Touchstone Documentation

Release 6.4.0

AEGIS.net, Inc.

Feb 21, 2025

CONTENTS

1 Introduction **1**

2 Registration and Login **3**
2.1 Register . . . 3
2.2 Membership . . . 5
2.2.1 New Organization . . . 5
2.2.2 Become a member . . . 6
2.2.3 Approving Membership . . . 7
2.3 Subscribe . . . 8
2.4 Login . . . 10
2.5 Email Addresses . . . 11
2.6 FAQ . . . 14

3 Test Systems **19**
3.1 Create Test System . . . 19
3.2 Capability Statement . . . 27
3.3 Test System List . . . 27
3.4 FAQ . . . 30

4 Executing Tests **31**
4.1 Creating Test Setup . . . 31
4.1.1 Repeated Variables . . . 34
4.2 Dynamic Fixtures . . . 35
4.2.1 Test Setup . . . 36
4.2.2 Conformance Testing . . . 39
4.3 Test Executions . . . 39
4.4 OAuth2 Test Executions . . . 41
4.4.1 Static Token . . . 41
4.4.2 Dynamic Token . . . 42
4.4.2.1 Authorization Code . . . 42
4.4.2.2 Client Credentials . . . 45
4.4.2.3 JWT Assertion . . . 46
4.5 Test Execution Results . . . 47
4.6 FAQ . . . 49

5 Multi-Profile Testing **51**
5.1 Validator Testing . . . 51
5.1.1 Uploading . . . 52
5.2 Updating Validators . . . 54
5.3 Viewing Validators . . . 56

5.3.1 To inspect the contents of a Validation Package . . . 57
5.4 FAQ . . . 57

6 Org Groups **59**
6.1 Creating Org Group . . . 59
6.2 Joining Org Group . . . 60
6.3 Leaving Org Group . . . 61
6.4 Access . . . 62
6.4.1 Test System access . . . 62
6.4.2 Test Definition access . . . 63
6.4.3 Test Results access . . . 66

7 Conformance Testing **69**
7.1 Conformance Suites . . . 69
7.1.1 Listing . . . 70
7.1.2 Launching Executions . . . 72
7.2 Current . . . 72
7.2.1 Single Test Group . . . 72
7.2.2 Multi Test Group . . . 83
7.3 Results Summary . . . 90
7.4 Publishing Results . . . 101
7.5 FAQ . . . 108

8 Client/Peer-to-Peer Testing **113**
8.1 Launching Executions . . . 113
8.2 Execution Matching . . . 120
8.3 Exchanges Screen . . . 124
8.3.1 Common Errors . . . 125
8.4 FAQ . . . 128

9 TestScript Authoring **131**
9.1 FHIR TestScript . . . 131
9.2 Upload on UI . . . 132
9.3 TestScript Editor . . . 137
9.3.1 Download . . . 137
9.3.2 Start TestScript Editor . . . 138
9.3.3 Create TestScript Project . . . 139
9.3.4 Create a Test Group . . . 141
9.3.5 Create a TestScript . . . 143
9.3.6 Code Completion . . . 148
9.3.7 Touchstone Integration . . . 149
9.3.7.1 Upload to Touchstone . . . 149
9.3.8 Simplifier Integration . . . 159
9.3.8.1 Simplifier Preferences . . . 159
9.3.8.2 Create New Project . . . 160
9.3.8.3 Download from Simplifier . . . 163
9.3.8.4 Upload to Simplifier . . . 164
9.3.9 Updating TestScript Editor . . . 166
9.3.10 References . . . 172
9.4 Best Practices . . . 172
9.4.1 Version Control . . . 172
9.4.2 Location . . . 172
9.4.3 Paths to resources . . . 173
9.5 Exclusions . . . 173
9.5.1 Props Location . . . 174

9.5.2 Props format . . . 174
9.5.3 Upload Exclusions . . . 175
9.5.4 Parsing Exclusions . . . 181
9.5.5 Validation Exclusions . . . 186
9.5.6 Conformance Exclusions . . . 187
9.6 Test Groups . . . 188
9.6.1 Access . . . 188
9.6.2 Deactivate . . . 191
9.6.3 Deletion . . . 194
9.7 Placeholders . . . 195
9.7.1 Predefined User Unique Data Values . . . 195
9.7.2 Functions for Dynamic Data Generation . . . 195
9.7.2.1 CURRENTDATE[TIME] . . . 196
9.7.2.2 DATE[TIME] . . . 196
9.7.2.3 UUID[-ST|-NODASH|-ST-NODASH] . . . 197
9.7.3 Usage . . . 197
9.8 Rule Authoring . . . 198
9.8.1 Rule Basics . . . 198
9.8.1.1 Definition . . . 198
9.8.1.2 Declaration . . . 202
9.8.1.3 Assertion . . . 202
9.8.1.4 Summary and Description . . . 203
9.8.1.5 Bindings . . . 208
9.8.2 Parameters . . . 210
9.8.2.1 Definition . . . 210
9.8.2.2 Supplying params . . . 211
9.8.2.3 Operator parameter . . . 213
9.8.2.4 Expected parameter . . . 215
9.8.3 Ruleset . . . 216
9.8.3.1 Definition . . . 217
9.8.3.2 Declaration . . . 220
9.8.3.3 Assertion . . . 223
9.8.3.4 Overriding rules . . . 223
9.8.4 Rules API . . . 224
9.8.4.1 Body . . . 224
9.8.4.2 Capability / Support . . . 226
9.8.4.3 Content-Type . . . 230
9.8.4.4 Header . . . 232
9.8.4.5 Minimum . . . 238
9.8.4.6 Path . . . 238
9.8.4.6.1 FHIRPath . . . 239
9.8.4.6.2 JSONPath . . . 246
9.8.4.6.3 NonFHIRPath . . . 254
9.8.4.6.4 XPath . . . 262
9.8.4.7 Profile . . . 268
9.8.4.8 Request Method . . . 271
9.8.4.9 Request URL . . . 273
9.8.4.9.1 Request URL Prefixes . . . 276
9.8.4.10 Resource . . . 278
9.8.4.11 Response Code . . . 280
9.8.5 Short-circuiting . . . 283
9.8.5.1 Failing . . . 283
9.8.5.2 Warning . . . 283
9.8.5.3 Skipping . . . 284

9.8.6 Multiple Assertions . . . 286
9.8.6.1 Short-circuiting when a request or response assertion fails . . . 286
9.8.6.2 Continuing rule-execution when a request or response assertion fails . . . 287
9.8.7 Rule Outputs . . . 288
9.8.7.1 Assertions on one Rule Output . . . 290
9.8.7.2 Assertions on multiple Rule Outputs . . . 291
9.8.7.3 Rule Output when outputting a Fixture . . . 291
9.8.8 XSLT and Schematron . . . 293
9.8.8.1 Declarations . . . 293
9.8.8.2 Header assertions . . . 293
9.8.8.2.1 Groovy . . . 293
9.8.8.2.2 XSLT . . . 294
9.8.8.2.3 Schematron . . . 295
9.8.8.3 Payload assertions . . . 296
9.8.8.3.1 Groovy . . . 296
9.8.8.3.2 XSLT . . . 296
9.8.8.3.3 Schematron . . . 297
9.9 OAuth2 Capabilities . . . 298
9.9.1 Fixture ID's . . . 298
9.9.2 Base64Encoding . . . 299
9.9.3 PKCE Functionality . . . 299
9.10 Bulk Data Capabilities . . . 301
9.10.1 NDJSON File Evalutation and Validation . . . 301
9.10.2 NDJSON Assertion Prefix . . . 301
9.11 TestScript Extensions . . . 302
9.12 FAQ . . . 302

10 Conformance Suite Authoring 305
10.1 Server Suites . . . 305
10.2 Client Suites . . . 313
10.3 Categorization . . . 321
10.3.1 Overview . . . 321
10.3.2 Definition . . . 331
10.3.2.1 Root Node . . . 331
10.3.2.2 Leaf Nodes . . . 332
10.3.2.3 Parent Nodes . . . 335
10.3.2.4 Node Deletion . . . 336
10.3.2.5 Paths . . . 339
10.3.2.5.1 FHIR . . . 339
10.3.2.5.2 CDS Hooks . . . 342
10.4 Resource Ownership . . . 342
10.5 Versioning . . . 343
10.5.1 Version Increments . . . 343
10.5.2 No Version Increments . . . 345
10.6 Best Practices . . . 346
10.6.1 Suite Versioning . . . 346
10.6.2 Anchor Systems . . . 347
10.6.3 Publishing . . . 347
10.6.4 Deletion . . . 347
10.7 FAQ . . . 349

11 Continuous Integration 351
11.1 API . . . 351
11.1.1 Authentication . . . 352

11.1.2 Launching Executions . . . 353
11.1.3 Retrieve Execution status . . . 353
11.1.4 Retrieve Execution Detail . . . 354
11.1.5 Retrieve Script Exec Detail . . . 354
11.1.6 Pseudo code . . . 355
11.1.7 Retrieve FHIR Test Report . . . 356
11.1.8 OAuth2 Static Token . . . 357
11.1.9 OAuth2 Grant Flow . . . 358
11.2 Jenkins Integration Example . . . 360
11.2.1 Groovy Script Definition . . . 360
11.2.1.1 Authentication . . . 361
11.2.1.2 Execute Test . . . 361
11.2.1.3 Wait for Test Execution Completion . . . 362
11.2.2 Jenkins Job Configuration . . . 364
11.2.2.1 Post Steps Initial Configuration . . . 364
11.2.2.2 Execute system Groovy script . . . 364
11.2.2.3 Groovy script file . . . 365
11.2.2.4 Advanced - Expand Classpath . . . 365
11.2.2.5 Advanced - Enter Classpath . . . 365

12 Downloads **369**

13 Release Notes **371**
13.1 Touchstone . . . 371
13.2 TestScript Editor . . . 402

CHAPTER

ONE

INTRODUCTION

The Touchstone Project is an Infrastructure as a Service (IaaS) and Testing as a Service (TaaS) Open Access Solution for health information exchange. Touchstone offers over 1500 tests in an easy-to-use system for determining a test system's conformance and interoperability against published specifications, standards, and profiles, including templates and implementation guides.

The Touchstone Project…

- allows for automated, internet-based interoperability testing against the HL7 FHIR specifications and standards.
- tests interoperability with other FHIR Server and FHIR Client implementations.
- has been engineered from the ground up to leverage the new FHIR TestScript resource.
- is a blend between Test-Driven-Development (TDD) methodologies and Natural Language Processing (NLP) test scripts.
- has been featured at HL7 FHIR Connectathons and is being leveraged in a continuous testing environment by numerous leading HL7 FHIR implementers.
- plays an active role in the HL7 Conformance Testing community, the HL7 Argonaut Project, and the HSPC Implementation community.

Features include…

- self-registration of user accounts and organizations.
- ability for users to execute FHIR test scripts against test systems with Touchstone serving as the initiator of message exchanges.
- ability for users to initiate message exchanges from their test systems against other peer test systems with Touchstone serving as the intermediary.
- ability to drill down to individual operations and assertions in test execution results.
- ability to save configured sets of test scripts as named "test setups" for re-execution.
- controlled access to test scripts, test systems, and test results at the user, organization, and organization group levels.

