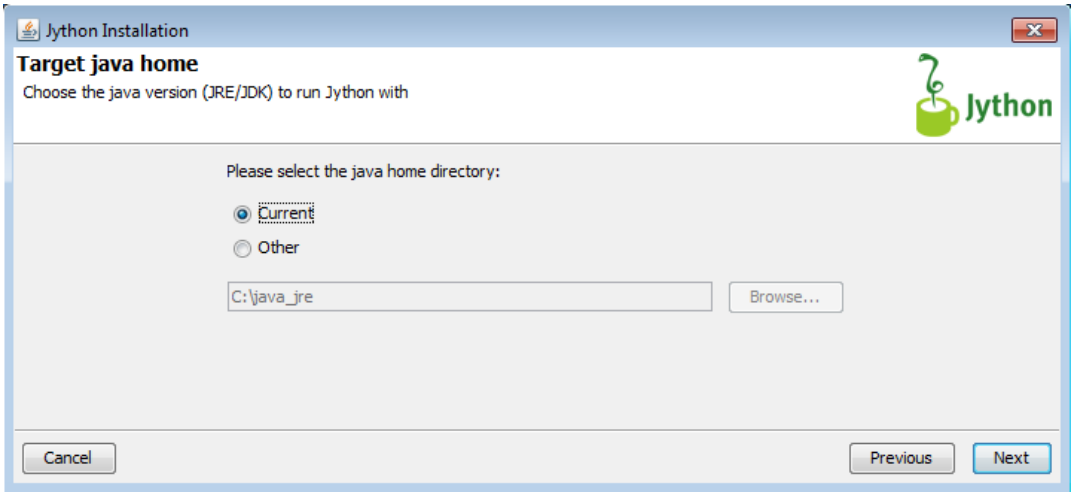
```
java -jar jython_installer-2.5.2.jar
```

1. Choose language and click Next.
2. Accept license.
3. Choose installation type. If you don't know what you need, choose standard. Click Next

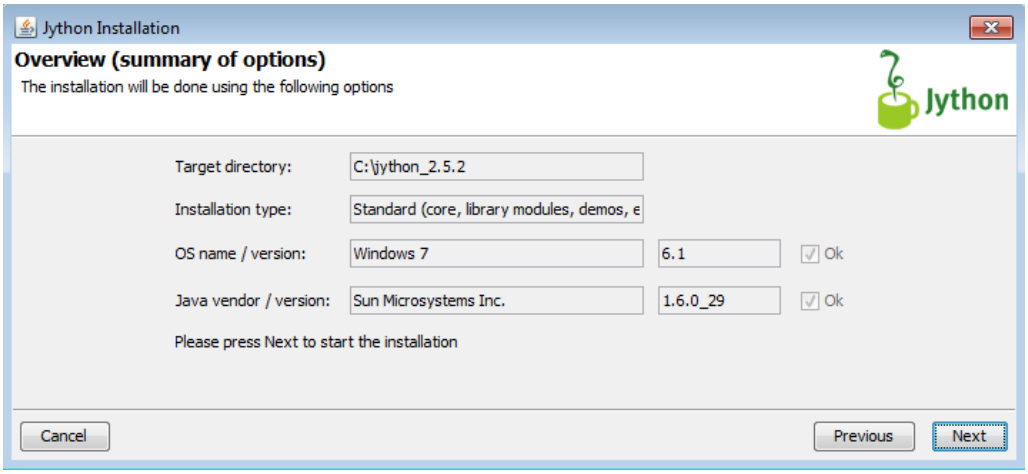


4. Choose target folder and click Next.

5. Choose Java folder. "Current" will use active Java and "Other" will let you choose which Java you want to use to run Jython. Choose the one you want and click Next.



6. Check that everything is okay at summary page and click Next.



7. When installation is done, append system Environmental variable "*Path*"

```
;<path_to_jython_folder>
```

8. Verify installation. Start Command Prompt and type

```
jython --version
```

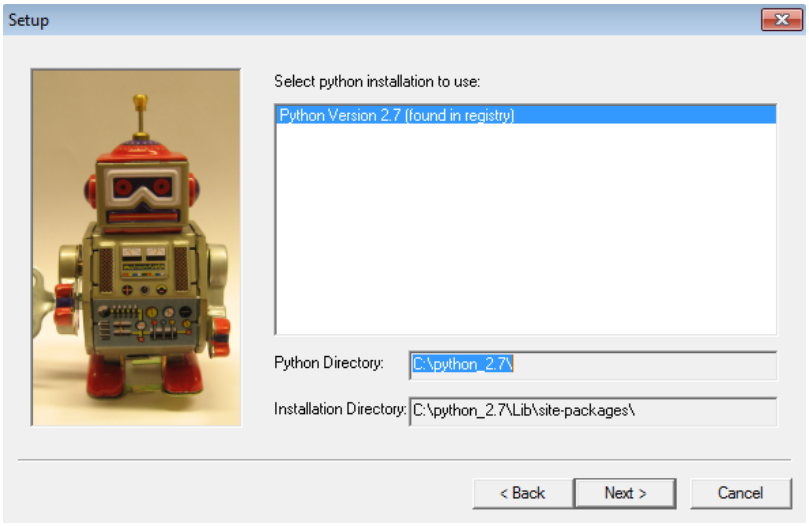
System should print your Jython version like shown below.

```
C:\Users\hemisto>jython --version
Jython 2.5.2
```

Install Robot Framework 2.6.3

Robot Framework installation is divided into 3 steps: download, install and verify. Robot Framework 2.6.3 can be downloaded from Robot Framework homepage (http://code.google.com/p/robotframework/downloads/list). When you have downloaded Windows 32-bit installation package double click it to start installation:

1. Read info text and click Next
2. Check that Robot Framework finds your Python



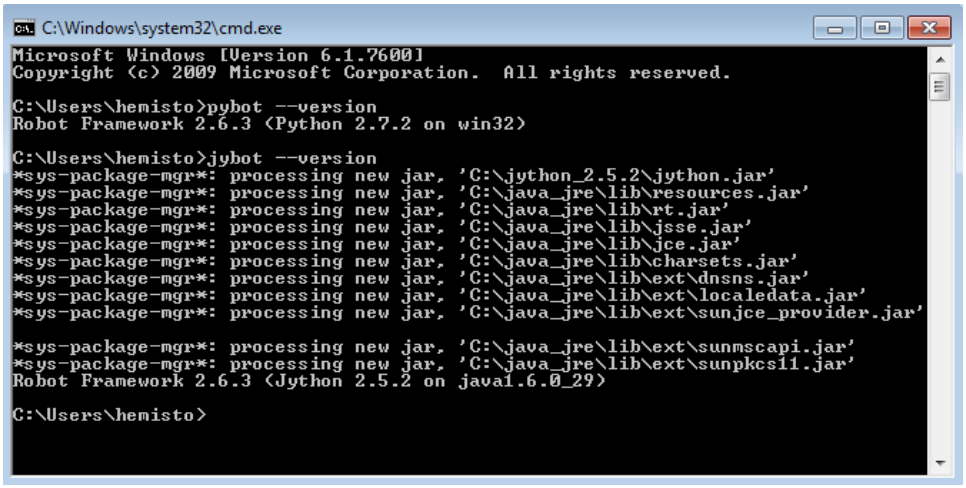
3. Click Next again to start installation
4. When installation is done click Finish
5. Finally append system Environmental variable "*Path*"

```
;<path_to_python>\Scripts
```

6. Verify installation. Start Command Prompt and type

```
pybot --version
jybot --version
```

System should print Robot Framework version like shown below. Notice that jybot -command will first time process some jar -files.

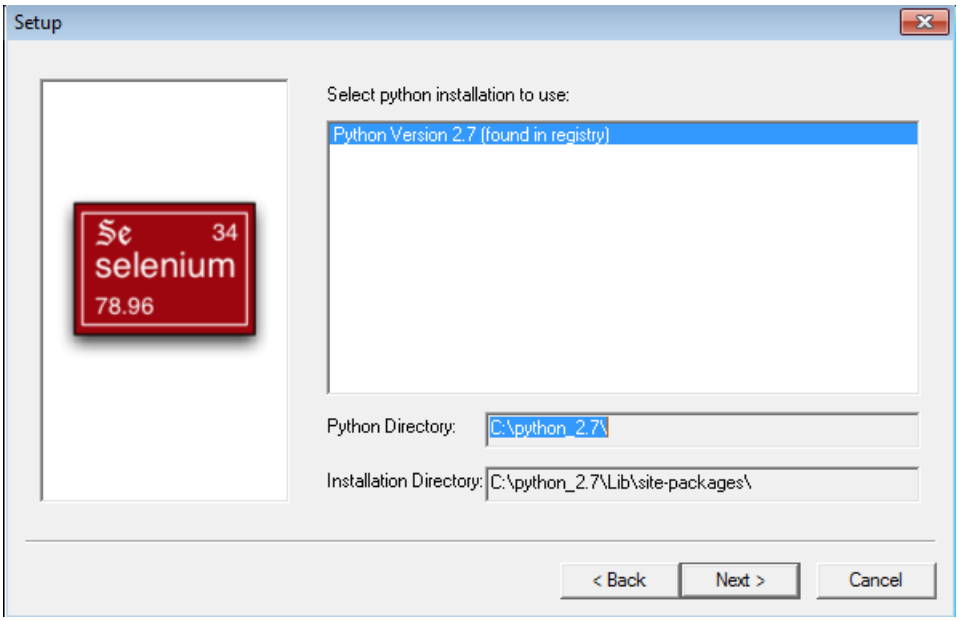


Robot Framework Selenium library 2.8

Finally you will need to install Robot Framework Selenium library that is used to test web applications.

After you have downloaded Selenium library 2.8 Windows 32bit installation file from Robot Framework homepage (http://code.google.com/p/robotframework-seleniumlibrary/downloads/list), start installation by double clicking it.

1. Read info about Selenium library and click Next
2. Check that Robot Framework Selenium library finds your Python and click Next



3. Click Next to start installation
4. Click Finish

Selenium library installation can only be tested by writing a test file and executing this file. Take a look to *Test your environment* tutorial.

Links

Java. http://www.java.com/en/download/index.jsp

Jython. http://wiki.python.org/jython/DownloadInstructions

Preconditions. http://code.google.com/p/robotframework/wiki/Installation#Preconditions

Python. http://www.python.org/

Robot Framework. http://code.google.com/p/robotframework/downloads/list

Robot Framework Selenium Library. http://code.google.com/p/robotframework-seleniumlibrary/downloads/list

Liite 2. Test your environment

Test your environment

Contents

About this guide

This guide will tell how you can write simple test cases that will test your environment. Main point is to find out whether your environment is working as it should be. You will create one test case that will succeed and one test case that will fail. From failed test case Robot Framework should take a screenshot and link it to reports that will be generated when a run is completed.

You can follow this guide no matter what environment you are using. When the guide tells you that you have to open a text editor, open the text editor you prefer. Depending on your system, use Command Prompt or Terminal.

Prepare to write test cases

First something about the syntax Robot Framework uses. Robot Framework uses tabular syntax, which means that the next two lines mean different things

```
A B C
A  B  C
```

The difference between those two lines is that on the first line there is only one whitespace between A, B and C, whereas on the second line there is two whitespaces between A, B and C. First line will call keyword "A B C" without parameters. Second line will call keyword A with parameters B and C. Between words you have to use at least two whitespaces but of course you can use as many as you want to make test file easier to read.

Topping to that files are normal text files so you don't need any extra programs. If you want to use graphical editor then your choice will be RIDE. More about RIDE can be found at http://code.google.com/p/robotframework-ride/. In this guide you will use normal text editor like Gedit or Notepad.

Because this tutorial is written thinking that you will use a text editor, it is really important that you will learn how to comment your code. If you don't comment your scripts others might not understand what you meant when you wrote it. For Robot Framework the comment mark is #. If the # character is the first character of the cell Robot Framework will handle it as a comment. And if you want to use the # character normally you will have to type \#.

```
#For example this line is ignored because it is a comment
\#This line will NOT be ignored
```

You can also document your code with [Documentation] prefix. This prefix is nice to use to describe what your test case does or what your keyword does. Nice thing about this is that it is shown on a log file (you will learn more about it later). After this prefix you can write as much documentation as you like, but if you want to write it to multiple lines you have to rewrite prefix or write three dots. The three dots mean that data is split to several lines.

```
[Documentation]   This is documentation line that you can add to your
...               test case or keyword. Notice that you can split your
...               comment to multiple lines. You just have to rewrite
[Documentation]   documentation prefix or write three dots. In both cases
```

```
[Documentation]    your documentation is shown at log file as it was
...                written to one line.
```

Start Selenium server

What is Selenium server and where do you need it? Selenium server is a program that is designed to test web clients. There are two different kinds of Server setups: one is a basic Selenium server where you have only one server that will execute your tests. The other one is a multiplatform Selenium Grid. Selenium Grid allows you to run same tests in different environments at the same time. For example one Grid server can run test cases on the Windows XP operating system with Internet Explorer 7 and other Grid server can run same test cases on the Ubuntu operating system with Firefox 7. More information about Selenium server and Selenium Grid can be found at http://seleniumhq.org/projects/

When you start your test program it will send commands to Selenium server which will execute them and then report results back to your program.

Before you can start your Selenium server you will have to find it. The needed Selenium jar file can be found:

- In Windows 7 from the folder "<path_to_python_folder>\Lib\site-packages\SeleniumLibrary\lib"
- In Ubuntu 11.04 from the source tar ball "<source_tar_bal>/src/SeleniumLibrary/lib"

You can start the server by with double clicking the jar file from the file system but if you do it that way what about possible error messages? Where they are printed and how you can see them? So I highly recommend that you start Command Prompt or Terminal and navigate to folder where the Selenium Server jar file is and start it with "java –jar selenium-server.jar". If you start it from Command Prompt or Terminal you can easily add start up parameters like what is the port that the Selenium server will use but let's not talk about them in this tutorial.

Write your first test file

Now that you have Selenium server running let's write a test file that will start browser, Firefox, and run couple of easy, I mean really easy, tests. If a test fails Robot Framework will take a screenshot.

First create a test file for example "my_test.txt". Insert following lines to this file and let's then talk about what each line does. Remember that whitespaces matter!

```
*** Settings ***
#normally the Selenium library is used to a web testing but in this
#example it is used to take screenshots.
Library   SeleniumLibrary


*** Test Cases ***
my_test_pass
    #call a keyword that will start a browser
    Set Up

    #compares whether the two values are equal
    Should Be Equal   1   1

    #call the teardown method that will close browser
```

```
    #prefix [TearDown] will make function called even if the test fails
    [Teardown]    Tear Down


my_test_fail
    [Documentation]    This will start a browser and after that it will
    ...                compare two numbers twice.

    Set Up
    Should Be Equal    2    2
    Should Be Equal    2    3
    [Teardown]    Tear Down


*** Keywords ***
Set Up
    [Documentation]    Open Firefox and navigate to google.com

    #start Firefox and navigate to google.com
    Open Browser    http://www.google.com    firefox


Tear Down
    [Documentation]    Will run "Capture Screenshot" keyword (found
    ...                from the Selenium library) only if the test failed.
    ...                After that, keyword will close the browser.

    #take a screenshot if test failed
    Run Keyword If Test Failed    Capture Screenshot

    #close browser if it was open
    Close Browser
```

Now that you are done lets go through this file line by line so you understand what this file do before you run it.