CHAPTER

TWO

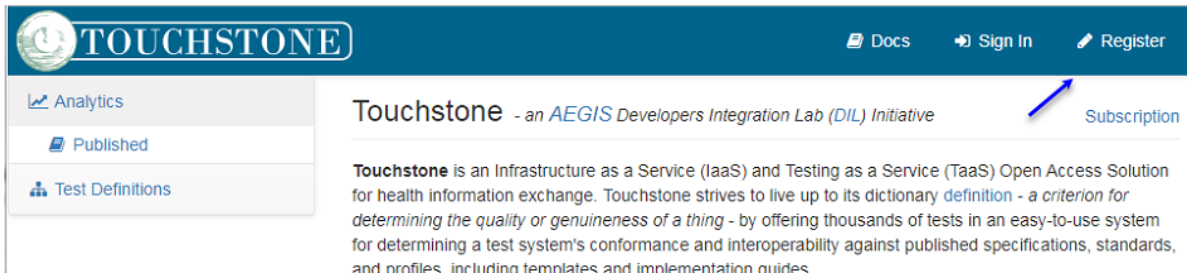
REGISTRATION AND LOGIN

2.1 Register

Touchstone Registration - Video Walkthrough

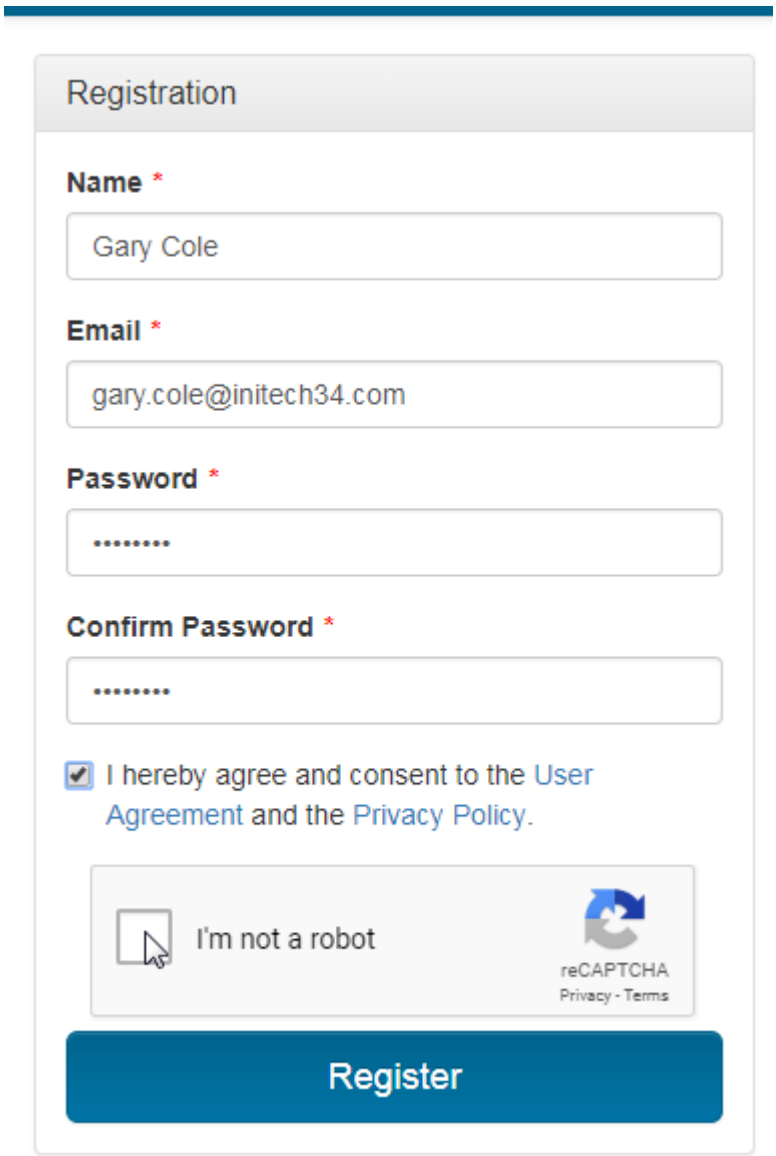
You can register into the system by taking the following steps:

1. Click the Register link:



2. Fill in the fields and hit the Register button.

Note: Password must be at least 8 characters long and must contain number, upper case letter, and special character (@#$%^&+=).



Warning: Please avoid using a fictitious email address as important sign-in information will be emailed to you at this address. You will need that information in the future to reset your password. Look in your Spam folder if you are unable to find emails sent from Touchstone_Support@aegis.net.

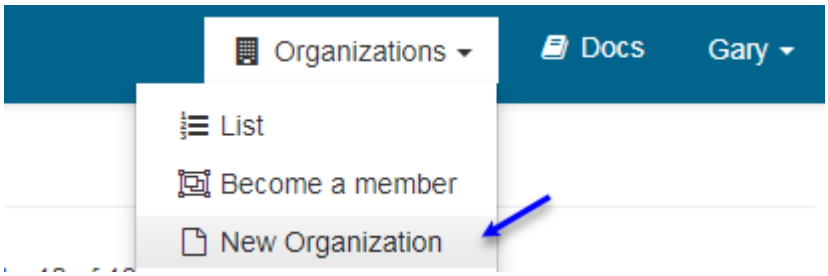
2.2 Membership

To execute tests in Touchstone, you must either create an organization or become a member of an existing organization. If your organization does not yet exist in Touchstone, please feel free to create one.

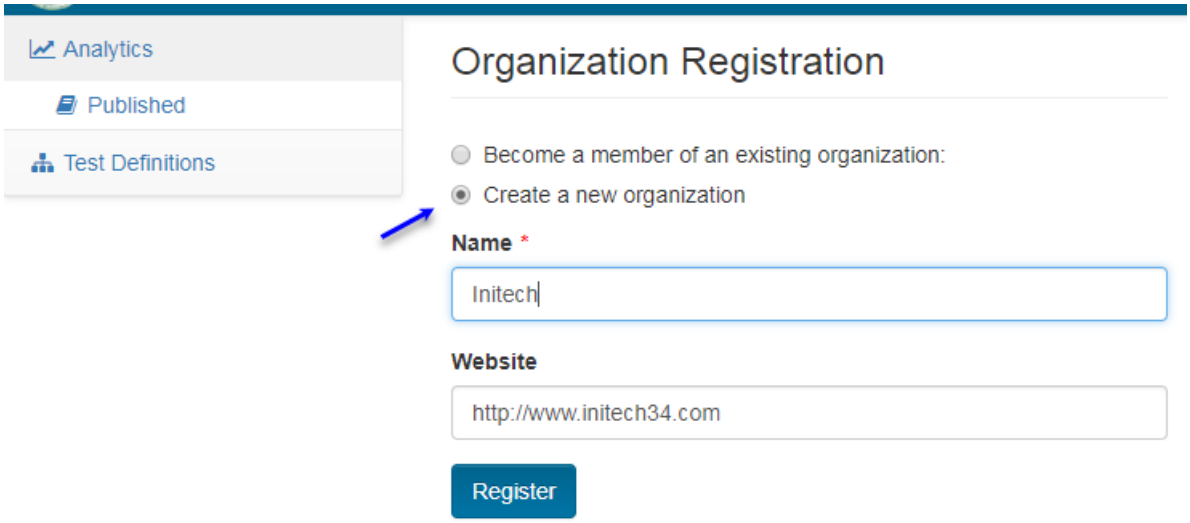
2.2.1 New Organization

If you are the first user from your organization to register into Touchstone, you can take the following steps:

1. Click on the New Organization link in the top menu:

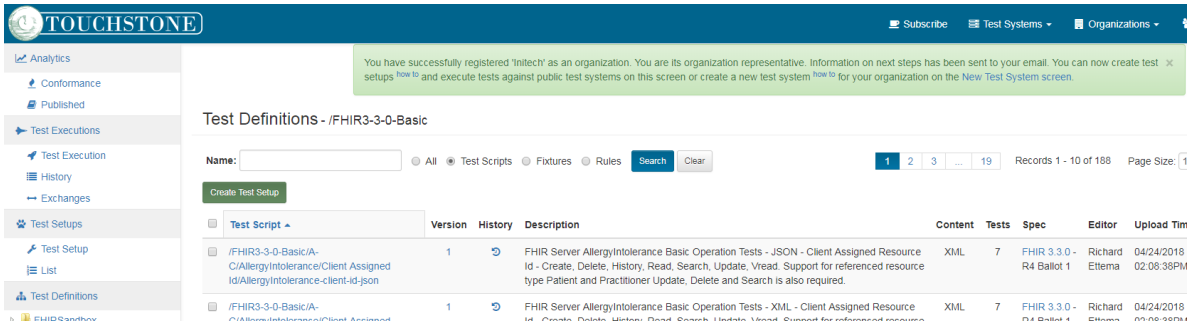


2. Select `Create a new organization` if it's not already selected. Fill in the fields and hit Register:



Note: Website is the organization's website and not the Base URL of a test system.

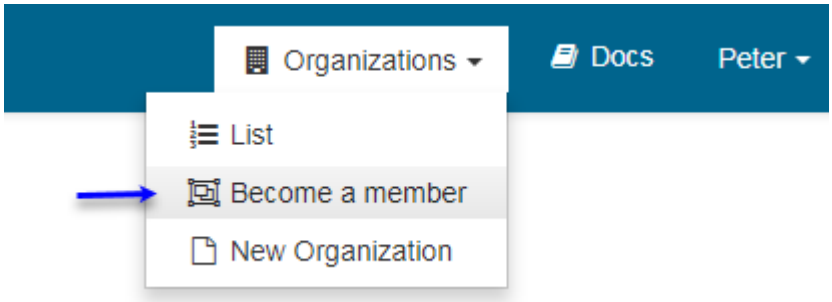
3. Congratulations! You should see the screen below. You are now able to execute tests in Touchstone. You are the sole representative of this organization. Other users can request to become a member of your organization. You will be notified of such requests. You can approve or reject their requests under the Users section.



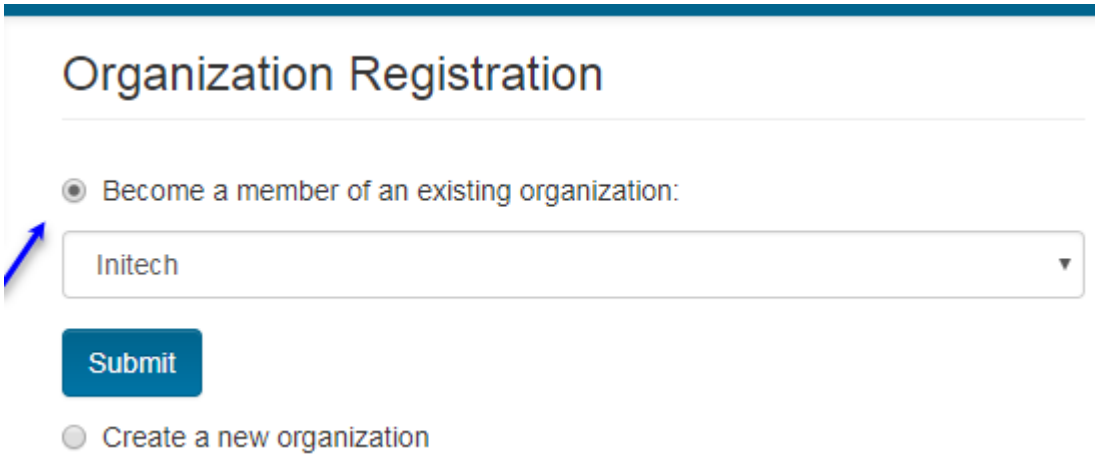
2.2.2 Become a member

If your organization already exists in the system, after going through the initial registration, you can take the following steps:

1. Click on the Become a member link in the top menu:

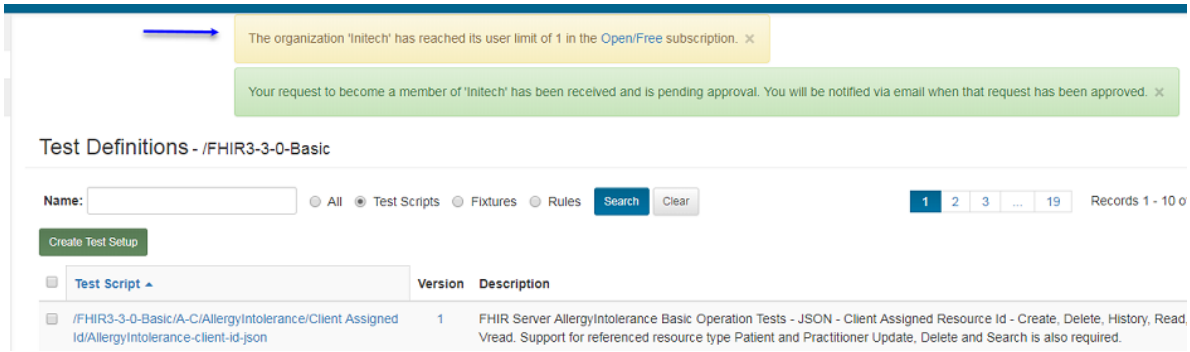


2. Click the `Become a member of an existing organization` option. Select the organization that you wish to become a member of and hit Submit:



After your request has been submitted, you will need to wait for approval before you can execute tests in the system. You will be notified of approval (or rejection) via email. Check your Spam folder in your email system in case the emails get directed there.

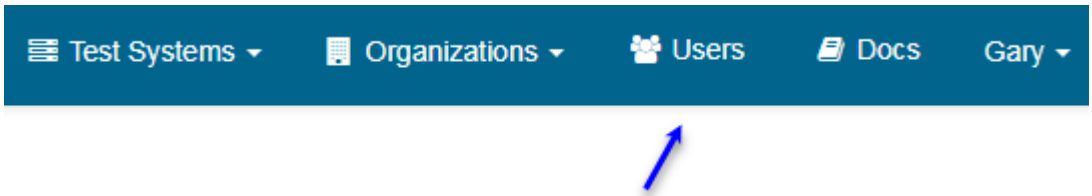
If you get the warning below, then the Org Rep of the organization has to upgrade the subscription before being able to approve your subscription.



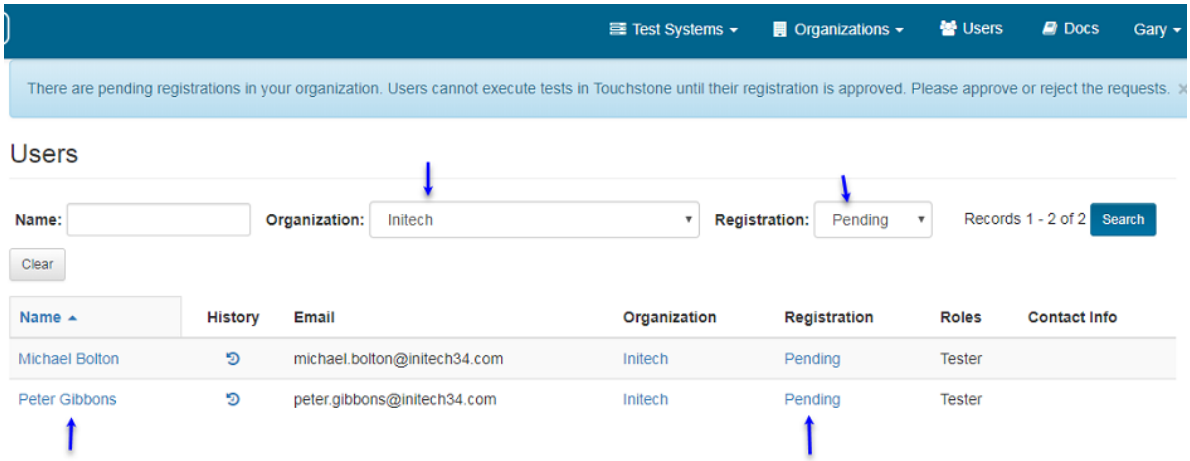
2.2.3 Approving Membership

If you are one of the organization reps of your organization, you will be notified (by email) of membership requests. Below are the steps you can take to approve or reject the user's request to become a member of your organization:

1. Click on Users in the top menu and filter by your organization:



2. Notice that there are two users with Pending registrations. These users are waiting for your approval before they can execute tests.

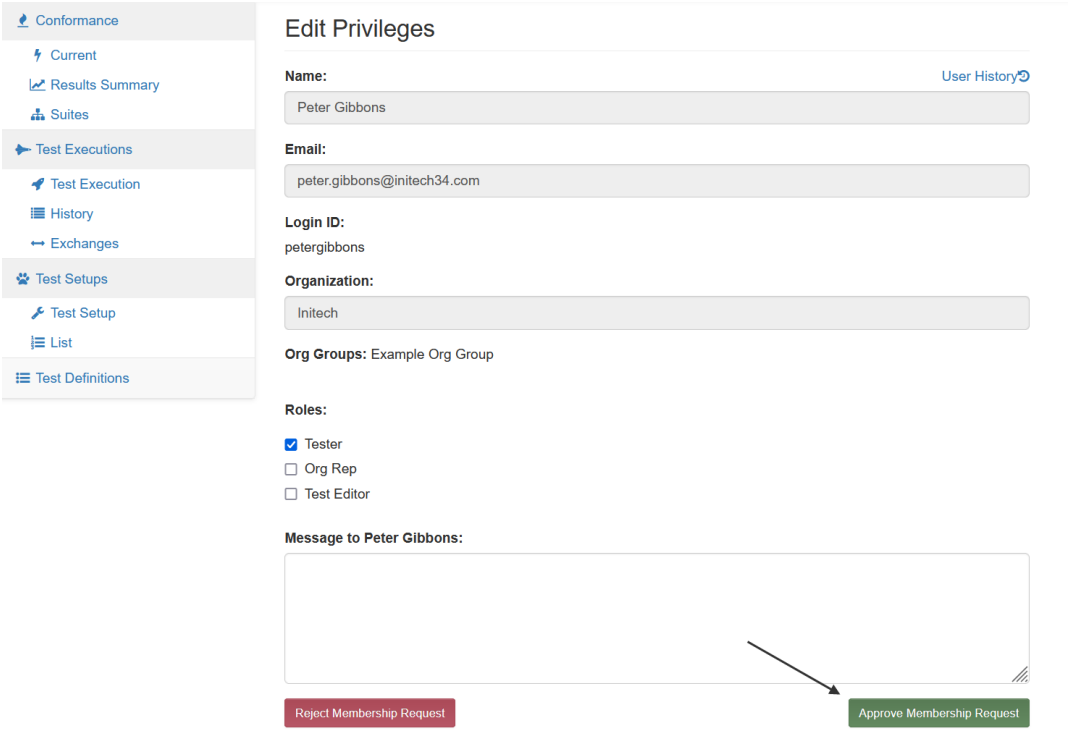


3. To change Peter's registration status, you can click either on the link `Peter Gibbons` or `Pending` as indicated above.
4. On the Edit Privileges screen, click on `Approve Membership Request` if you'd like to approve the user's registration or `Reject Membership Request` if you'd like to reject it.

> **Warning:** You do NOT need to give all your users the Org Rep role. Testers can create test systems and execute tests the same way as Org Reps.

Org Reps can perform the following functions that Testers cannot:

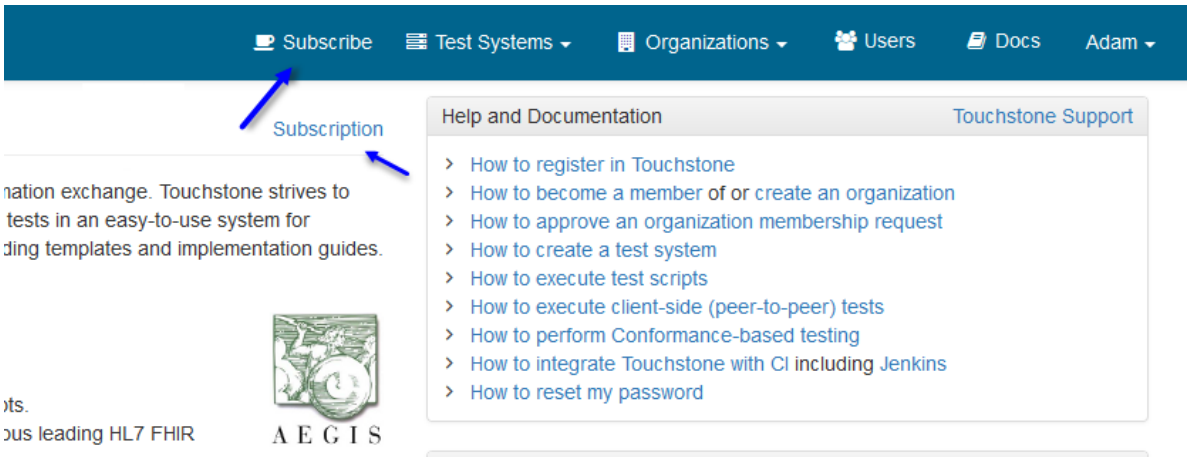
- Org Reps can approve and reject membership requests.
- They can publish test results that belong to the organization.
- They can submit requests on behalf of the organization for membership into Org Groups.
- They can view the roles and user history of other users within the organization.
- They can be contacted by site administrator for issues that pertain to the organization and its test systems.