Line by line explanation

Robot Framework can be divided into three sections. Each section header will start and end with three stars.

- ***Settings***
- ***Test Cases***
- ***Keywords***

These three sections are described in the next three chapters.

Settings

The settings section is used to import external files, for example libraries or other files that contains keywords. If you create a keyword file you can use those keywords from many different places and that way make your tests easier to handle.

```
*** Settings ***
#normally Selenium library is used to a web testing but in this
#example it is used to take screenshots.
Library    SeleniumLibrary
```

These first four lines are used to include external library. The first line is a header and it specifies that this code block will import other things and it does not include test cases. Both lines that start with the #

character are comment lines and will be ignored when test is running. "Library SeleniumLibrary" line includes Selenium library, which is used to take screenshots in this example.

There are lots of external libraries and you can find more info about them at Robot Framework website (http://code.google.com/p/robotframework/).

Test cases

Test Cases section will contain all test cases. Test cases can use the libraries mentioned on the Settings section and the keywords mentioned on the Keywords section. There can be unlimited amount of different test cases but in the test file you created there are only a couple of really simple cases.

```
*** Test Cases ***
my_test_pass
    #call keyword that will start a browser
    Set Up

    #compares two values are they equal
    Should Be Equal   1   1

    #call teardown method that will close the browser
    #prefix [TearDown] will make function called even if test fails
    [Teardown]   Tear Down


my_test_fail
    [Documentation]   This will start a browser and after that it will
    ...               compare two numbers twice.

    Set Up
    Should Be Equal   2   2
    Should Be Equal   2   3
    [Teardown]   Tear Down
```

First test case is called “my_test_pass” and it has three keywords “Set Up”, “Should Be Equal” and “TearDown”. “Set Up” and “TearDown” are keywords you created to avoid copy-paste as much as possible. Using similar keywords before a test case and after it, will lead you to good code conventions. For example "Set Up" keyword can contain database insert script and "Tear Down" keyword will remove inserted data from database.

"Should Be Equal" is Robot Framework build-in keyword that will compare two strings whether they are equal or not. From the test above you can see that "my_test_fail" will fail because 2 is not equal to 3. There are many other build-in keywords and you can learn more at Robot Framework website (http://code.google.com/p/robotframework/).

There is something special about “TearDown” keyword. As you can see there is [Teardown] prefix. This prefix makes keyword to be run even if a test itself fails. Without this prefix the "Tear Down" keyword will be run only when all steps above succeed.

Keywords

Keywords section will contain all keywords. Keywords can use libraries that are included in Settings section. Keywords are used to avoid copying. You could have written

```
my_test_pass
    #start Firefox and navigate to google.com
```

```
    Open Browser   http://www.google.com   firefox

    #compare two values and they should be equal
    Should Be Equal   1   1

    #take a screenshot if a test failed
    Run Keyword If Test Failed   Capture Screenshot

    #close browser if it was open
    Close Browser
```

If you had done this then you would have to copy paste "Open Browser" to every single test case. Same goes for "Close Browser".

There are three different kinds of keywords. Build-in keywords are keywords that come with Robot Framework itself, like "Should Be Equal". There are also those kinds of keywords that come with an external library, for example "Capture Screenshot" will come when you include the Selenium library. And finally keywords you have written by yourself, look below.

```
*** Keywords ***
Set Up
    [Documentation]   Open Firefox and navigate to google.com

    #start Firefox and navigate to google.com
    Open Browser   http://www.google.com   firefox


Tear Down
    [Documentation]   Will run "Capture Screenshot" keyword (found
    ...               from the Selenium library) only if test failed.
    ...               After that, keyword will close the browser.

    #take a screenshot if test failed
    Run Keyword If Test Failed   Capture Screenshot

    #close the browser if it was open
    Close Browser
```

First keyword is ”Set Up” and second keyword is “Tear Down”. Keywords can have as many steps as you want. I named every first letter to uppercase because it seemed to be what Robot Framework is using, of course you can use what you want as long as it doesn't break the Robot Framework syntax. Do not forget that there is difference between uppercase and lowercase, "mom" is not same as "Mom".

Now let’s take a look what these keywords really do:

The “Set Up” keyword has only one step that is “Open Browser http://www.google.com firefox”. This step will try to open browser “firefox” and to navigate to the page www.google.com. Open Browser can be found from the Selenium library you included earlier on the “Settings” part.

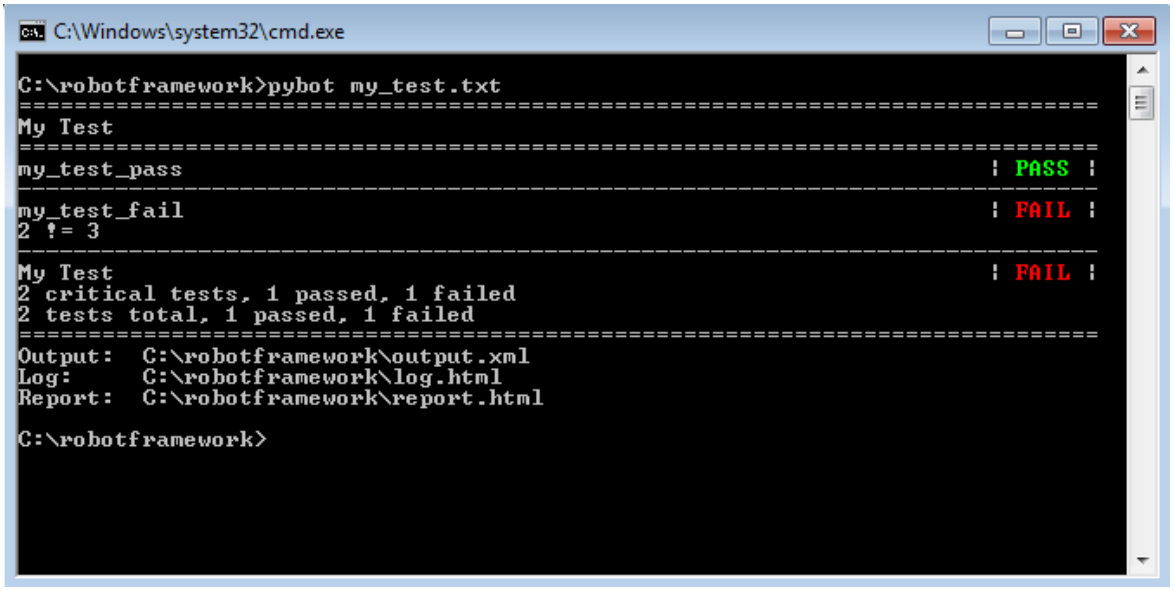
The “Tear Down” keyword has two steps. The first step is “Run Keyword If Test Failed Capture Screenshot”. The keyword "Run Keyword If Test Failed" is executed when a test case failed. This keyword can be found from the Robot Framework build-in library. The keyword "Capture Screenshot" is a keyword that can be found from the Selenium library. This keyword will capture a screenshot from a desktop.

And finally close the browser with "Close Browser" keyword. This command can also be found from the Selenium library.

More keywords can be found at the Robot Framework website (http://code.google.com/p/robotframework/wiki/TestLibraries).

Run your test

Open Command Prompt or Terminal and navigate to the folder where you saved your test file. To run tests type “pybot <test_file_name>”. This will run tests and will print results to Command Prompt or Terminal. Here is a screenshot from the Windows environment:

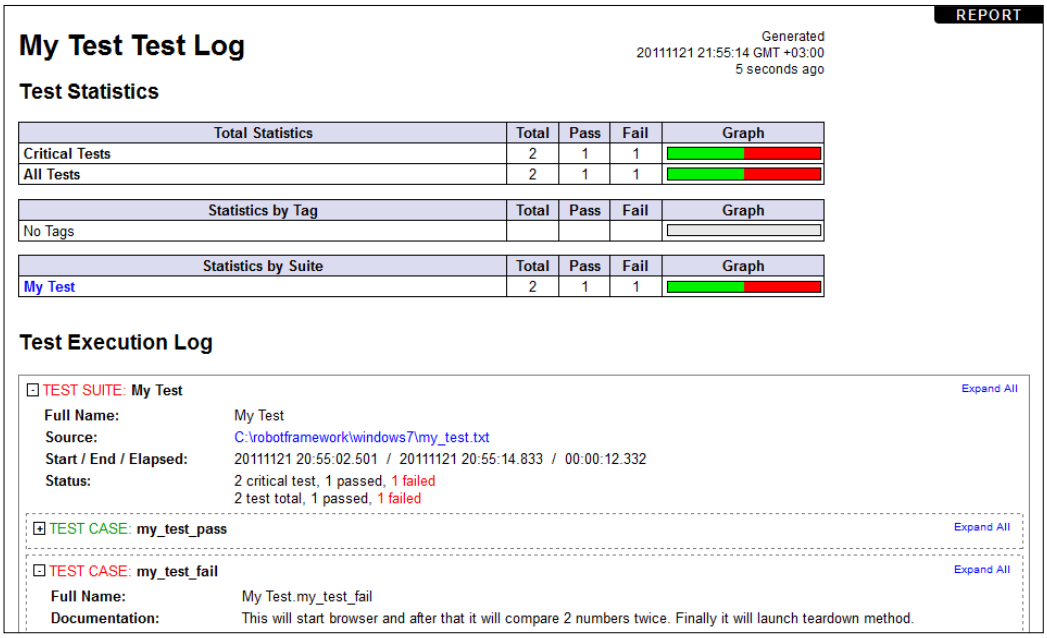


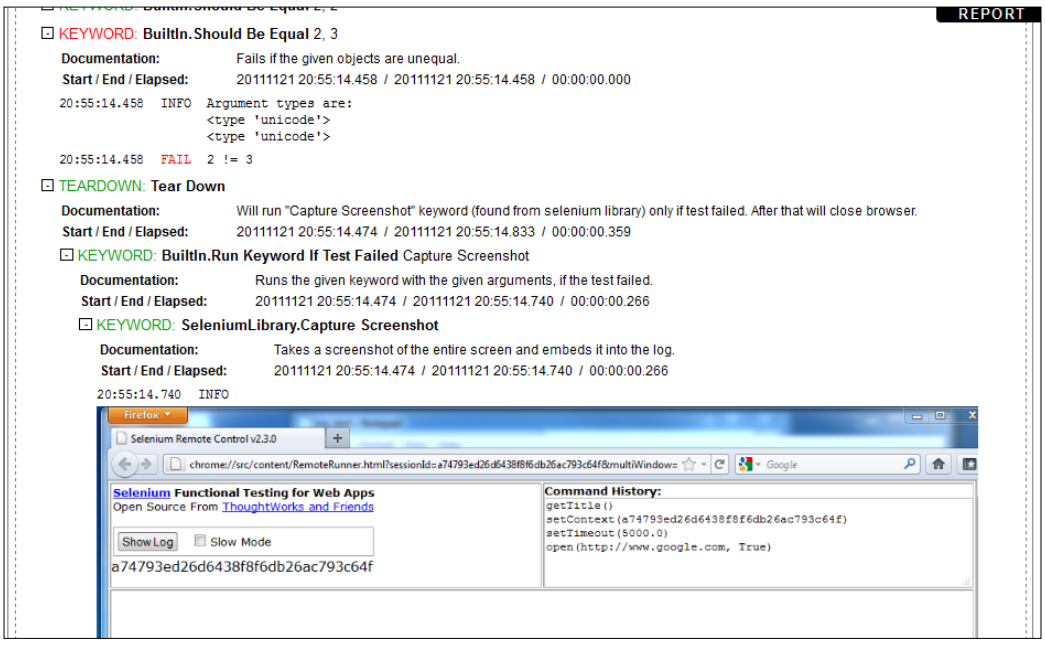
In additional this will create three files: output.xml, log.html and result.html. Next there are screenshots from each file and brief information about what the files contain.

- output.xml
  - The output.xml file records Robot Framework actions and log.html and result.html will read data from this file. A save location can be changed with the “--output” parameter.

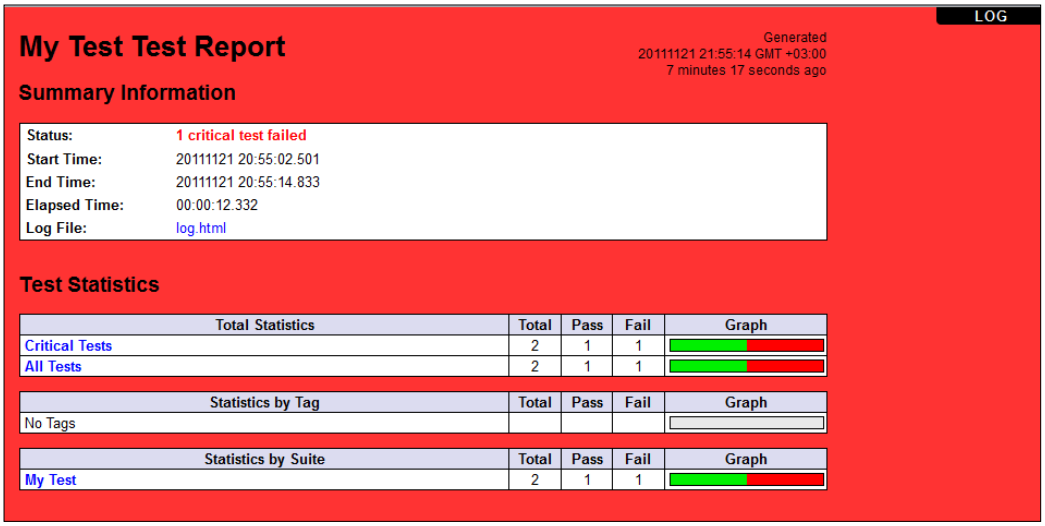
- log.html

- The log.html file contains information about test cases in the html form. This is read from the output.xml file. A save location can be changed with "--log" parameter.
- Here you can see your screenshot from a failed test case. The screenshot, which is taken from failed test case, is shown under keyword "TearDown".





- report.html
  - The report.html file contains an overview of the executed test cases. If both log and result files are generated, result file contains links to test cases that are in the log file. A save location can be changed with the “--report” parameter.



Summary

Now you have written a test set, which contain one passed and failed test case. Robot Framework will also generate a report that will tell you how many tests failed, passed and how many tests there were. Robot Framework will also generate detailed log file that contains step-by-step information from test cases. The screenshots are automatically linked to log file. Next time you will learn more details about writing tests. You will, for example, learn how you can tag your tests and make them run only when system runs smoke tests and so on.