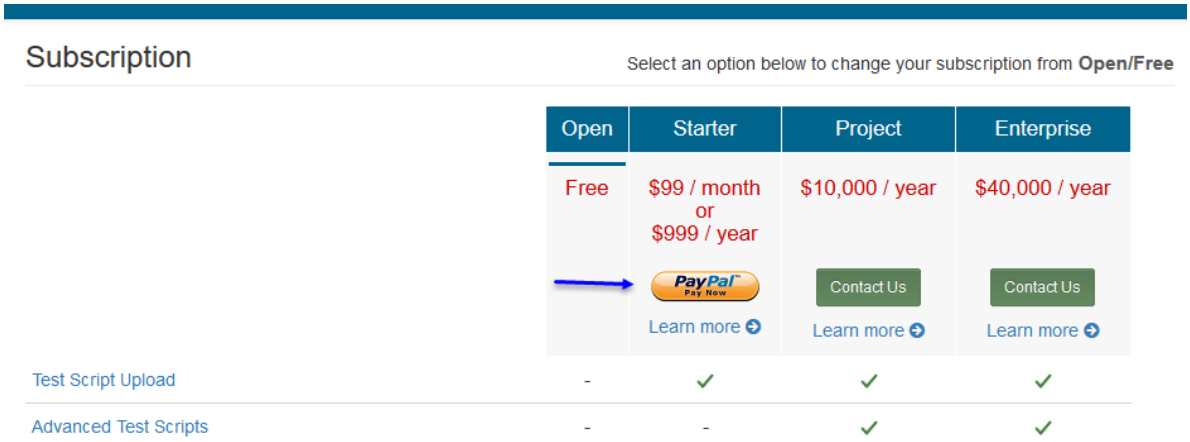
2.3 Subscribe

You can subscribe to to Starter level via PayPal:

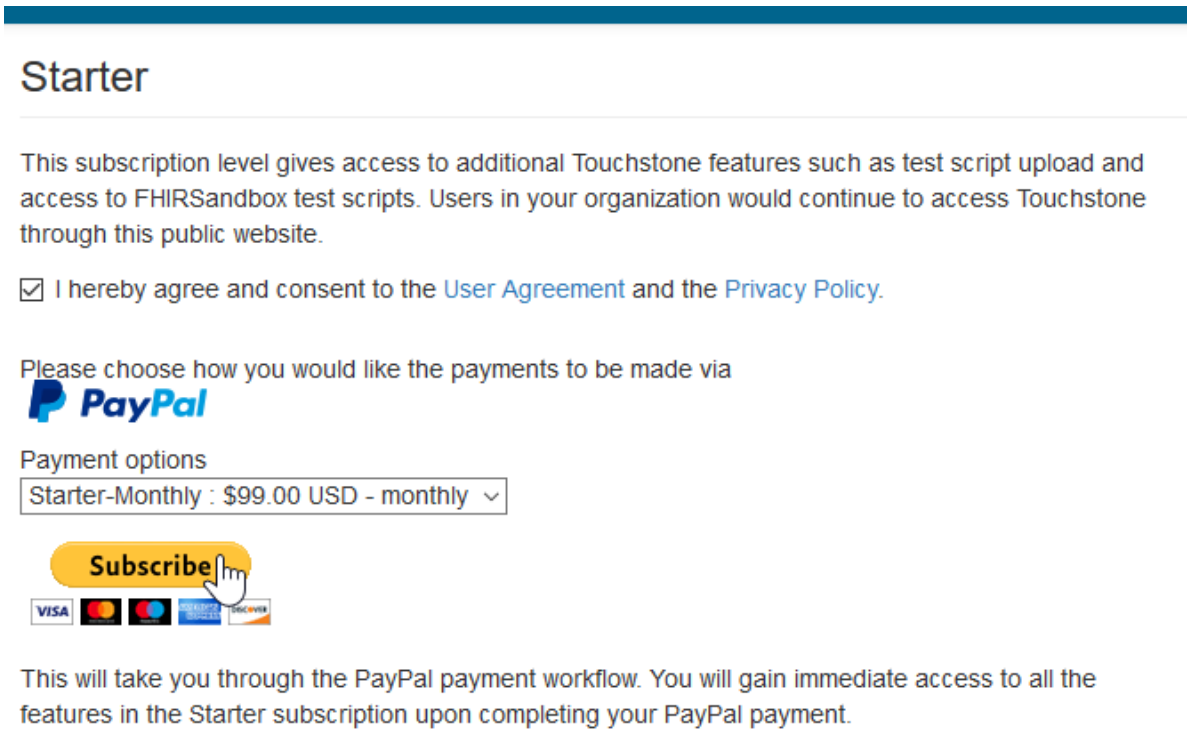
1. Click either the Subscription link on the home page or the Subscribe link on the banner:



2. Click the PayPal button under the Starter header:



3. Fill in the fields and click the PayPal Subscribe button.



4. Follow the standard PayPal payment procedure and you will be redirected back to Touchstone. Once this happens your Org will be automatically upgraded to a starter level subscription.

> **Warning:** The automatic upgrade may, in rare cases, not happen immediately. If you aren't upgraded immediately on redirect, you will be upgraded when Touchstone receives the notification from PayPal that the payment has been verified.

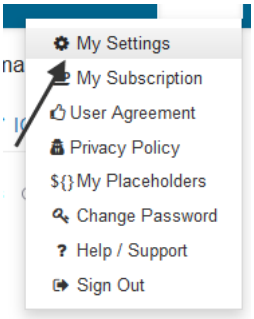
2.4 Login

As part of registration process the "Name" supplied will be converted to all lowercase, spaces removed, special characters removed and that will be stored as a Login ID. You can log into your Touchstone Account using your Primary Email OR generated Login ID.

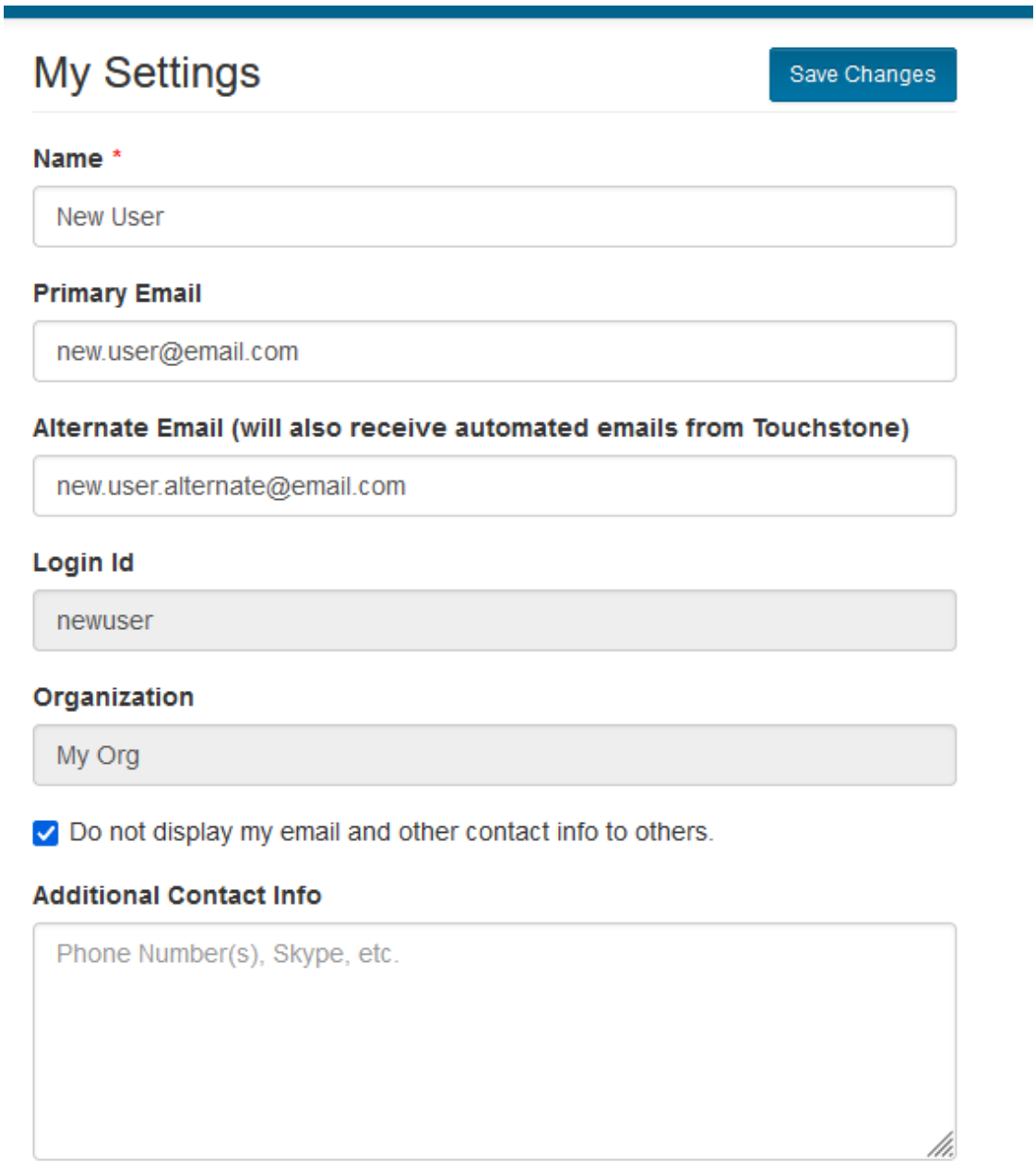
Steps to view Login ID in Touchstone:

1. Sign into your Touchstone account using your Primary Email.

2. After signing in, hover over your user name in the upper right corner of Touchstone. Select "My Settings" from the list.



On the "My Settings" screen you will find your Primary Email and Login ID, both valid as Touchstone login values.



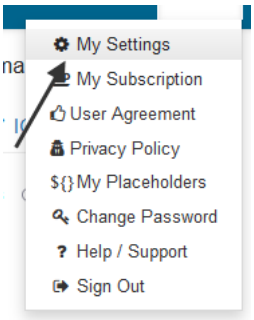
2.5 Email Addresses

As part of the registration process, the "Email" supplied was set to your Primary Email address. This email address can be used to login, and will also receive any automated emails that Touchstone sends out to you, the user. You do have the optional choice of having an Alternate Email address. If you choose to have an Alternate Email, it will also receive any automated emails from Touchstone that are sent to your Primary Email.

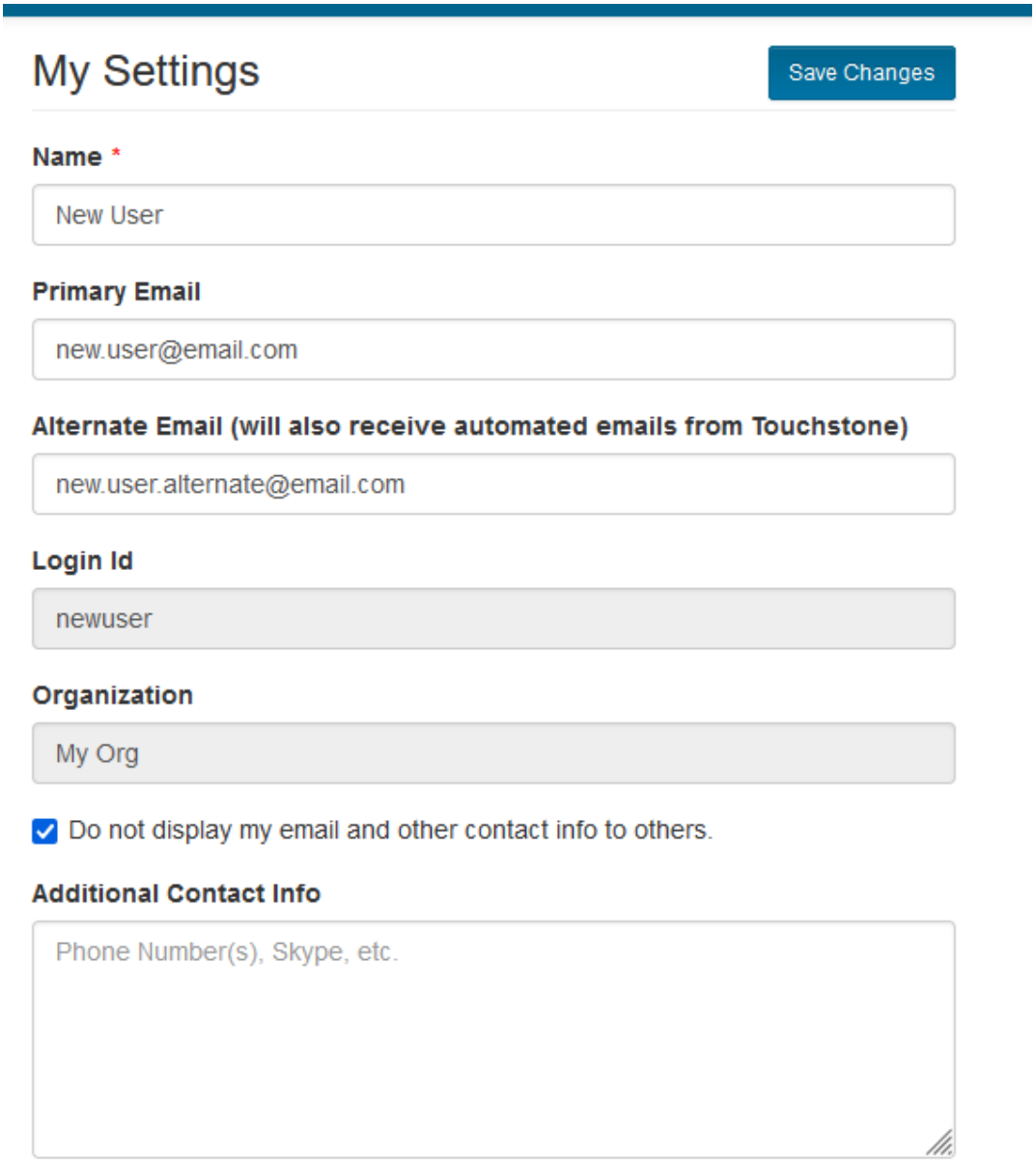
Steps to add an Alternate Email address to your profile:

1. Sign into your Touchstone account using your Primary Email.

2. After signing in, hover over your user name in the upper right corner of Touchstone. Select "My Settings" from the list.