Useful links

Robot Framework libraries can be found from http://code.google.com/p/robotframework/wiki/TestLibraries. For example build-in keywords can be found when you select "BuiltIn" and then version you are using.

Selenium Grid and Selenium Server can be downloaded from http://seleniumhq.org/projects/

Robot Framework IDE (RIDE) can be downloaded from http://code.google.com/p/robotframework-ride/

Liite 3. Install Robot Framework. Ubuntu 11.04

Install Robot Framework

Ubuntu 11.04

Contents

About this guide

In this guide you will install Java Runtime Environment 1.6.0 update 29, Jython 2.5.2, Robot Framework 2.6.3 and Robot Framework Selenium library 2.8 to Ubuntu 11.04 operating system. You will not install Python because Ubuntu installation already contains Python. Robot Framework needs Python and Jython to work and Jython needs Java Runtime Environment to work. Selenium library is extension to Robot Framework that can be used to test web applications.

Robot Framework runs on Python and Jython, you will need to have at least one of them installed. Robot Framework recommends that you will always install Python. Starting from Robot Framework 2.5 supports Python 2.5 or newer. Earlier versions are compatible with Python 2.3 or newer. Robot Framework is not compatible with Python 3.x. (Preconditions.)

You will have to install Python and Robot Framework to get started. You will have to install Java and Jython if you want to use 3rd party libraries that are written using Java. You will also have to install Selenium library if you want to test web applications.

After you have completed this tutorial you have an environment where you can create and run Robot Framework tests. Installation is complete when you are able to:

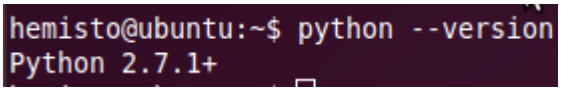
- Run simple test, where 1 will fail and 1 will pass
- Robot Framework takes a screenshot from failed test case
- Read a report that tells you how many tests succeeded and how many failed

Check Python

Robot Framework 2.6.3 requires at least Python 2.5, but not Python 3. Ubuntu 11.04 should have Python installed but you should check it before moving on to install Java and Jython. Start Terminal and type:

```
python --version
```

Your system will print Python version.



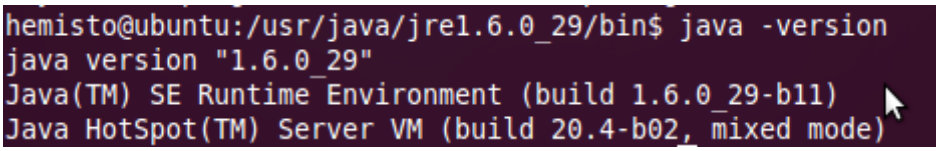
If you do not have Python installed or its version is less than 2.5 or over 3, you will have to install Python before you can install Robot Framework. You can download Python from http://www.python.org/download/ and you can find installation instructions inside source file.

Install Java Runtime Environment

Robot Framework 2.6.3 requires at least Jython 2.5. Jython 2.5.2 requires at least Java Runtime Environment 1.5. Before installing Java you will have to check if you already have it installed. Start Terminal and type

```
java -version
```

If you have Java Runtime Environment installed, your system will print something like



If you have Java Runtime Environment installed and its version is at least 1.5, you can skip Java Runtime Environment installation and move on to installing Jython.

Java installation is divided into 3 steps: download, install and verify. Java 1.6 update 29 can be downloaded from Java homepage (http://www.java.com/en/download/index.jsp). When you have downloaded Linux self-extracting files start terminal.

1. Navigate to folder where you want to install Java.

```
cd <folder_where_you_want_to_install_java>
```

2. Move downloaded file f to folder you want to install it

```
sudo mv <folder_where_java_file_is>/jre-6u29-linux-i586.bin
<folder_where_you_want_to_install_java>
```

3. Change file rights to make file executable

```
sudo chmod a+x jre-6u29-linux-i586.bin
```

4. Extract all files to this folder

```
sudo ./jre-6u29-linux-i586.bin
```

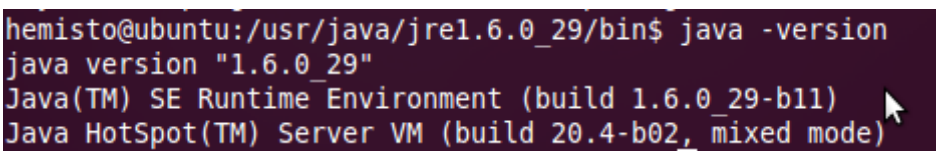
5. Now you have to tell system that you installed Java to this folder

```
sudo update-alternatives --install "/usr/bin/java" "java"
"<folder_where_you_want_to_install_java>/bin/java" 1
```

6. Verify installation. Open terminal and type.

```
java -version
```

System should print Java version like shown below



Install Jython

Robot Framework 2.6.3 requires at least Jython 2.5. First you will have to check if your system already has Jython installed. Start Terminal and type:

```
jython --version
```

If you have Jython installed, your system will print something like

```
hemisto@ubuntu:~/Downloads$ jython --version
Jython 2.5.2
```

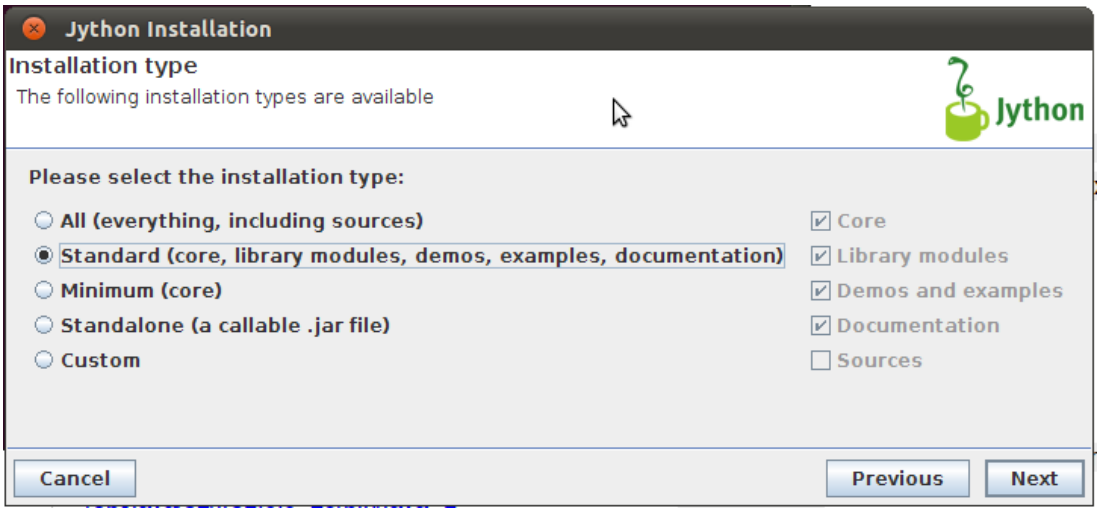
If you have Jython installed and its version is at least 2.5, you can skip Jython installation and move on to installing Robot Framework.

Jython installation is divided into 3 steps: download, install and verify. Jython 2.5.2 can be downloaded from Jython homepage (http://wiki.python.org/jython/DownloadInstructions). When you have downloaded jar -file start Terminal and navigate to folder where you downloaded it:

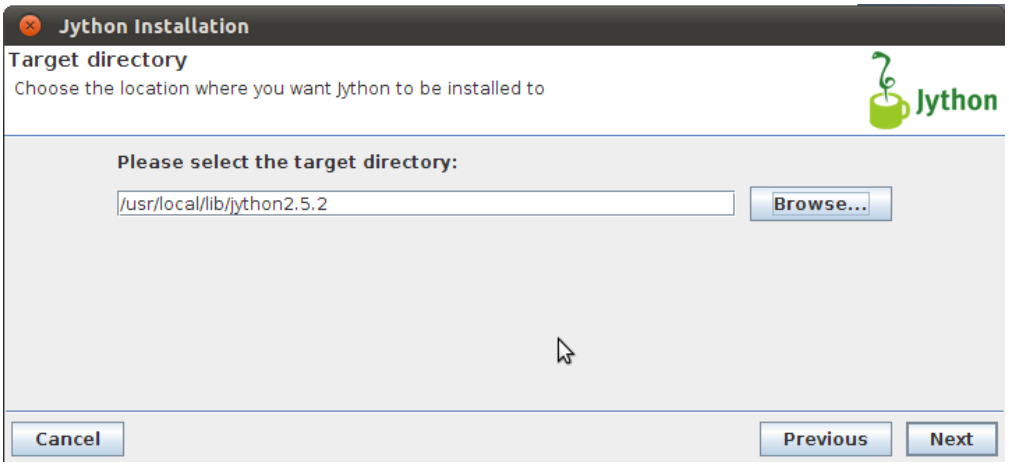
1. Start installation with

```
sudo java -jar jython_installer-2.5.2.jar
```

2. Choose language you want to install and click Next
3. Accept license
4. Choose installation type. Standard is your choice if you do not know what you need. Click Next.



5. Choose directory where you want to install Jython and click Next



6. Choose Java folder and click Next.
7. Check that summary window shows every value right and click Next.
8. When installation is done open Terminal and tell system that you installed Jython here

```
sudo update-alternatives --install "/usr/bin/jython" "jython"
"<jython_installation_folder>/bin/jython" 1
```

7. Verify installation. Open terminal and type.

```
jython --version
```

System should print Jython version like shown below.

```
hemisto@ubuntu:~/Downloads$ jython --version
Jython 2.5.2
```

Install Robot Framework 2.6.3

Robot Framework installation is divided into 3 steps: download, install and verify. Robot Framework 2.6.3 can be downloaded from Robot Framework homepage (http://code.google.com/p/robotframework/downloads/list). What you will need to download is "*.tar.gz"-packet.

1. Open Terminal and navigate to folder where you downloaded tar-file. Extract files with

```
tar -zxvf robotframework-2.6.3.tar.gz
```

2. Install Robot Framework with command

```
sudo python setup.py install
```

3. Verify installation with

```
pybot --version
```

```
sudo jybot --version
```

Notice that first time you will have to have to use "sudo" with jybot -command because Jython will process some files. After first time "sudo" is not necessary.

```
hemisto@ubuntu:~/Downloads/robotframework-2.6.3$ pybot --version
Robot Framework 2.6.3 (Python 2.7.1+ on linux2)
```

```
hemisto@ubuntu:~$ jybot --version
Robot Framework 2.6.3 (Jython 2.5.2 on java1.6.0_22)
```

Robot Framework Selenium library 2.8

Finally you will need to install Robot Framework Selenium library that is used to test web applications.

Robot Framework Selenium library 2.8 can be downloaded from Robot Framework homepage (http://code.google.com/p/robotframework-seleniumlibrary/downloads/list). You will need to download "*.tar.gz"-packet.

1. Open Terminal and navigate to folder where you downloaded tar-file. Extract files with

```
tar -zxvf robotframework-seleniumlibrary-2.8.tar.gz
```

2. Install Robot Framework Selenium library with command

```
sudo python setup.py install
```

Selenium library installation can only be tested by writing a test file and executing this file. Take a look to *Test your environment* tutorial.

Links

Java. http://www.java.com/en/download/index.jsp

Jython. http://wiki.python.org/jython/DownloadInstructions

Preconditions. http://code.google.com/p/robotframework/wiki/Installation#Preconditions

Python. http://www.python.org/download/

Robot Framework. http://code.google.com/p/robotframework/downloads/list

Robot Framework Selenium Library. http://code.google.com/p/robotframework-seleniumlibrary/downloads/list

Liite 4. Robot Framework. Atlassian Bamboo on Ubuntu 11.04

Robot Framework

Atlassian Bamboo on Ubuntu 11.04

Contents

About this guide

In this guide you will set up Robot Framework 2.6.3 to work with Atlassian Bamboo 3.4.2 build 2810 which has been installed to Ubuntu 11.04 operating system. You will not install Atlassian Bamboo, only create new plan that will run Robot Framework. During the setting up you will create a plan, tasks, shared artifact and variables.

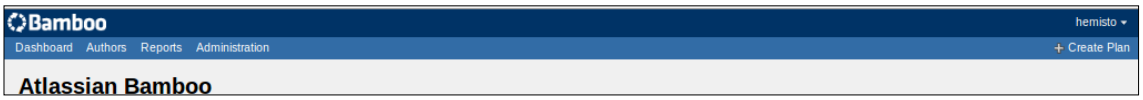
There are some preconditions:

- You have to have Atlassian Bamboo 3.4.2 installed
- You have to have Robot Framework 2.6.3 installed (you will run "pybot" command)
- You have to have Robot Framework Selenium library 2.8 installed
- You have to have Selenium Server or Grid running somewhere

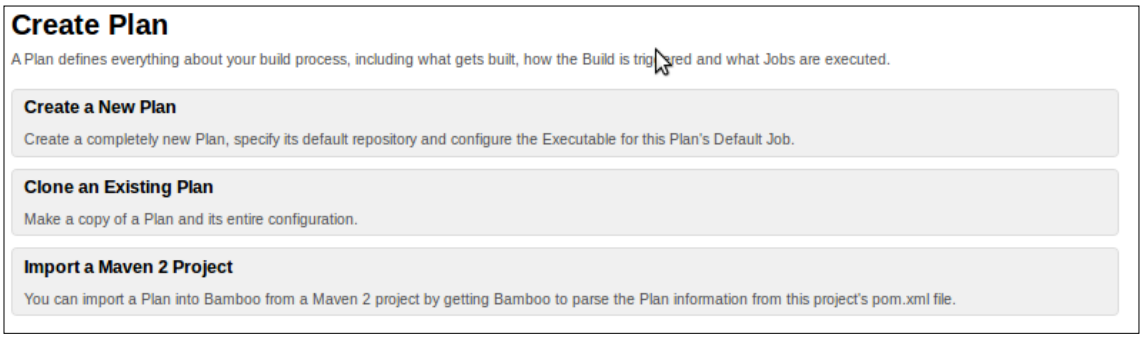
Create plan and repository task

In this chapter you will create plan and needed tasks to it.

1. Click Create plan button from top right corner



2. Press Create new plan

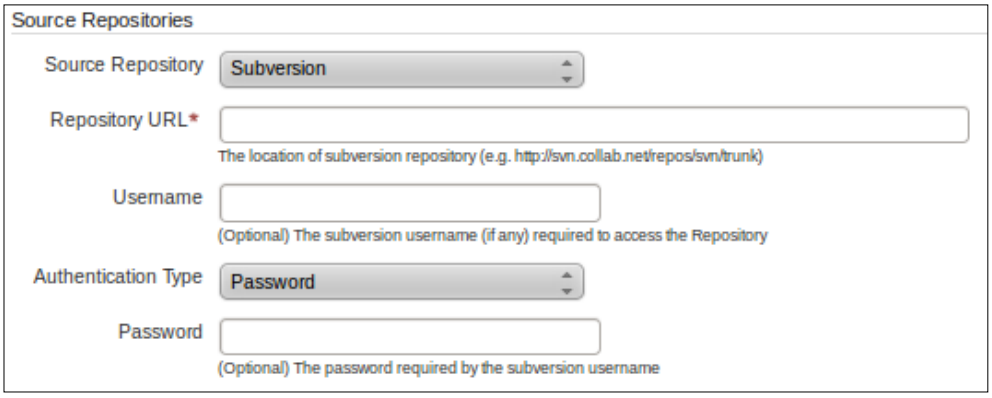


3. Fill in plan details

```
Plan name is name of your plan, for example "Robot Framework".
Plan key is unique key that can contain only uppercase alphanumeric
characters, for example "SMOKE".
Plan Description is description that you can give to your plan. For
example "This plan will run Robot Framework smoke tests"
```

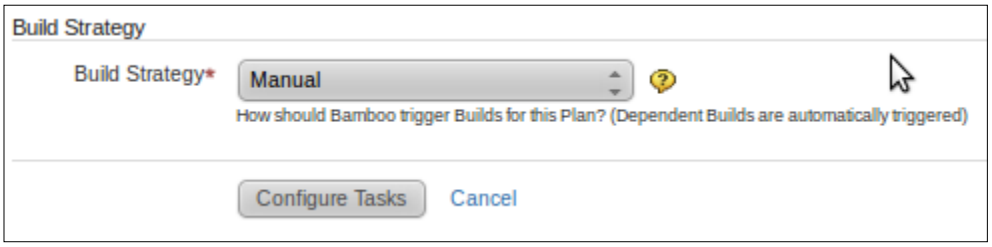
4. Fill in repository information. Notice that this is the repository that contains your tests

```
Source repository is target repository type.
Repository URL is your subversion URL that contains your tests.
Username, authentication type and password are things that you will
use to read data from repository. It is highly recommended that you
ask your subversion administrator to create an extra user that can
read data but not commit.
```

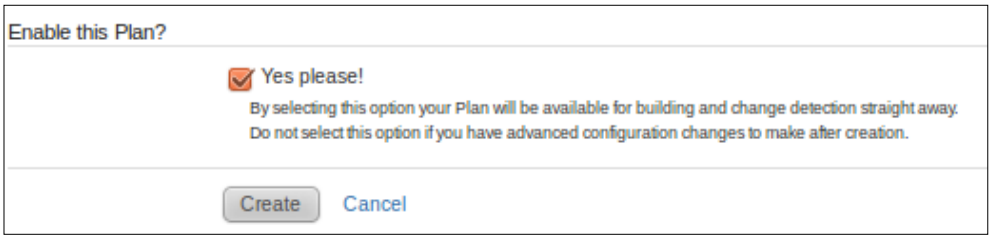


5. Choose build strategy to be manual. This will make easier to configure the plan. Later you can change it if you want.

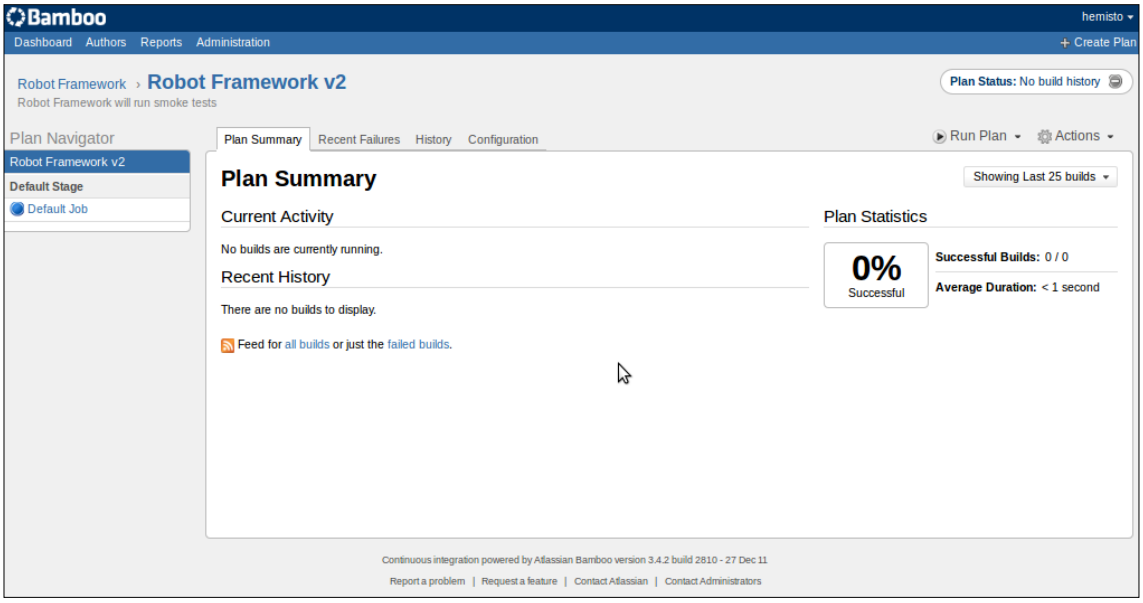
```
Build Strategy contains five options:
- Polling the Repository for changes
- Repository triggers the build when changes are committed
- Cron Based Scheduling
- Single daily build
- Manual
```



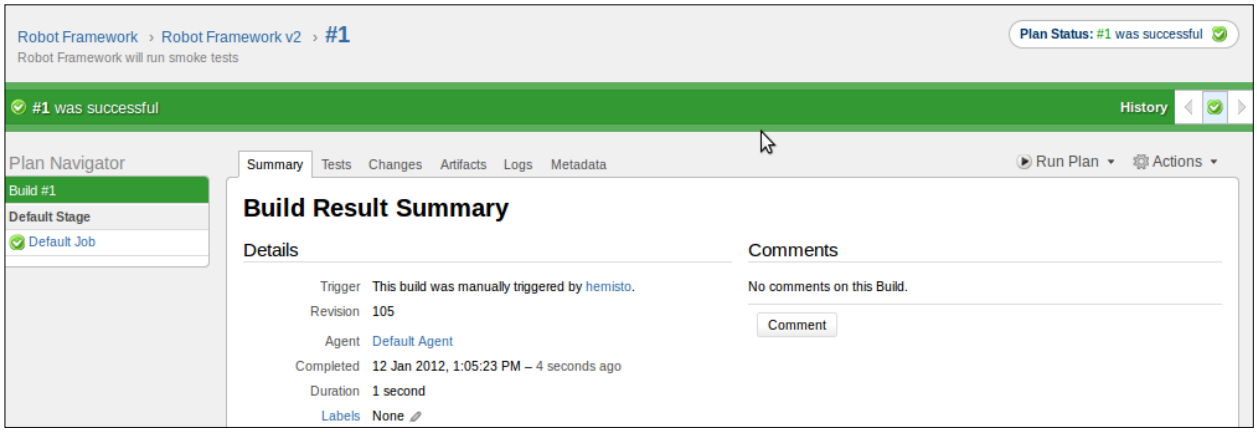
6. Click Configure Tasks.
7. Enable plan and click Create



8. Click Run Plan from the top right corner



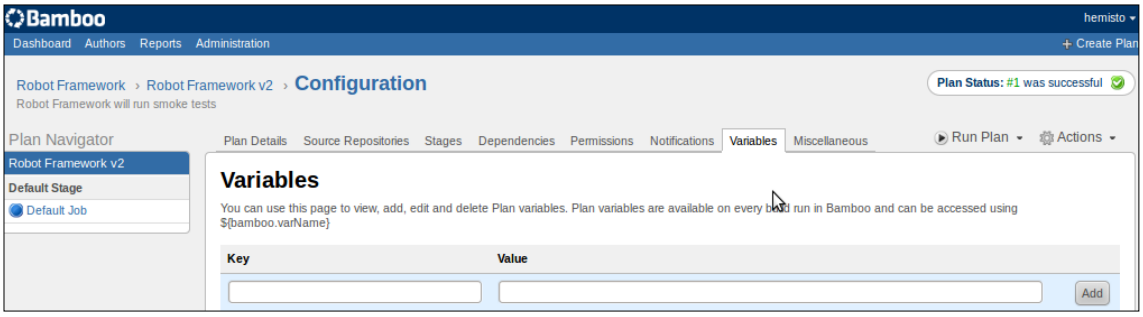
The plan will only fetch your tests and a result is shown below.



Create variables

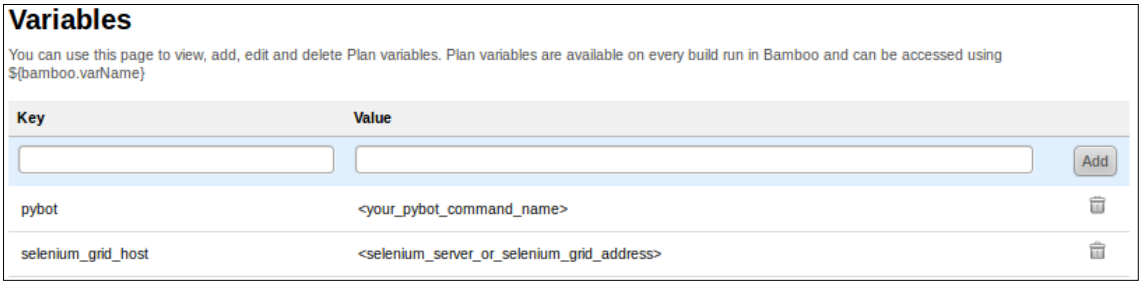
Next you will add variables that Atlassian Bamboo can store. This will make your scripting easier.

1. Navigate to your Plan
2. Click Actions -> Configure Plan
3. Navigate to tab Variables



4. Add two variables. One variable is for your pybot command and another is for Selenium server or Selenium Grid address. The key is a name that will be replaced with Value when Bamboo is running. You can call your variables with:

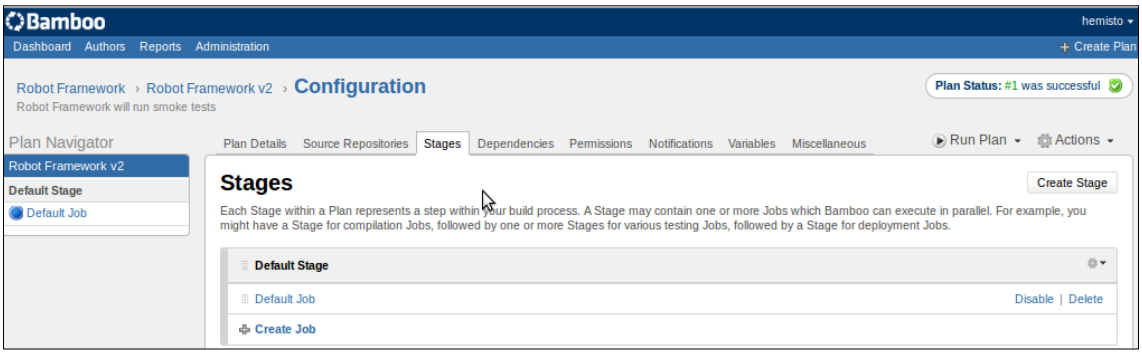
```
${bamboo.<key>}
```



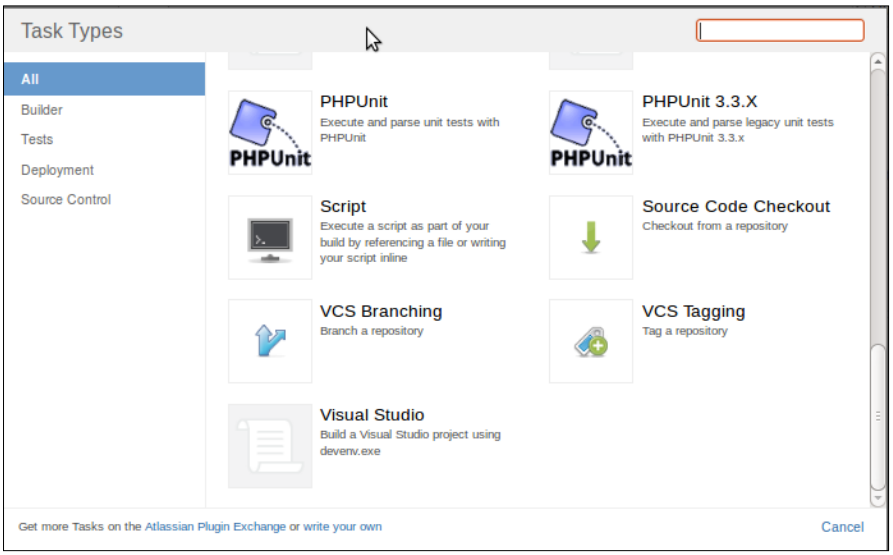
Create tasks

Next you will add two tasks to your plan, one is to run tests and the other one is to parse test results.