3. On the "My Settings" screen you will find your Primary Email (required) and your Alternate Email (optional), both of which you can update if you choose to do so.

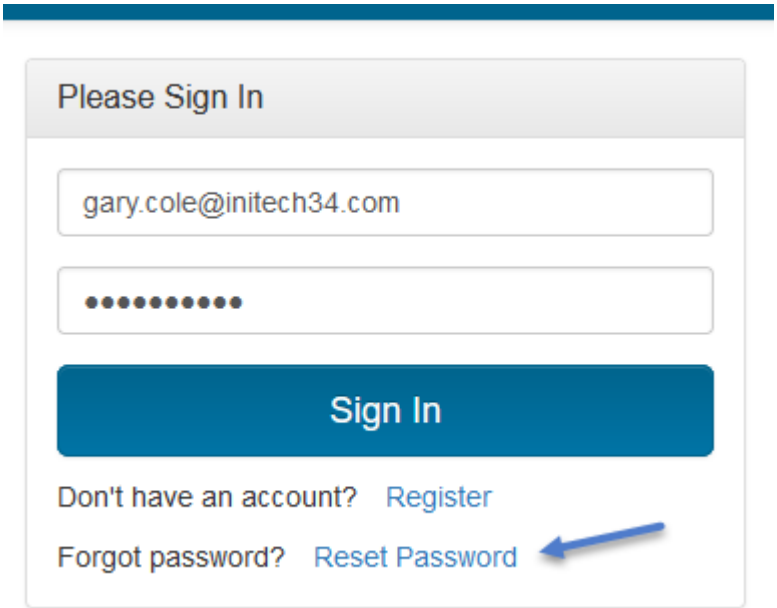


Warning: Your Primary Email and Alternate Email cannot be the same, and niether can be the same as any other Primary or Alternate Email used by other users in Touchstone.

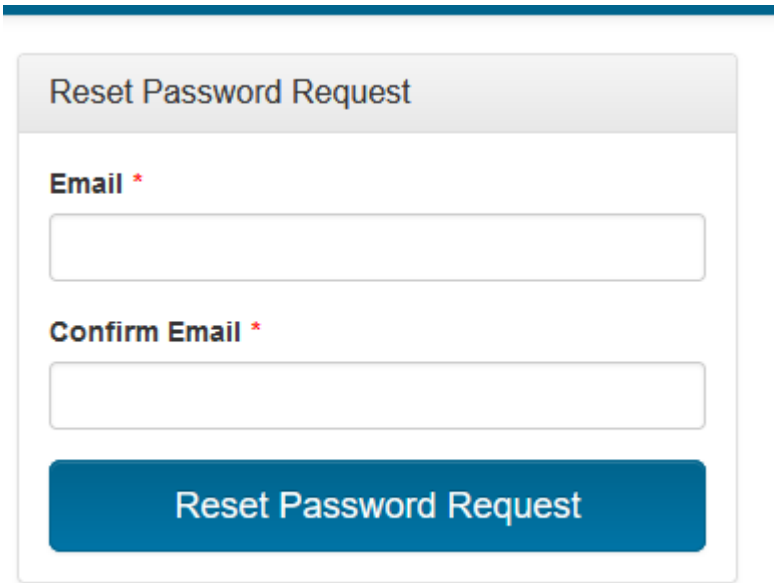
2.6 FAQ

1. How can I change my password?
   - If you do not remember your old password, you will need to go to Reset Password Request page:

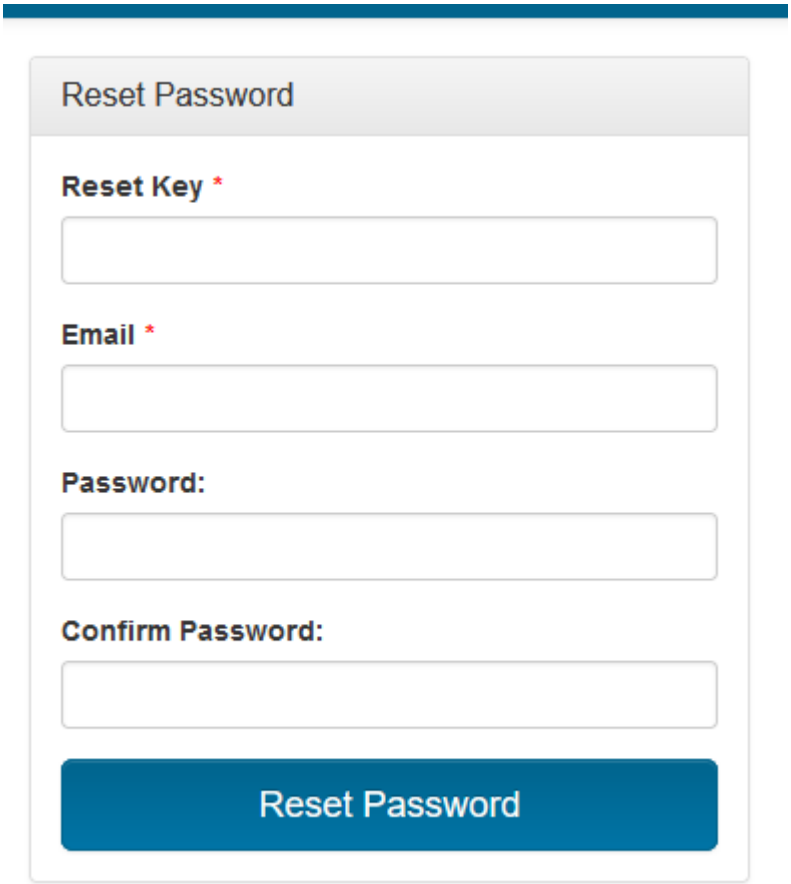
     A link to this page can also be found on the Login page



     Enter your email address and confirm your email address on the next screen and you should get a new email with your new `Reset Key` value.



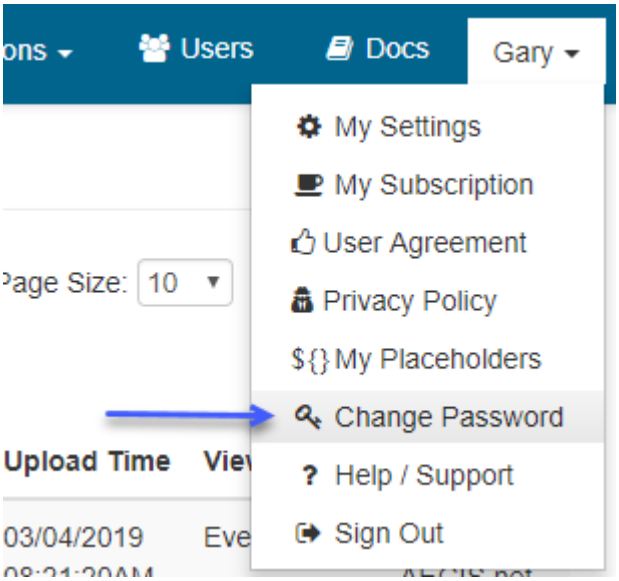
     This `Reset Key` will be used on the Reset Password page, with the same link being provided in the email with your `Reset Key`. On this page, enter your `Reset Key`, email address, new password, and confirm new password.



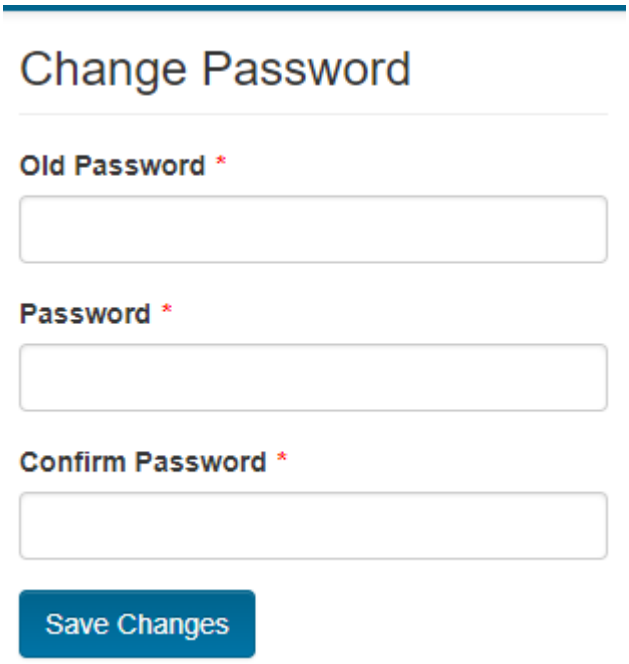
> **Warning:** Your new password cannot be the same as your current password, and it cannot be the same as the last 20 passwords that you have used.

If you're unable to find your Reset Key or cannot reset your password, please contact Touchstone_Support@aegis.net

- If you remember your old password, you can go to Change Password page:

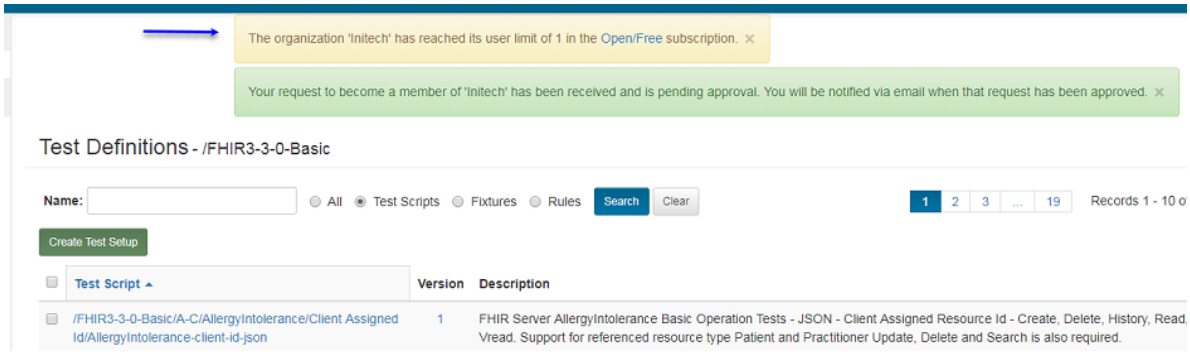


Enter your `Old Passowrd` on the next screen along with your new password and click on `Save Changes`:



2. I requested membership into an organization but am not getting approved? It is likely that the organization does not have the right subscription level to approve your registration.