1. Navigate to your Plan
2. Click Actions and then Configure Plan
3. From Stages tab choose Default job

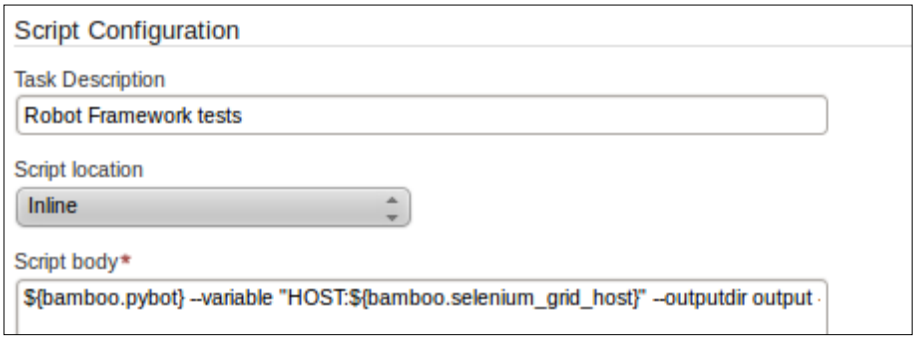


4. Click Add Task.
5. Find and choose task Script.

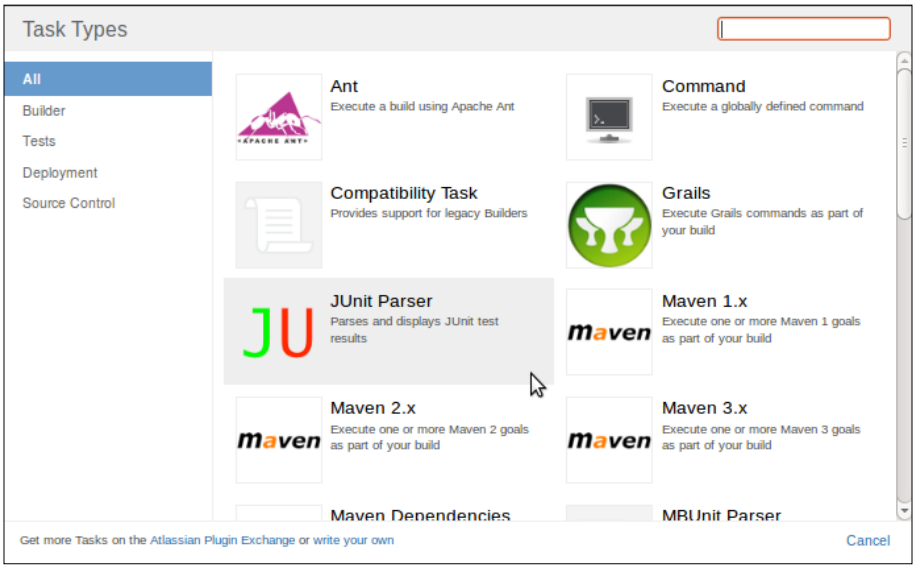


6. Write Task Description, for example Robot Framework tests
7. Set Script location to Inline
8. Write following command to Script Body

```
${bamboo.pybot} --variable "HOST:${bamboo.selenium_grid_host}" --
outputdir output --report index.html --xunitfile my_junit.xml
my_test.txt


${bamboo.pybot} -command will call your variables you added in the
previous chapter
--variable "HOST:${bamboo.selenium_grid_host}" -command pass Selenium
Server or Selenium Grid location to your pybot command
--outputdir <output_directory>  -command will change pybot command
output directory where all data will be stored. You can change the
location but remember it because you will need it soon.
--report index.html -command will create a report that is named to
index. If you want to open your report automatically you have to add
this parameter.
--xunitfile <xunit_file> -command will ask Robot Framework to create
xunit-file also. Bamboo uses this one to parse test results.
my_test.txt -command will tell Bamboo what your test file name is.
```

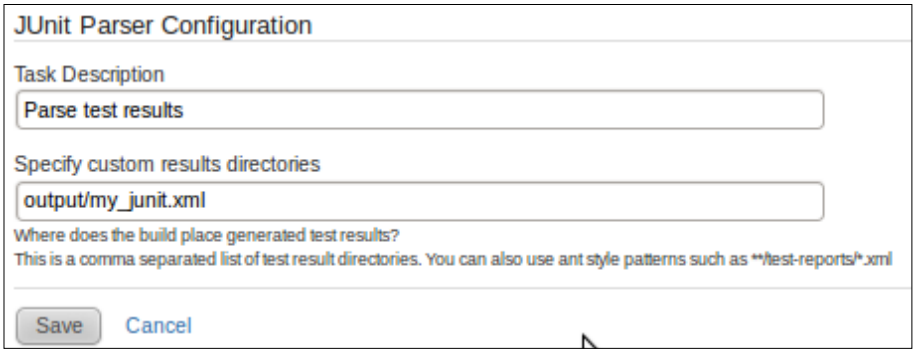


9. Click Save to save changes
10. Click Add Task.
11. Find and choose task JUnit Parser

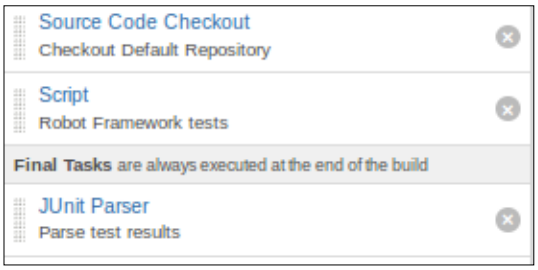


12. Fill in Task Description, for example Parse test results
13. Write next command to Specify custom results directories -field

```
<output_directory>/<xunit_file>
This will tell Atlassian Bamboo where xunit file is located
```



14. Click Save button.
15. Now drag-and-drop the JUnit Parser task to the Final Tasks section. This will make sure test results are always regenerated.



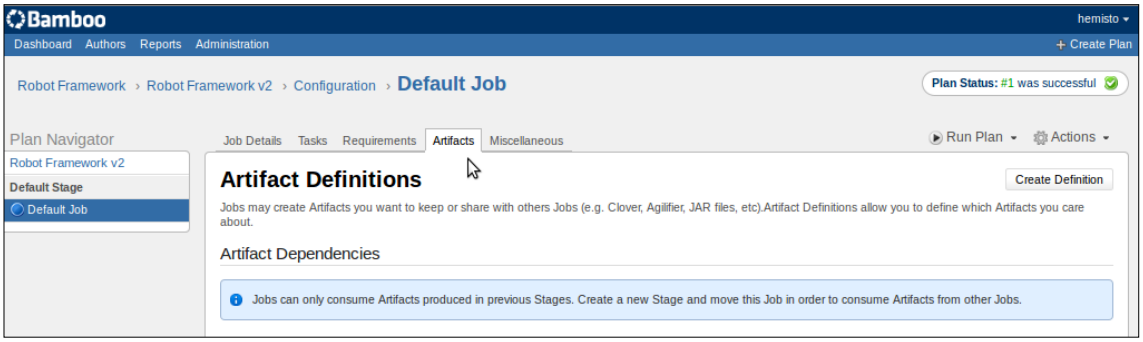
Agents

If you do not use Atlassian Bamboo on local server or you have more than one agent, you will need to add Remote Agent that is capable to run Robot Framework tests. This tutorial will not guide you in adding the agent. See http://confluence.atlassian.com/display/BAMBOO/Viewing+Bamboo%27s+Agents for more info.

Create shared Artifact

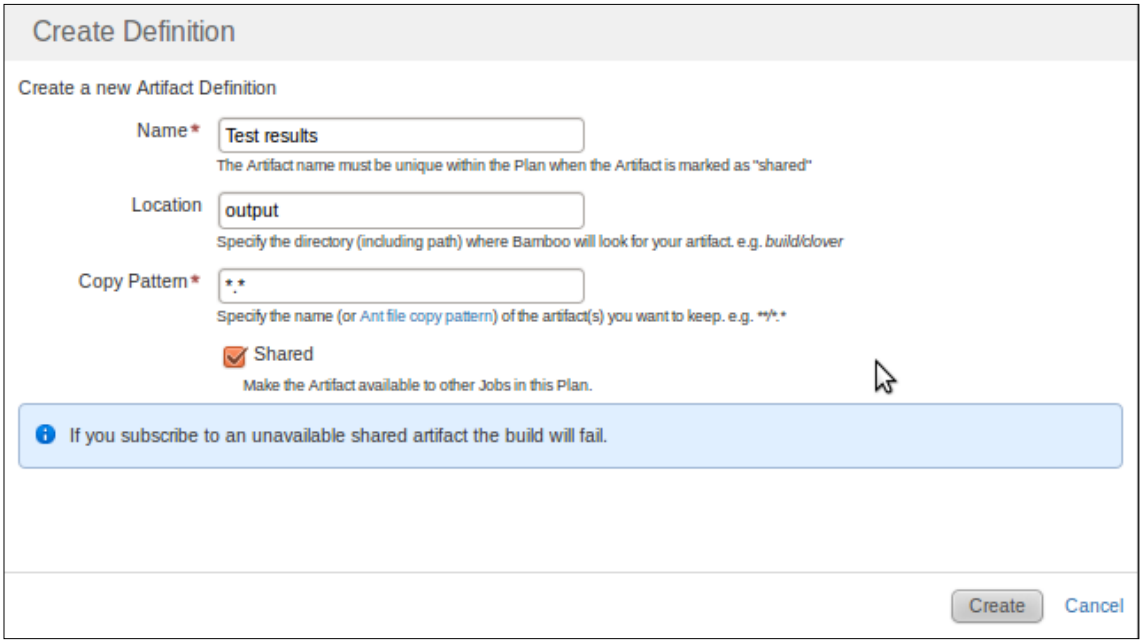
Next you will create a shared Artifact. This will link your test results to a summary page.

1. Navigate to your Plan
2. Click Actions and then Configure Plan
3. From Stages tab choose Default job
4. Open the Artifacts tab

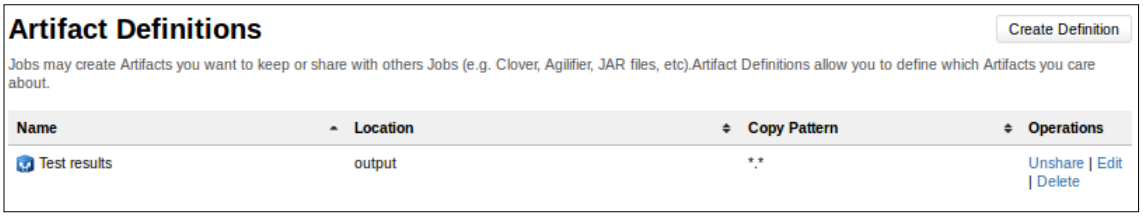


5. Click Create Definition
6. Fill in data

```
Name field is the name that Atlassian Bamboo will show on the summary
page
Location is the location you chose to save all data,
<output_directory>
Copy Pattern is the pattern that will match all shared files. Notice
that at the moment you can specify only one pattern / artifact
Check Shared. If you don't check this one the Artifact will not be
shown on the summary page
```



7. Click Create

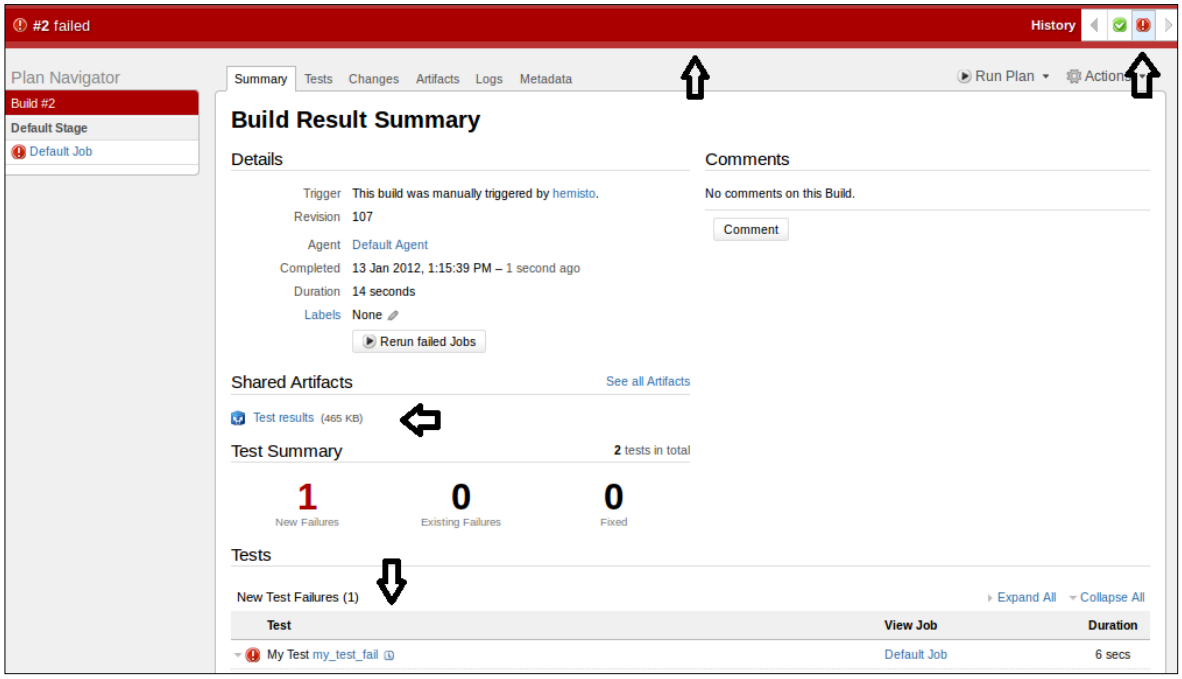


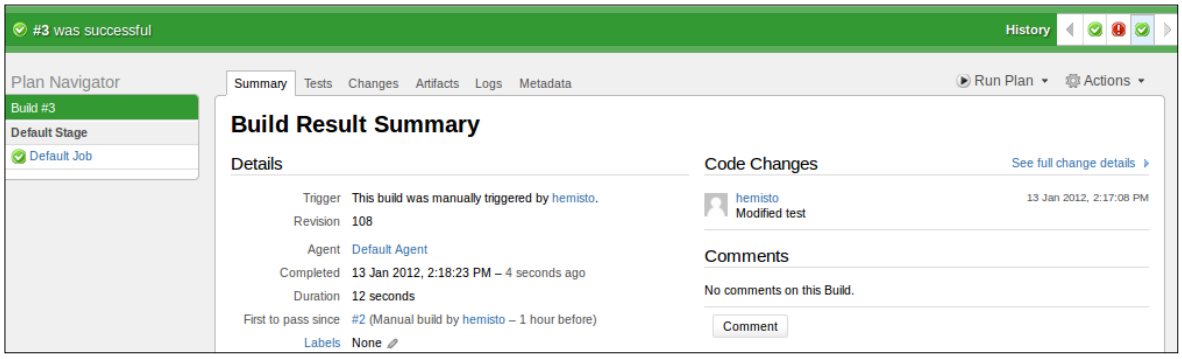
Run plan and look results

Now everything is done and you can run your plan.

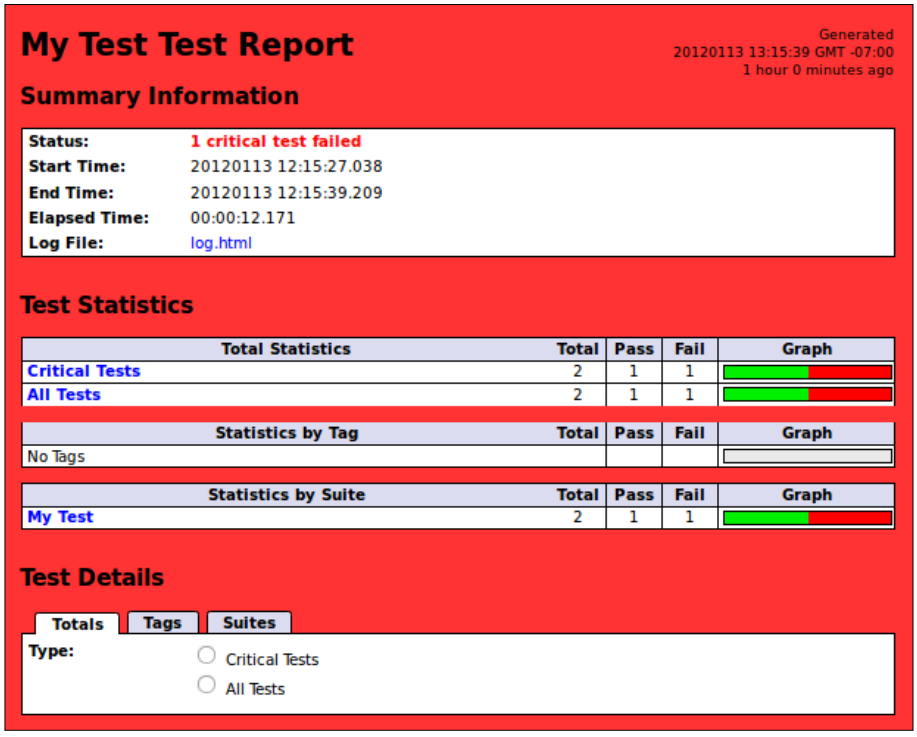
1. Navigate to your Plan
2. Click Run Plan from top right corner

When the run is completed you can see test results from Build Result Summary page. A background color tells you whether the plan succeeded (green) or failed (red). The top right corner shows history information. On the Summary tab you will see how many tests failed and how many passed. Reason why something failed can be seen on the bottom of the page. You can also navigate to the summary page by clicking Test results -link. You can see failed and passed test results below.





When you click Test results you will see Robot Framework's own detailed report.



My Test Test Report

Generated
20120113 13:15:39 GMT -07:00
1 hour 0 minutes ago

Summary Information

Status:	**1 critical test failed**
Start Time:	20120113 12:15:27.038
End Time:	20120113 12:15:39.209
Elapsed Time:	00:00:12.171
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	2	1	1	
All Tests	2	1	1	

Statistics by Tag	Total	Pass	Fail	Graph
No Tags				

Statistics by Suite	Total	Pass	Fail	Graph
My Test	2	1	1	

Test Details

Totals | Tags | Suites

Type: ○ Critical Tests ○ All Tests

Liite 5. Robot Framework. TeamCity on Ubuntu 11.04

Robot Framework

TeamCity on Ubuntu 11.04

Contents

About this guide

In this guide you will set up Robot Framework 2.6.3 to work with TeamCity Professional 6.5.6 build 18130 that has been installed to Ubuntu 11.04 operating system. You will not install TeamCity, only create new project that will run Robot Framework. During the setting up you will create new project and configure build settings. You will also add build steps, feature and artifacts.

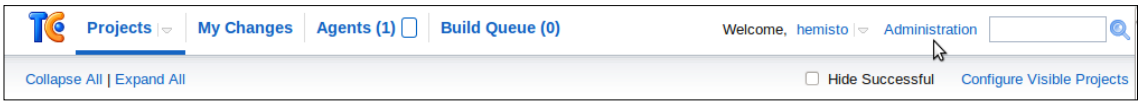
There are some preconditions:

- You have to have TeamCity 6.5.6 installed
- You have to have an Agent working
- An Agent have Robot Framework 2.6.3 installed (you will run "pybot" command)
- An Agent have Robot Framework Selenium library 2.8 installed
- You have to have Selenium Server or Grid running somewhere

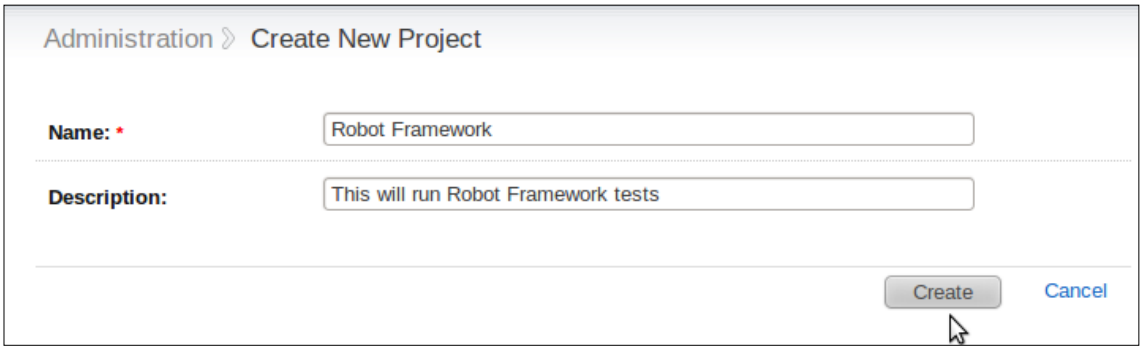
Create project and task

In this chapter you will create a project and task to it that will run your Robot Framework tests.

1. Navigate to Administration section

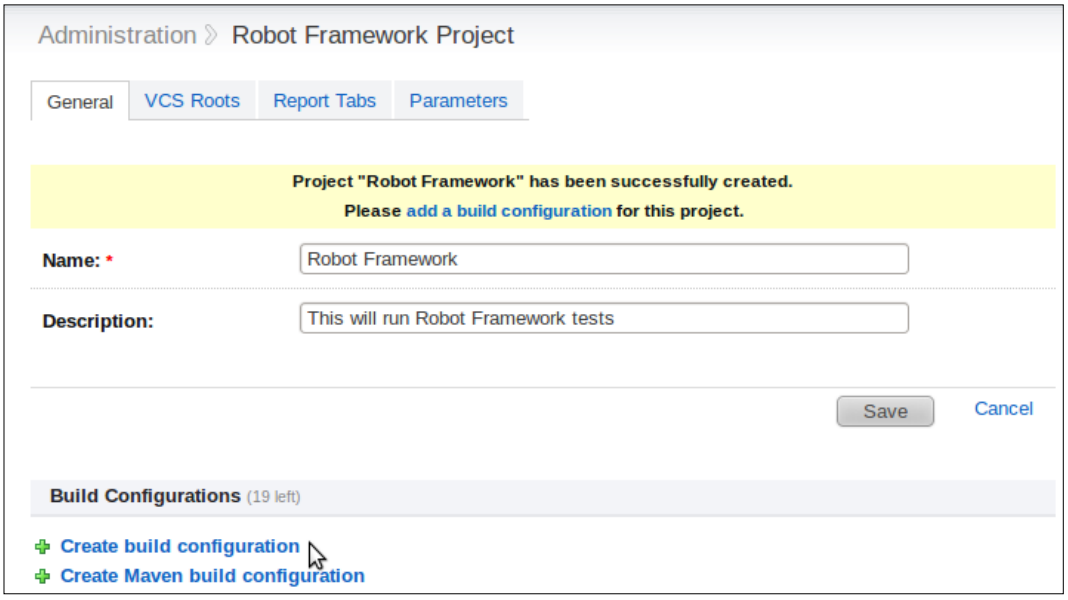


2. Click Create Project
3. Fill in project Name and Description

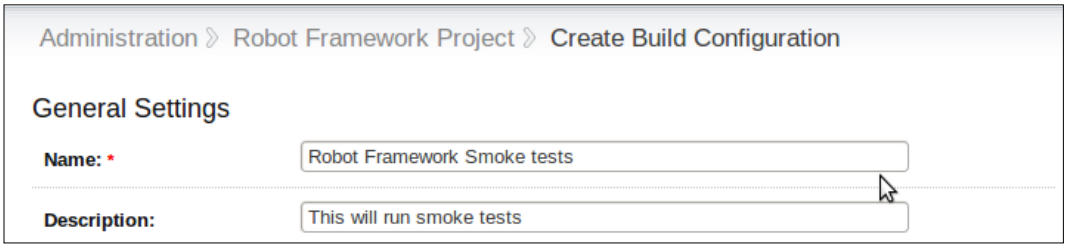


4. Click Create

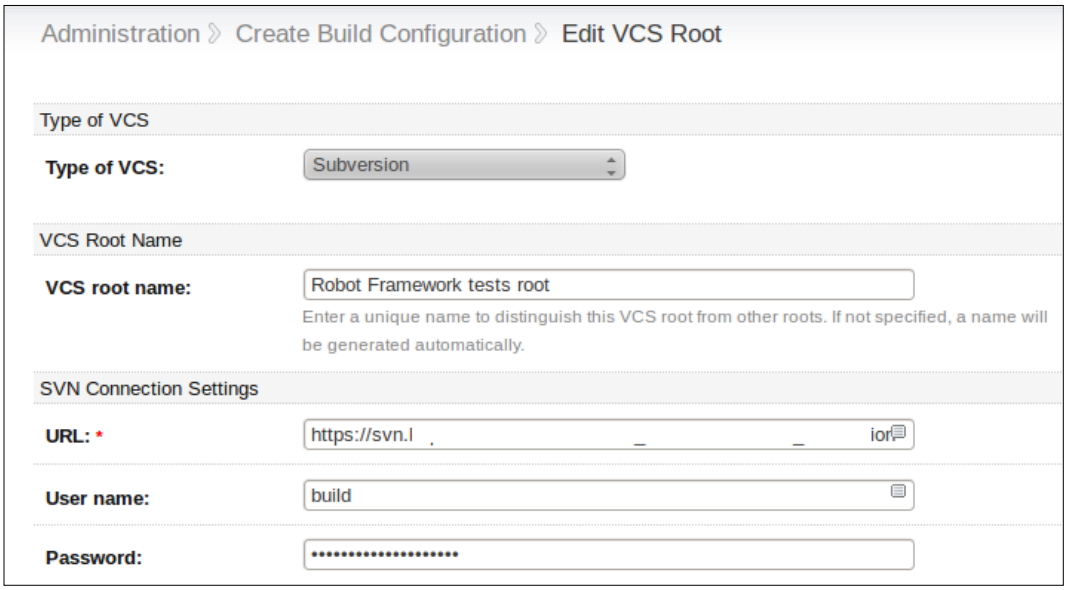
5. Click Create build configuration or add a build configuration link



6. Fill in Name and Description. Click "VCS settings >>"-button

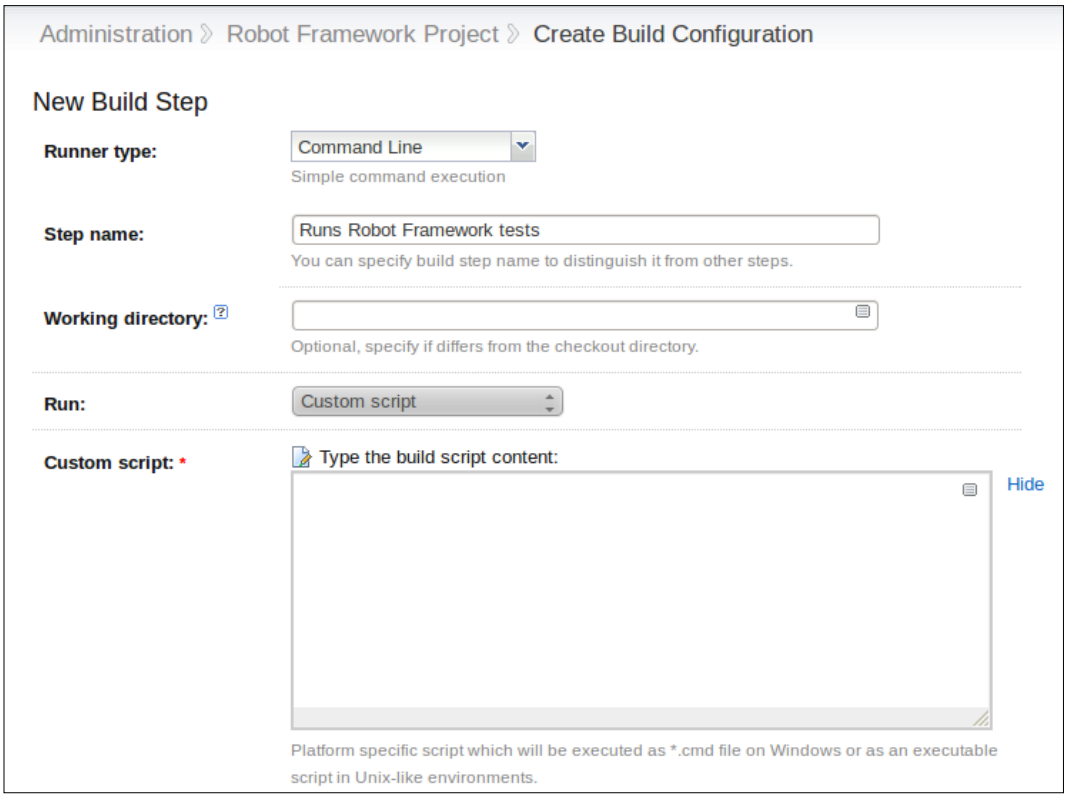


7. Click Create and attach new VCS root
8. Choose your version control type. For example Subversion.
9. Fill in needed data. In Subversion case VCS root name, URL, User name and Password.



10. Click Test connection. If your connection is successful. If problems occurred try to fix them and retest connection

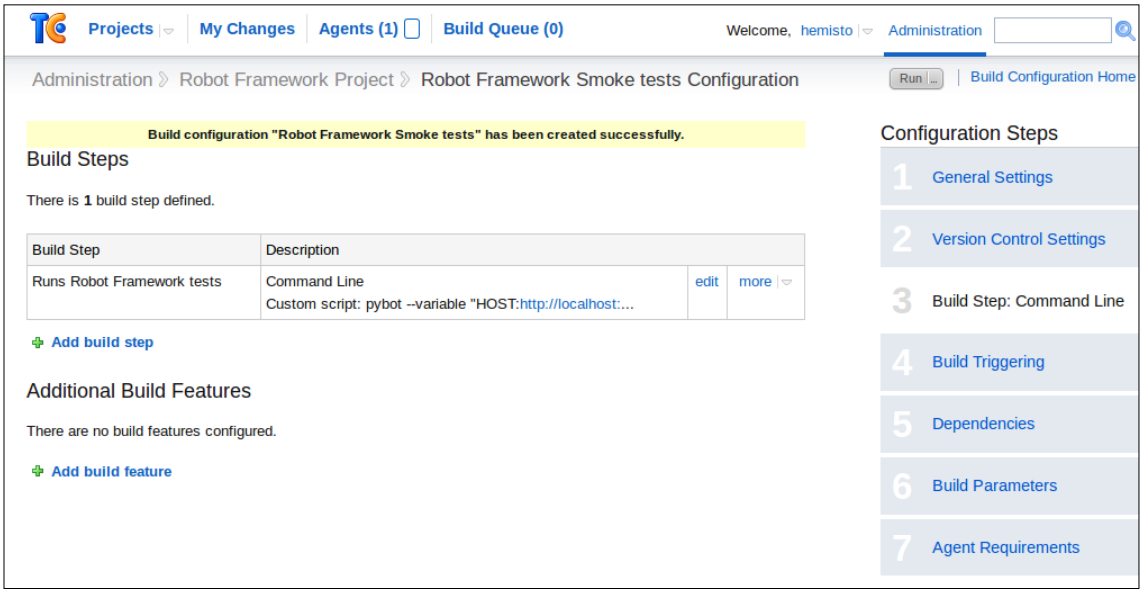
11. Check "Clean all files before build" to make TeamCity remove all files before building.
12. Click "Add Build Step >>"
13. Select Runner type to Command Line
14. Fill in Step name. For example “Runs Robot Framework tests”
15. Set Run to Custom script