   If you got the warning below during registration, then the Org Rep of the organization has to upgrade the subscription before being able to approve your subscription,



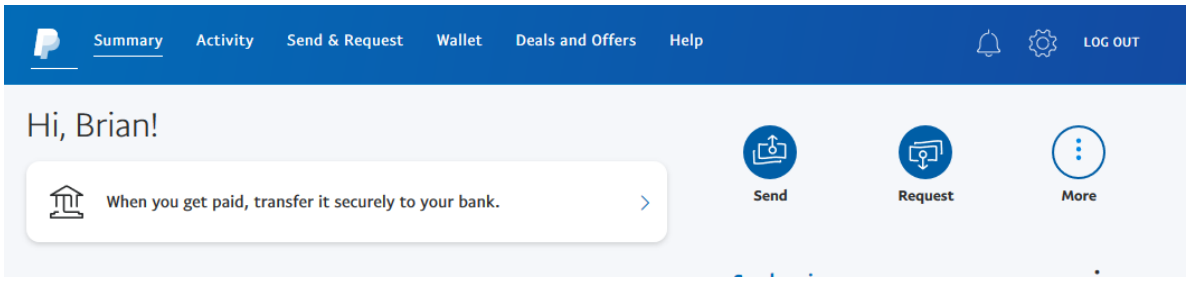
Please contact Touchstone_Support@aegis.net if you need help with registration.

3. Can I login with something other than the email I registered my account with? You can either login with your account's email address, or a LoginID that was created when you first registered.

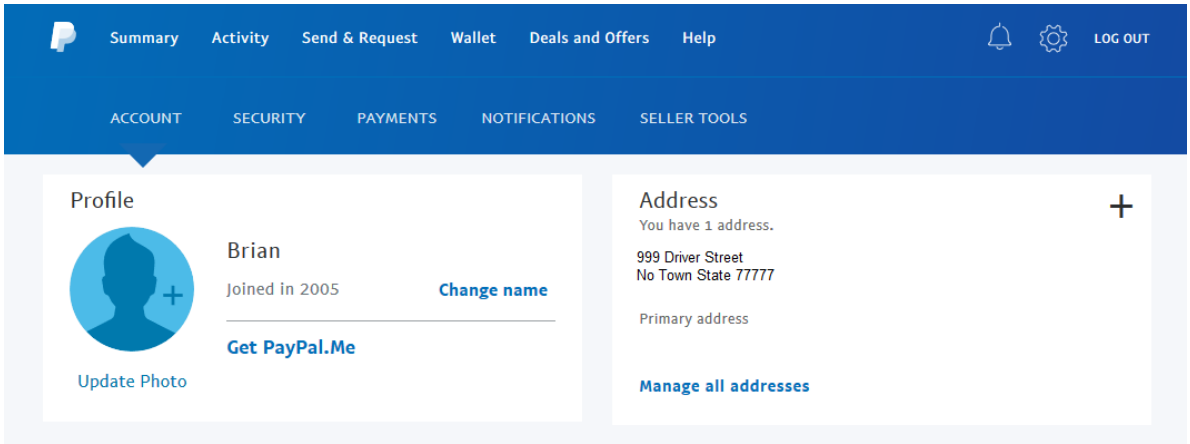
   You can find the LoginID that was created when you registered your account by going to the 'My Settings' page.

4. How do I change the name of my Org? Watch this quick walthrough of the Org rename process.

5. How do I cancel my Starter-level Touchstone subscription and stop the automatic PayPal charge? To unsubscribe from your Touchstone subscription when payment is made through PayPal, log in to your PayPal account by clicking on the button on the upper right corner of the www.paypal.com page.



Click Profile (the gear icon) on the top right corner of the page.



Click Pre-approved payments under "Payment settings". Select the merchant whose agreement you want to cancel, and click Cancel. Follow the instructions to cancel the agreement.

Touchstone will be notified of the PayPal cancellation and will move their org back to Open/Free 30 days after their last monthly payment is received.

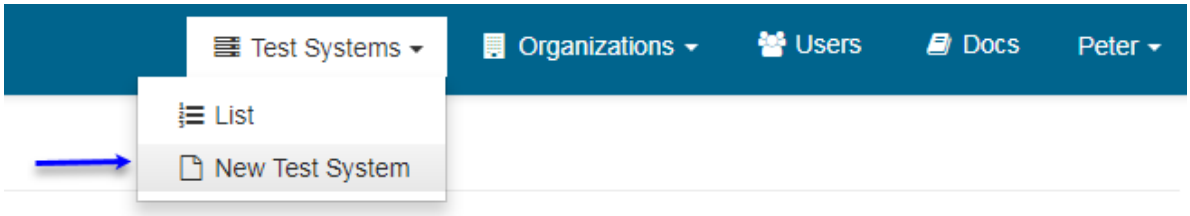
CHAPTER

THREE

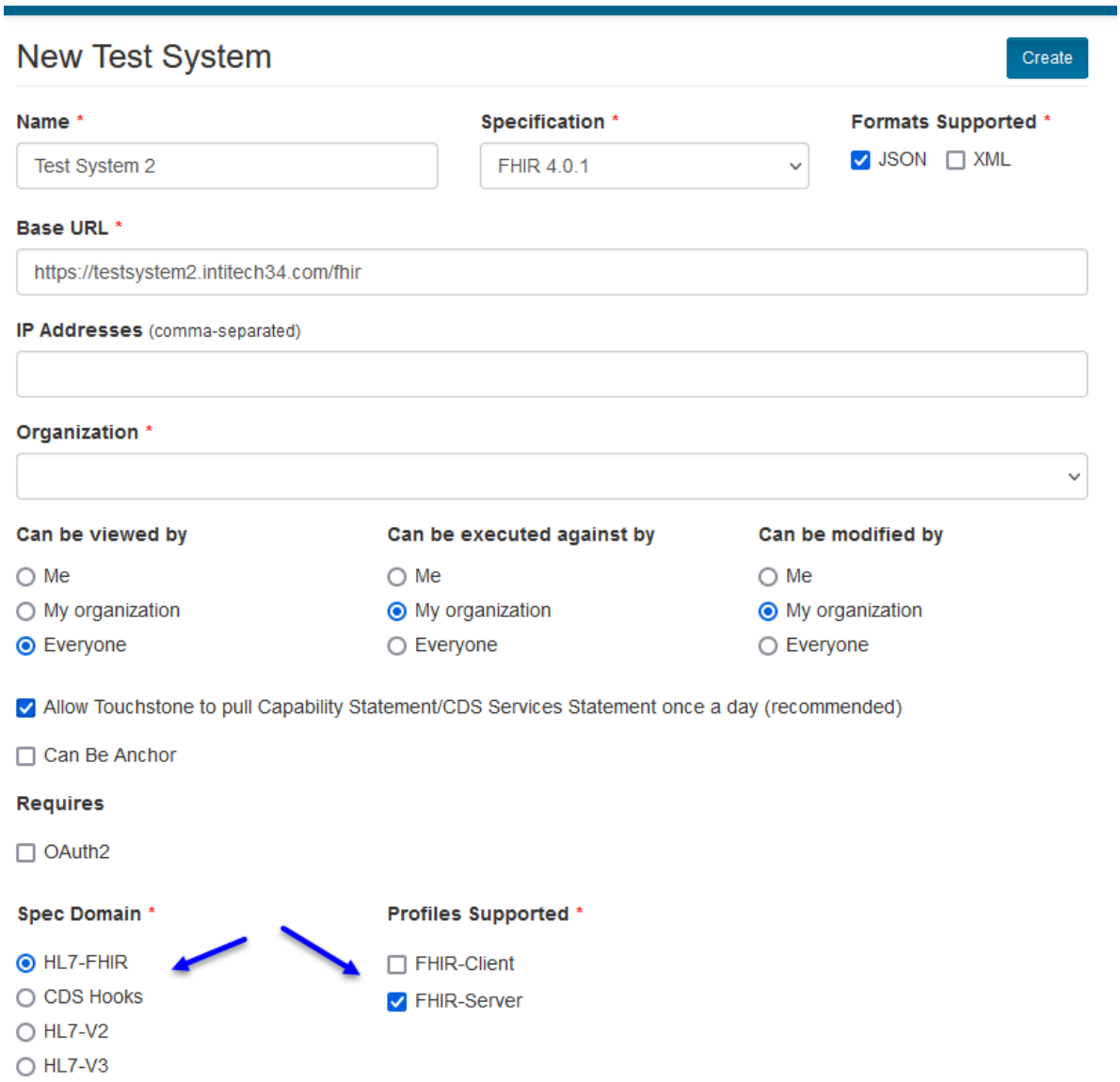
TEST SYSTEMS

3.1 Create Test System

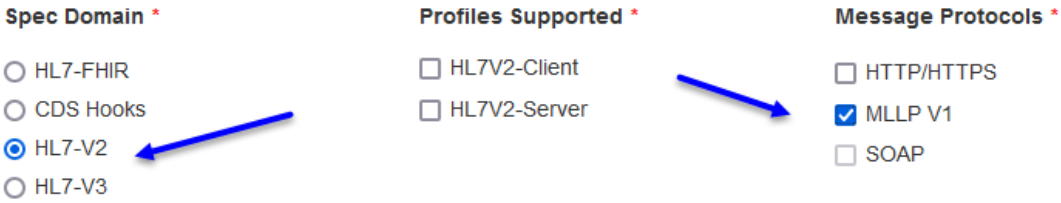
1. Sign in (as Tester or Org Rep) and click on the New Test System link in the top menu:



2. To select whether your Test System is an HL7-FHIR, CDS Hooks, HL7-V2, HL7-V3 test system, select the correct option under **Spec Domain**. To enable selection of your test system as the Destination (or 'target') of message exchanges in Test Setup, select the `Server` option for **Profiles Supported**:



3. If you select either `HL7-V2` or `HL7-V3` as your **Spec Domain**, then you will have a new option appear for **Message Protocols**:



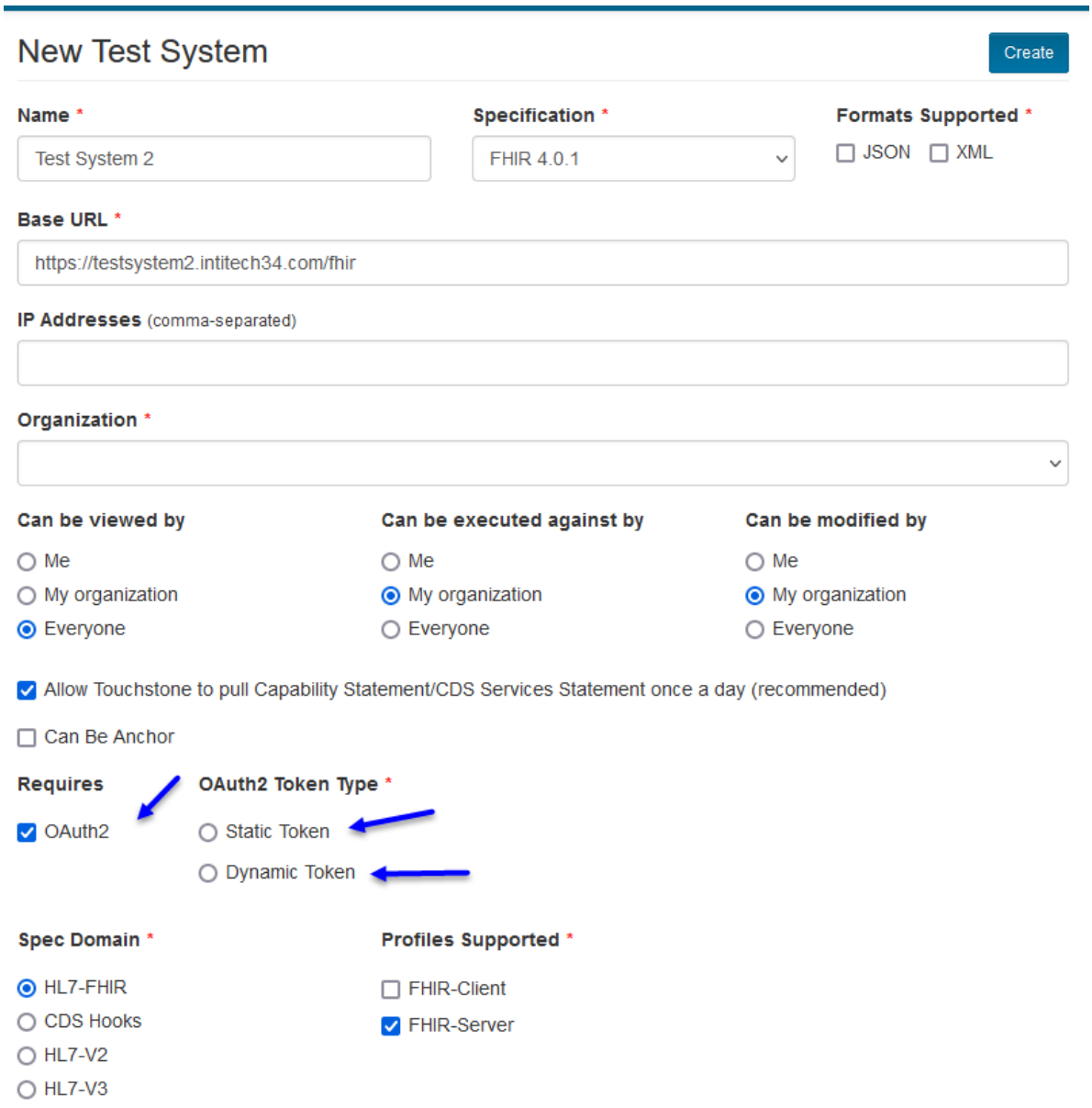
Note: You do **not** need to include your organization name in the test system name. Touchstone prefixes the test system name with your organization name where necessary.

- Name – This will be displayed along with your Organization name in Test System select-boxes in Touchstone.

- Base URL – Must be reachable on the public internet. Refer to Service Base URL for details. Base URL is limited to 512 characters. For a `HL7V2-Server`, the Base URL should be in the form of `hostname:port`.
- IP Addresses – This will be populated automatically by Touchstone. You can add additional IP addresses for the test system if the auto-detected one is incorrect. Note that the IP address is used primarily for Client test systems. As such it can be ignored if your system only responds to request and does not initiate message exchanges to other test systems.
- Can be viewed by – If `Me` or `My Organization` is selected, then test system will not be listed on the Test Systems screen for users outside your organization. If `Me` is selected, then even other users within your organization will be unable to see the test system.
- Can be executed against by – If `Me` or `My Organization` is selected, then users outside your organization cannot execute tests against the test system. If `Me` is selected, then even other users within your organization will be unable to execute tests against the test system.
- Can be modified by – If `Me` or `My Organization` is selected, then users outside your organization cannot modify this test system's attributes in Touchstone. If `Me` is selected, then even other users within your organization will be unable to modify attributes of this test system.
- Allow Touchstone to pull capability statement once a day – Touchstone conditionally evaluates assertions during test execution based on test system capabilities as defined by its Capability Statement. To ensure that Touchstone has the latest copy of your Capability Statement, allow Touchstone to download this statement from your server once a day (by checking this box) and ensure that your test system has the statement available.
- Can Be Anchor – Whether or not this test system can serve as an *Anchor System* in conformance suites. Conformance results are **NOT** collected on Anchor Systems. You can learn more about Anchor Systems *here*.
- Requires / OAuth2 – Leave this unchecked if your system is a client test system only or if it does not require OAuth2. Otherwise, choose whether this system uses a static token or a dynamic token. If Static Token is chosen, `Authorization` request header will be set to this value by Touchstone when your test system is the target of an interaction.
- Spec Domain – Choose which type of test system you are creating, one that utilizes `HL7-FHIR`, or one that utilizes `CDS Hooks`.
- Profiles Supported – If your test system only responds to requests and does not initiate message exchanges to other test systems, then select the `Server` option.
- Message Protocols – For `HL7-V2` and `HL7-V3` test systems, you have the option to designate what type of message protocols the test system is designed to use. For `HL7-V2`, `HTTP/HTTPS` and `MLLP V1` are available, and for `HL7-V3`, `HTTP/HTTPS` and `SOAP` are available. `HL7-FHIR` and `CDS Hooks` do not have these options becuase they are `HTTP/HTTPS` only.

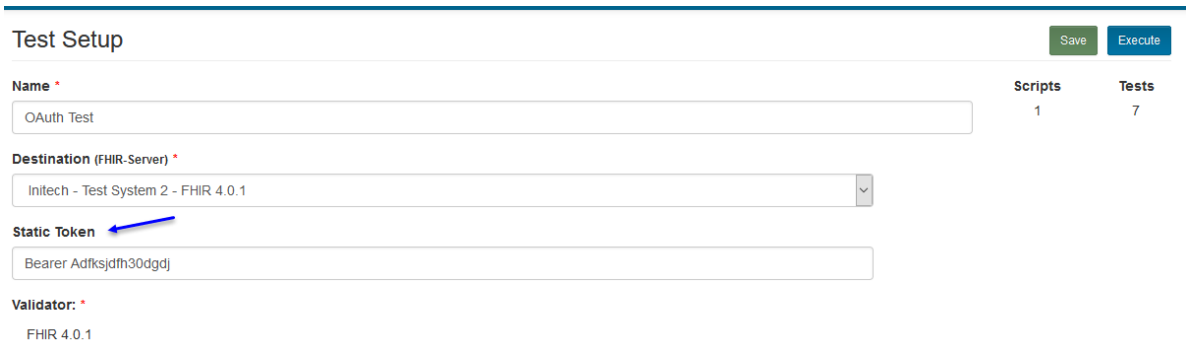
Note: While the options are there for `SOAP` and `HTTP/HTTPS` in the **Message Protocols**, at this time Touchstone does not support `HL7-V2 HTTP/HTTPS` and `HL7-V3 SOAP`, but will in the future.

4. To enable OAuth, check the OAuth2 checkbox and choose whether this system uses a static token or a dynamic token:

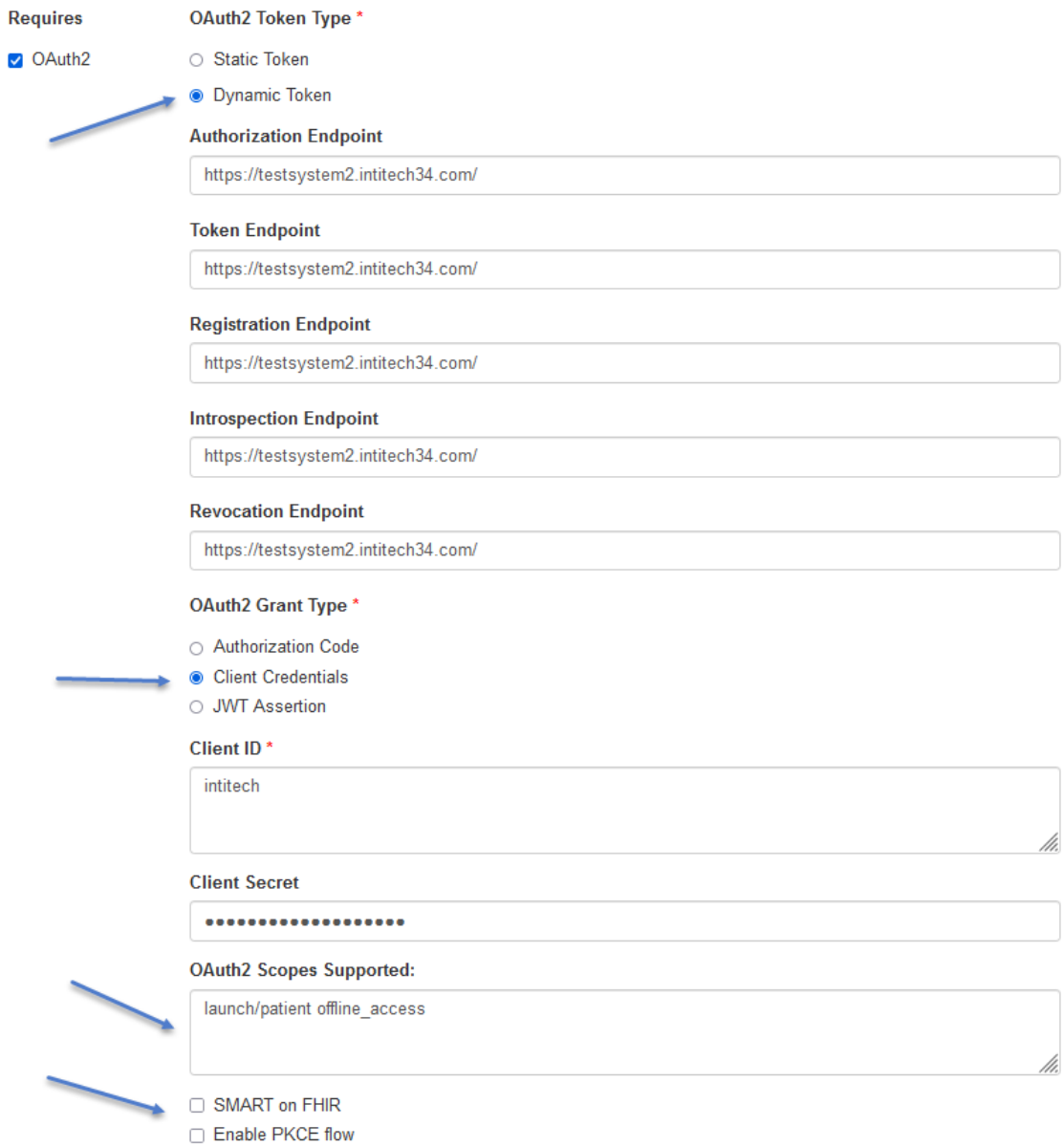


Note: Touchstone is currently not designed to use OAuth2 with Test Systems that support `HL7-V2` and `HL7-V3`.

5. If the Static Token button is checked, a default value for the static token is required, but the value can be overridden at Test Setup if needed.