16. Modify script below to run Robot Framework tests

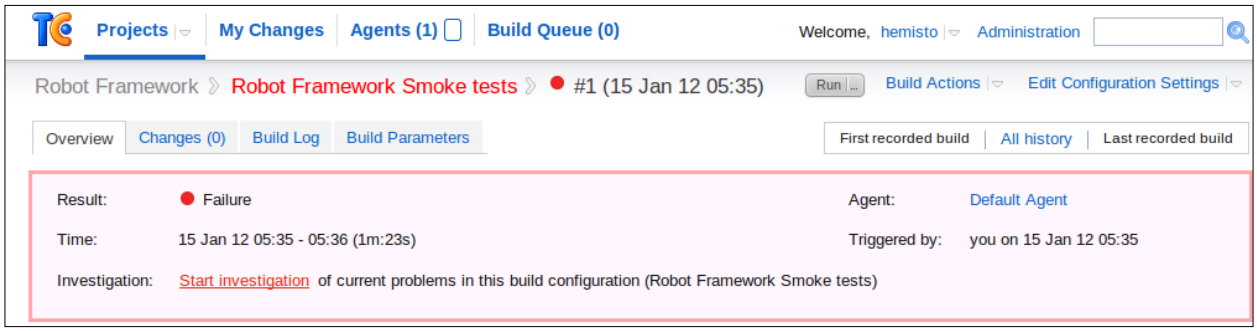
```
pybot --variable "HOST:http://localhost:4450/" --outputdir output --
xunitfile my_junit.xml my_test.txt

--variable "HOST:http://localhost:4450/" will pass Selenium Server or
Selenium Grid address to Robot Framework.
--outputdir output will change output directory what Robot Framework
use.
--xunitfile my_junit.xml will create test result in xunit-format.
my_test.txt is Robot Framework test file that will be run.
```

17. Click Save



18. Click Run from top right corner

The project will fetch your tests and a result is shown below



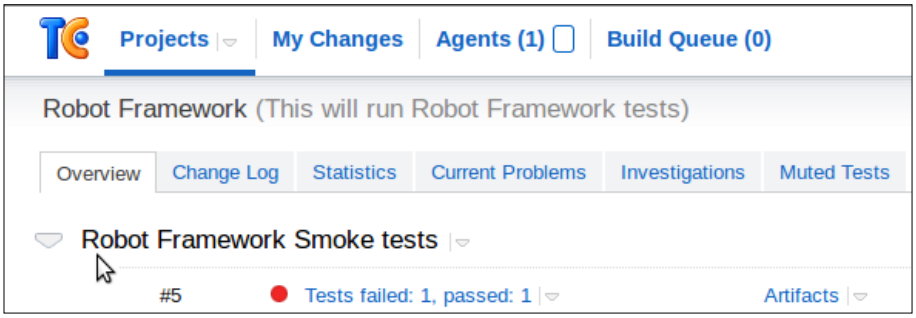
Agent requirements

If you have more than one agent you will need to add agent requirements. You need this to make sure TeamCity will pass your tests to an agent that is capable to run Robot Framework. This tutorial will not guide you in adding the agent. See http://confluence.jetbrains.net/display/TCD65/Agent+Requirements for more info.

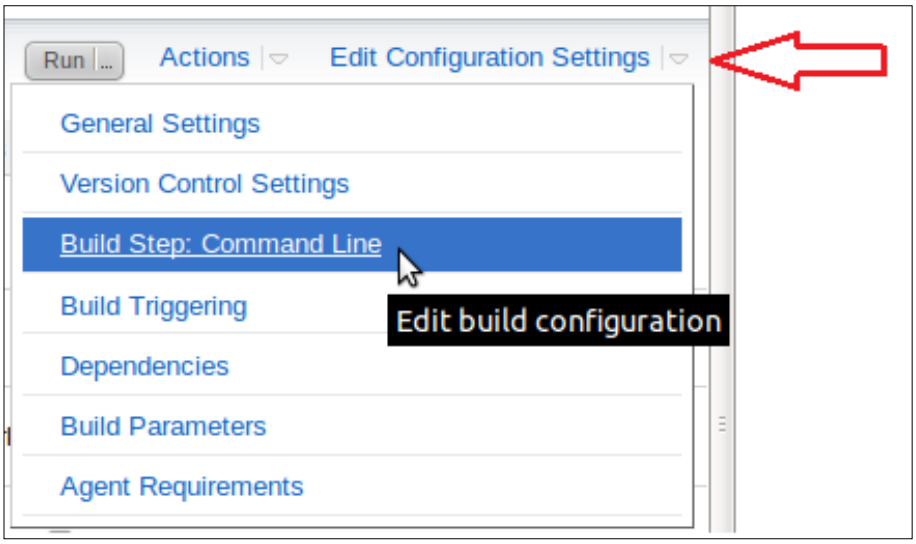
Parse test results

In this chapter you will create build feature that will read Robot Framework test results.

1. Navigate to your project
2. Select the build configuration you created before

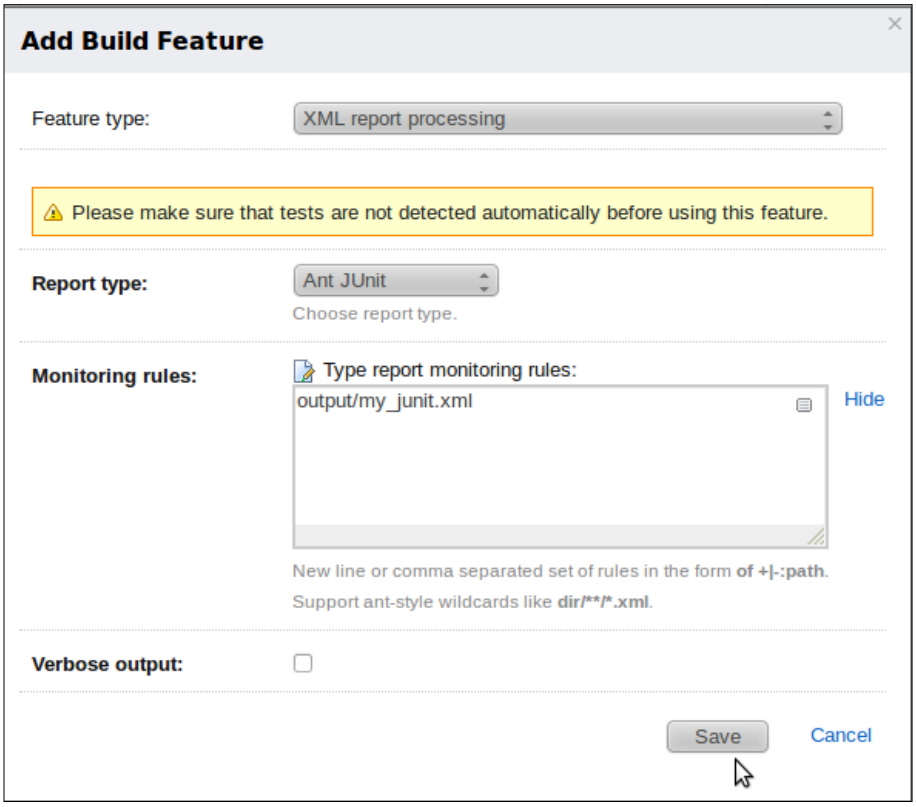


3. Mouse over arrow that is pointing down, it is next to Edit Configuration Settings
4. Choose "Build Step: <your_build_step_name_that_will_run_robot_framework>"



5. Click Add build feature
6. Select Feature type to XML report processing
7. Select Report type to Ant JUnit
8. Write xunit file name to Monitoring rules field

```
<directory>/<xunit_file>.xml
```

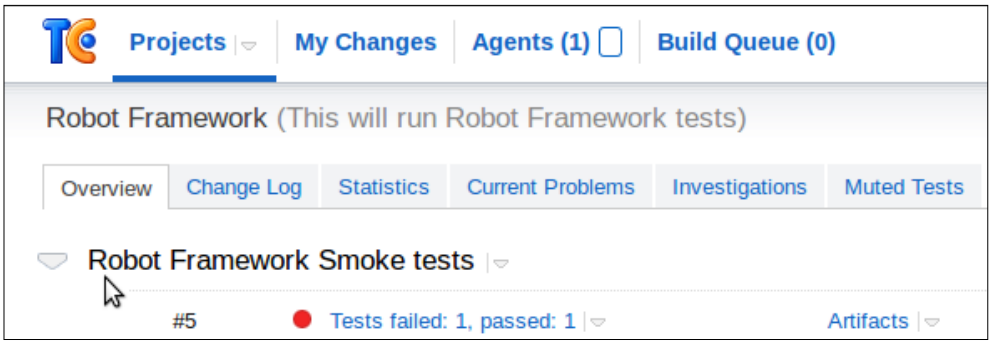


9. Click Save

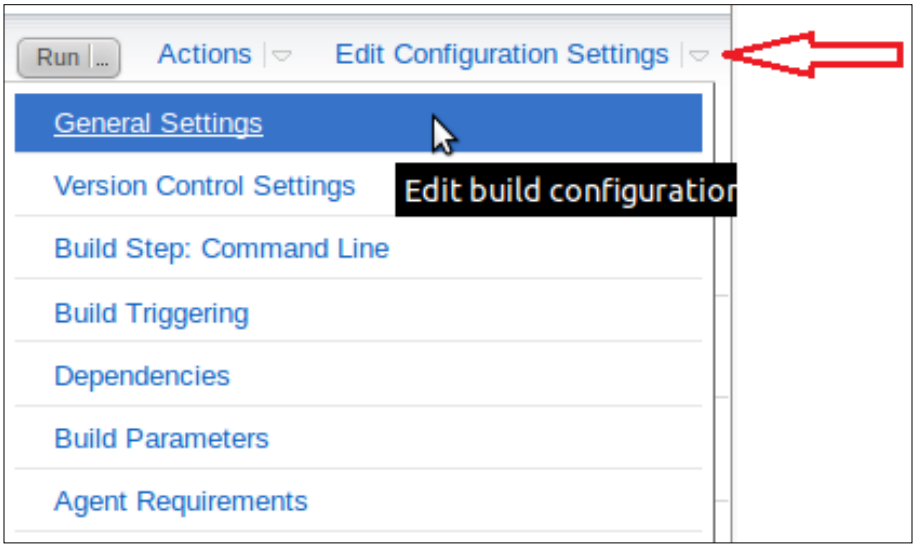
Add artifact

In this chapter you will add artifacts. Those artifacts are Robot Framework test result files.

1. Navigate to your project
2. Select the build configuration you created before

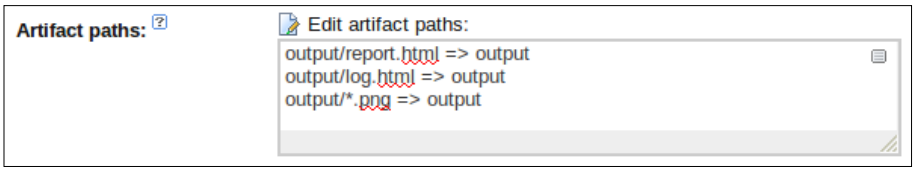


3. Mouse over arrow that is pointing down, it is next to Edit Configuration Settings
4. Choose General Settings



5. Write files you want to show to Artifact paths -field. Folder is Robot Framework output folder

```
<folder>/report.html => <artifact_name>
<folder>/log.html => <artifact_name>
<folder>/*.png => <artifact_name>
```

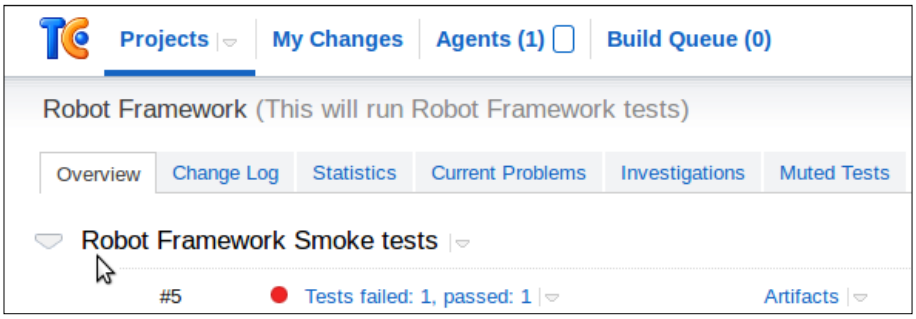


6. Click Save

Run build configuration and look results

Now everything has been done and you can run your build configuration.

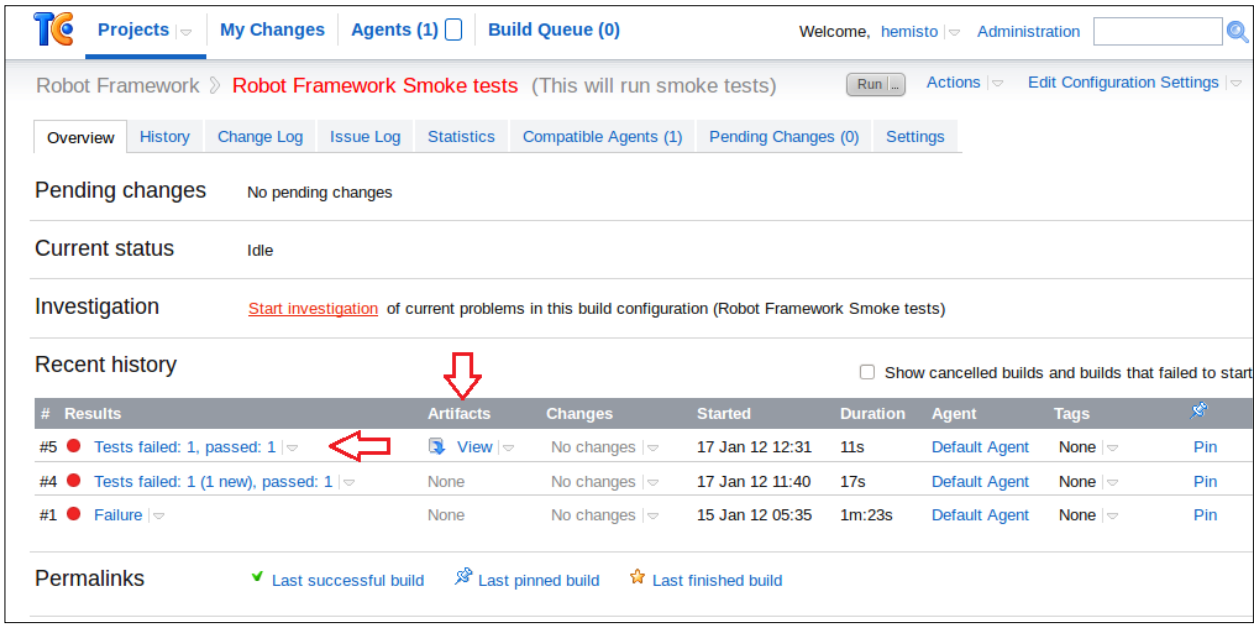
1. Navigate to your project
2. Select Build Configuration you created before

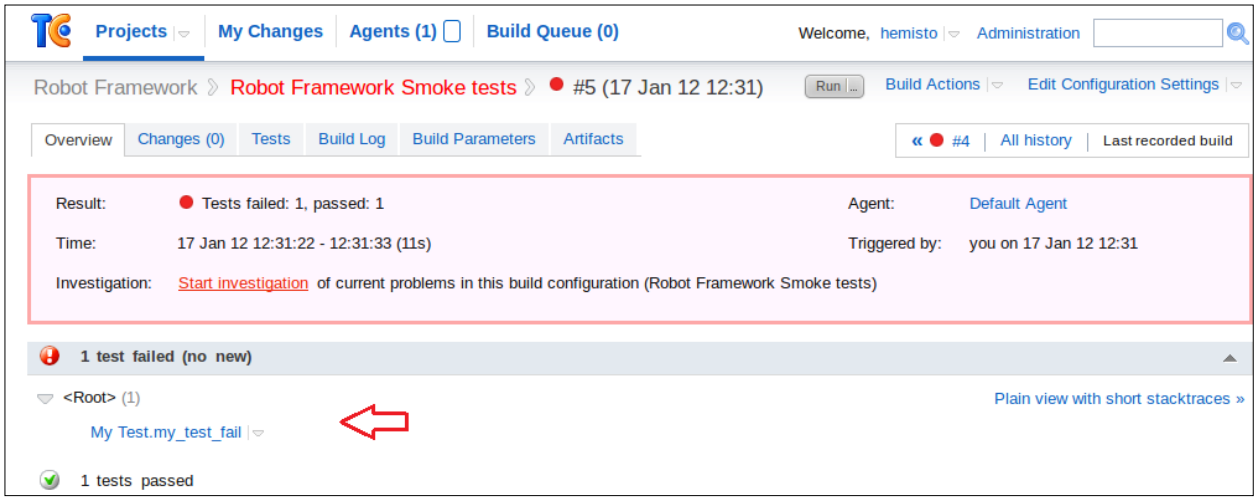


3. Click Run from top right corner

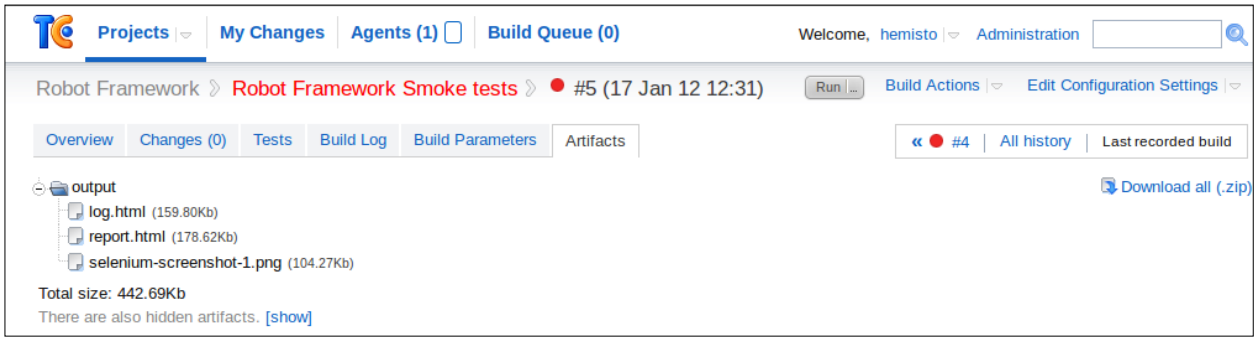
When the run is completed you can see test results from Build Configuration page. A background color tells you whether the plan succeeded (green) or failed (red). The center shows history information and you can

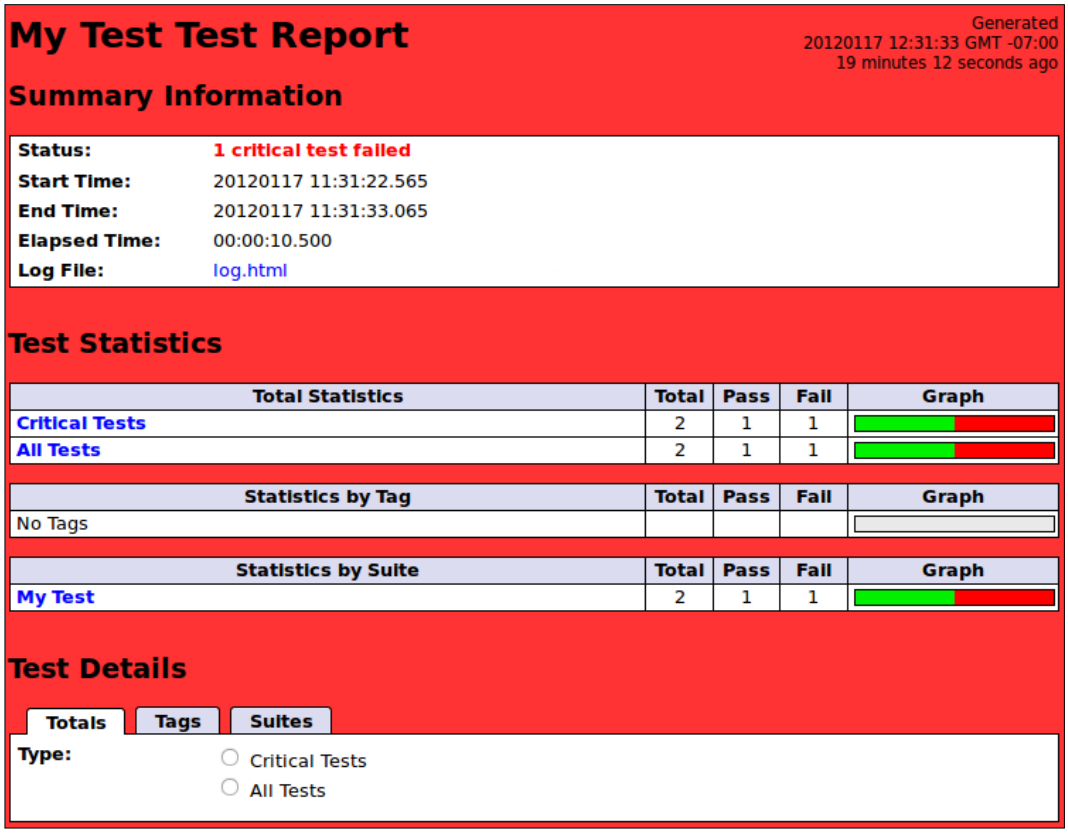
navigate to build summary tab by clicking Results column link. On the Summary tab you will see how many tests failed and how many passed. Reason why something failed can be seen on the bottom of the page





When you navigate to Artifacts you can open report and log files that Robot Framework generated. These are links to files that Robot Framework generated.





My Test Test Report

Generated
20120117 12:31:33 GMT -07:00
19 minutes 12 seconds ago

Summary Information

Status:	**1 critical test failed**
Start Time:	20120117 11:31:22.565
End Time:	20120117 11:31:33.065
Elapsed Time:	00:00:10.500
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	2	1	1	
All Tests	2	1	1	

Statistics by Tag	Total	Pass	Fail	Graph
No Tags				

Statistics by Suite	Total	Pass	Fail	Graph
My Test	2	1	1	

Test Details

Totals | Tags | Suites

Type:
- Critical Tests
- All Tests

Liite 6. Database Library. Robot Framework on Ubuntu 11.04

Database Library

Robot Framework on Ubuntu 11.04

Contents

About this guide

Robot Framework lists two different database libraries: The Robot Framework database library is written with Python and Robot Framework Database-library that is written using Java. In this guide you will learn to use Java based database library.

Database library is a Robot Framework test library that provides functionality for testing database contents. The library is written using JDBC (Java Database Connectivity) for connecting to database. The database library supports many different databases if a needed JDBC driver is available for the used database. (Robot Framework Database library, Java.)

Limitations:

- Works only with "jybot"-command
- Very limited amount of keywords (Robot Framework Database library, Java.)

Preconditions:

- Java installed. Java is needed to run and install Jython.
- Jython installed. The database library is written with Java so you will need to run Robot Framework tests with "jybot"-command.
- Robot Framework installed. The database library doesn't list what version is needed so you should be able to use what version you like.
- MySQL database is installed. The database can be on local computer or somewhere else.
    - The database should contain a table "users" and a couple of columns in it. The following script will create this table

```
CREATE  TABLE IF NOT EXISTS `users` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR(45) NOT NULL ,
  `password` VARCHAR(45) NOT NULL ,
  `extra_info` VARCHAR(255) NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB;
```

Download database library jar-file

Database library provides common keywords to test databases, for example *Execute SQL* and *Connect To Database*. The jar file can be downloaded from the database library homepage (https://github.com/ThomasJaspers/robotframework-dblibrary/wiki).

Download MySQL driver jar-file

Database library 1.0 requires MySQL JDBC (Java Database Connectivity) drivers but lists no version requirements. The JDBC drivers can be downloaded from http://dev.mysql.com/downloads/connector/j/ and more info about what the driver is can be found at http://dev.mysql.com/doc/refman/5.1/en/connector-j.html.

1. Download tar archive file

2. Extract files with

```
tar -zxvf mysql-connector-java-<version>.tar.gz
```

Write test file

Now you will have to write a simple test file. In this file you will need to:

1. Include a database library
2. Open a database connection
3. Execute a simple query
    a. For example query for that the table *"users"* exists
4. Close the connection

A test that will verify table "users" exists

```
*** Settings ***
#Include a database library
Library   org.robot.database.keywords.DatabaseLibrary

#Open a database connection
Suite Setup        Connect To Database    com.mysql.jdbc.Driver
jdbc:mysql://<MySQL server>/<database>    <username>    <password>

#close the connection
Suite Teardown     Disconnect from Database

*** Variables ***

*** Test Cases ***
check_table_exists
   #query that the table "users" exists
   Table Must Exist   users

*** Keywords ***
```

Run tests

Running tests is simple. First you will add both jar files (JDBC drivers and database library) to the Java classpath variable. Then you will execute tests using the jybot command. Notice that you will have to add the jar files to Java classpath variable every time you start new terminal. You can find more information at http://www.linuxheadquarters.com/howto/classpath.shtml.

1. Change terminal rights to super user. Notice that you will need to use the "sudo su" command only at the first time because then jybot will process some jar files.

```
sudo su
```

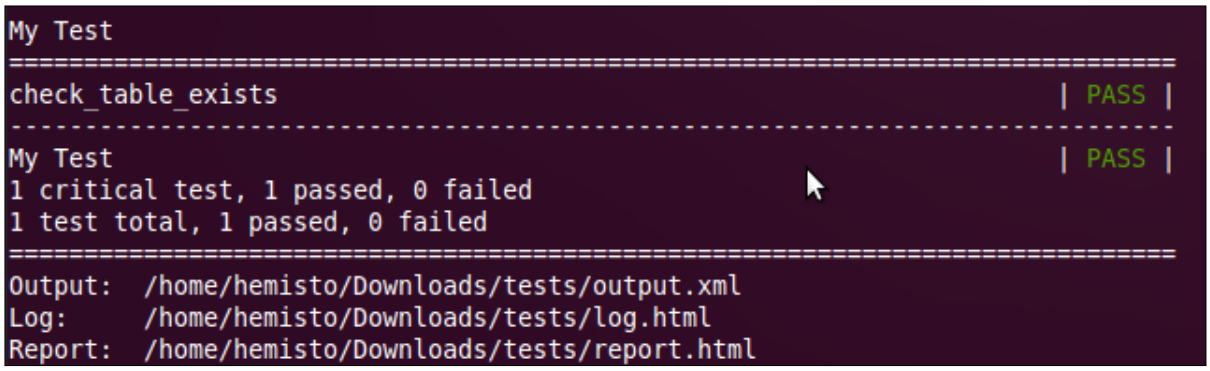
2. Add the jar files to Java classpath

```
export CLASSPATH=<path_to_folder_where_jar-file_is>/dblibrary-
1.0.jar:<path_to_folder_where_jar-file_is>/mysql-connector-java-
<version>-bin.jar
```

3. Run a test file

```
jybot <test_file>
```

If everything went smoothly Robot Framework should print something like



Useful keywords

In this chapter you will learn to use some useful database commands: how to establish database connection, insert data, make queries and remove data. The database library documentation for more information can be found at http://thomasjaspers.github.com/robotframework-dblibrary/DatabaseLibrary.html

Connection

Connect To Database keyword will establish a connection to the database. The keyword takes four parameters: driver, connection string, username and password. If you want to use different database you will need to change driver and the connection string. According to the database library documentation a database connection should be established during the Suite Setup phase. Suite Setup is executed before any test case.

Example:

```
Connect To Database    com.mysql.jdbc.Driver    jdbc:mysql://localhost/db
user    password
```

The ***Disconnect from Database*** keyword will release established database connection. The keyword will also log any SQL warnings that might have occurred while connected.

Example:

```
Disconnect from Database
```

Insert data

The ***Execute SQL*** keyword executes given SQL command. The keyword takes only one parameter and this parameter is SQL command that will be executed. Notice that this will **NOT** return anything. If you want to insert data and later remove it with unique id number, you have to use tricks. First you will have to insert

unique data to database for example current time. Next you will query unique number from database using the current time value you inserted earlier. Finally you will modify the current time value to original value.

Example:

```
#ask current time from system
${current_time}=   Get Time

#insert test data
Execute SQL    INSERT INTO users ( name, password ) VALUES
( '${current_time}', 'password')

#query id using current time value
${id}=   Read single Value from Table   users   id
name='${current_time}'

#modify current time value to original value
Execute SQL   UPDATE users SET name='username' WHERE id=${id}
```

Query

The ***Check Content For Row Identified By Where Clause*** keyword will check that a database row contains a proper content. The keyword takes four parameters: column names (separated by ','), expected values (separated by '|'), database table and where clause.

Example:

```
Check Content For Row Identified By Where Clause   name,password
admin|admin   users   id=2
```

The ***Read Single Value From Table*** keyword will read single value from a given table. The keyword takes three parameters: table name, column name and where clause. Notice that if more than one row is found keyword will return an error.

Example

```
Read single Value from Table   users    name   id=2
```

The ***Table Must Be Empty*** keyword checks that a table doesn't contain any rows. The keyword takes only one parameter that is table name.

Example:

```
Table Must Be Empty   users
```

The ***Table Must Exist*** keyword will query that a specified table exists. The keyword takes only one parameter that is table name.

Example:

```
Table Must Exist   users
```

Remove data

The ***Delete All Rows From Table*** keyword will remove all rows from a specified table. The keyword takes only one parameter that is table name.

Example:

```
Delete All Rows From Table   texts
```

The ***Execute SQL*** keyword can be used to remove a specific row from database. A common situation is that you want to remove a row by id.

Example:

```
Execute SQL    DELETE FROM users WHERE id = 2
```

Links

Robot Framework Database library, Java. https://github.com/ThomasJaspers/robotframework-dblibrary/wiki

Robot Framework Database library, Python. http://franz-see.github.com/Robotframework-Database-Library/

Robot Framework database library documentation. http://thomasjaspers.github.com/robotframework-dblibrary/DatabaseLibrary.html

MySQL JDBC drivers. http://dev.mysql.com/doc/refman/5.1/en/connector-j.html