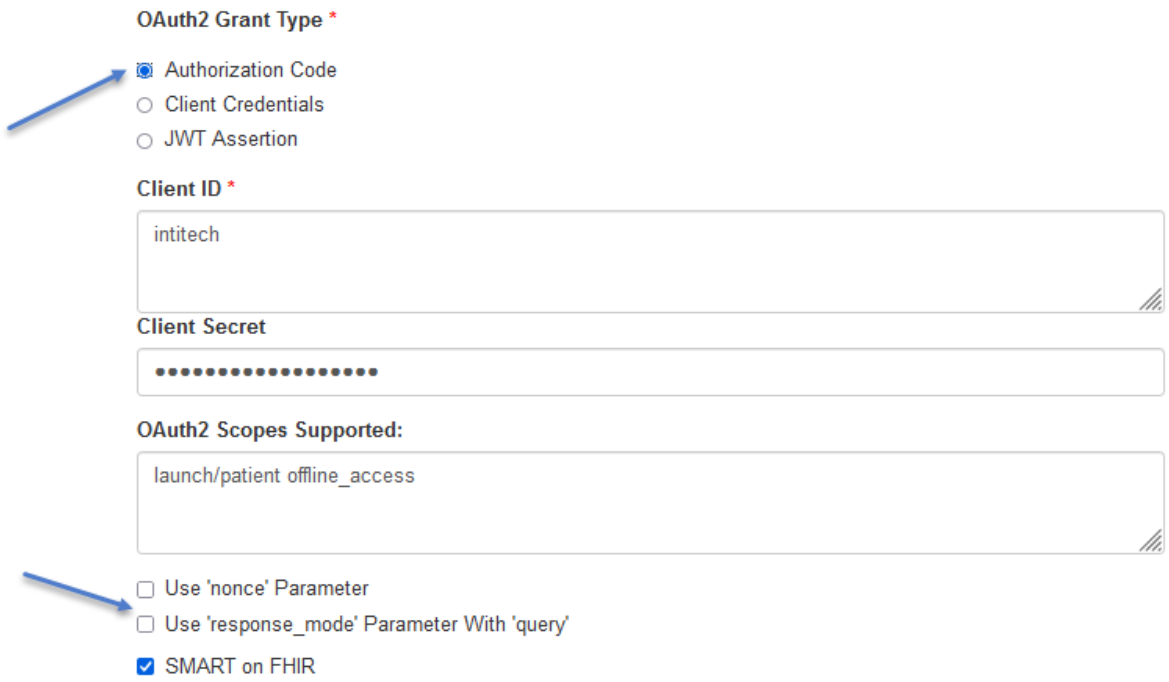
6. If the Dynamic Token button is checked, the following is shown to the user:



- Authorization Endpoint – The authorization endpoint URL of the FHIR Server.

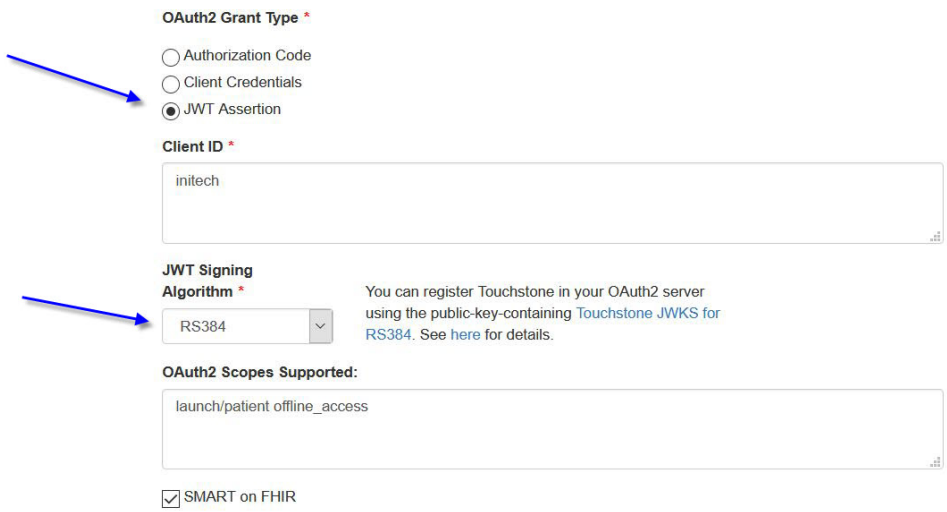
- Token Endpoint – The token endpoint URL of the FHIR Server.
- Registration Endpoint – The registration endpoint URL of the FHIR Server.
- Introspection Endpoint – The introspection endpoint URL of the FHIR Server.
- Revocation Endpoint – The revocation endpoint URL of the FHIR Server.
- OAuth2 Grant Type – The grant type used for the server, **Authorization Code**, **Client Credentials**, or **JWT Assertion**. Choosing one of these is required.
- Client ID / Client Secret – For **Authorization Code**, **Client Credentials**, and **JWT Assertion** grant type, **Client ID** is requried.
- OAuth2 Scopes Supported – Optional input of the OAuth2 scopes that are supported.
- SMART on FHIR – Whether or not the Test System supports SMART on FHIR. If this box is checked, then the URL endpoints will be overwritten if a FHIR CapabilityStatement is retrieved from the server.
- Enable PKCE flow – This checkbox allows Touchstone to recognize the need for code_challenge and code_verifier parameters and use them in automatic authorization requests per PKCE (Proof Key for Code Exchange) specification

7. If Authorization Code is selected, the following is shown to the user:



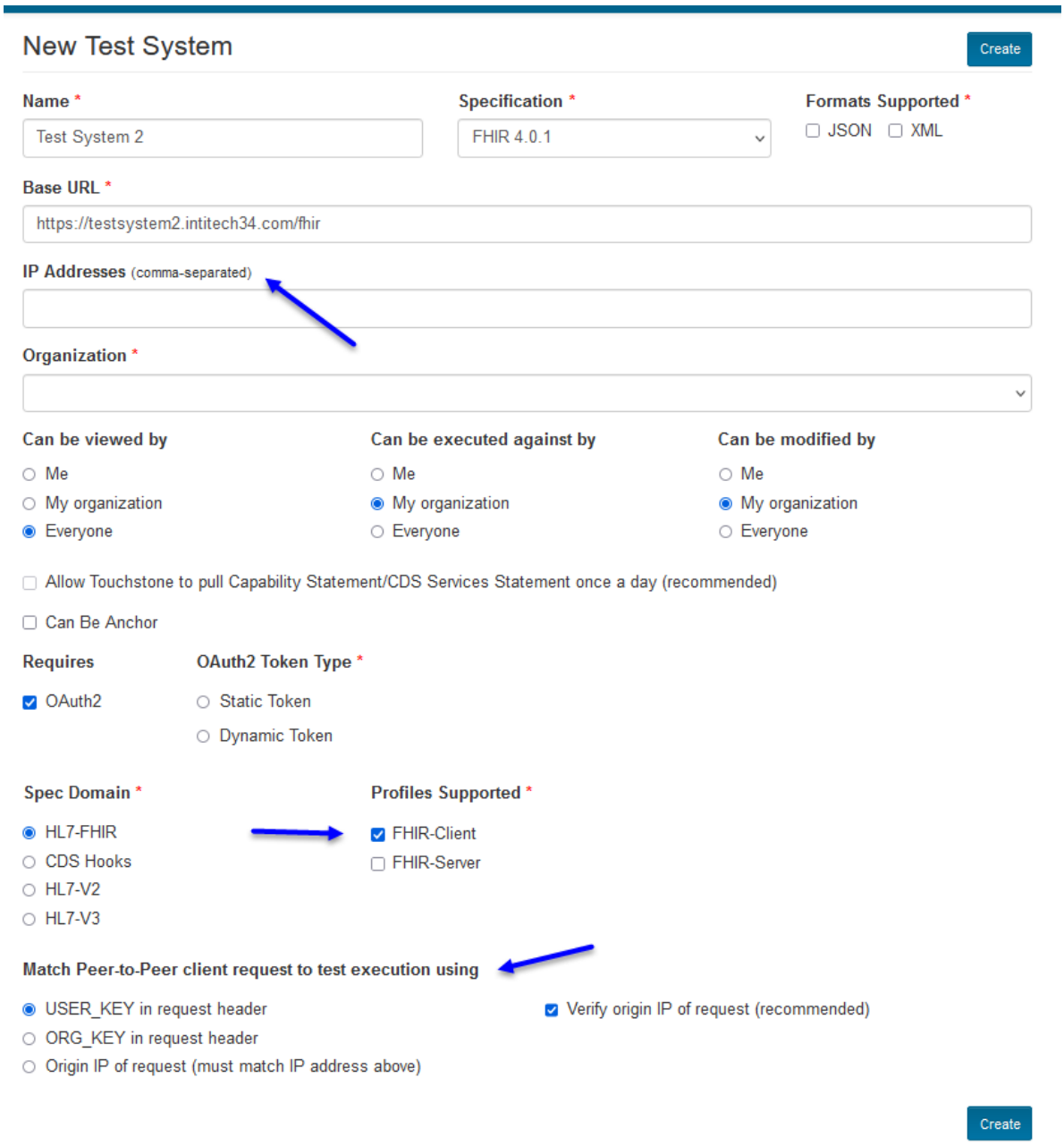
- 'nonce' Parameter – The 'nonce' parameter is sent in the Authorization Request with a randomly generated 12 character value.
- 'response_mode' Parameter – The 'response_mode' parameter is sent in the Authorization Request with a value of "query".

8. If JWT Assertion is selected, the following is shown to the user:



- JWT Signing Algorithm – Algorithm used to sign or encrypt the JSON Web Token. Required for **JWT Assertion**.
  - For detailed information on setting up your Test System with JWKS please refer to the SMART App Launch documentation here.
  - **Note:** The Touchstone public-key can be found using the link next to the **JWT Signing Algorithm**. Be sure to select the correct algorithm before navagating this link.

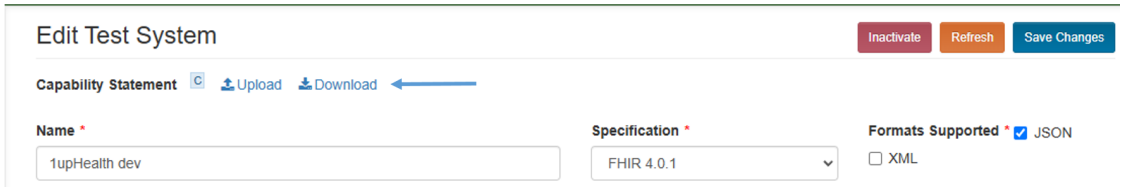
9. To enable selection of your test system as the Origin (or 'source') of message exchanges in Test Setup, select the `Client` option for **Profiles Supported**:



- Match Peer-to-Peer client request to test execution using – This is the mechanism by which Touchstone will match peer-to-peer request messages to test executions. Peer-to-peer message exchanges are covered under Peer-to-Peering testing.
- Verify origin IP of request – If checked, Touchstone will verify that the origin IP address of the request in peer-to-peer exchanges matches the client test system's IP address in Test Setup. Without this verification, other client test systems could pretend to be this test system.
- IP Addresses – This becomes critical if you have selected `Origin IP of request` for `Match Peer-to-Peer`. It's also critical if you have checked `Verify origin IP of request`.
- Requires / OAuth2 – Leave this unchecked if your system is a client test system only.
- Allow Touchstone to pull capability statement once a day – It is still recommended to have this checked even if the test system is a client system only. Capability statement is applicable to client test systems as well. See Rest

Mode.

10. On the Edit Test System page, you can download the Capability Statement that is on your test server if you have one, or you can manually upload one to Touchstone:



3.2 Capability Statement

Test Systems can declare what interactions, operations, and resources they support in the Capability Statement. These declarations are used:

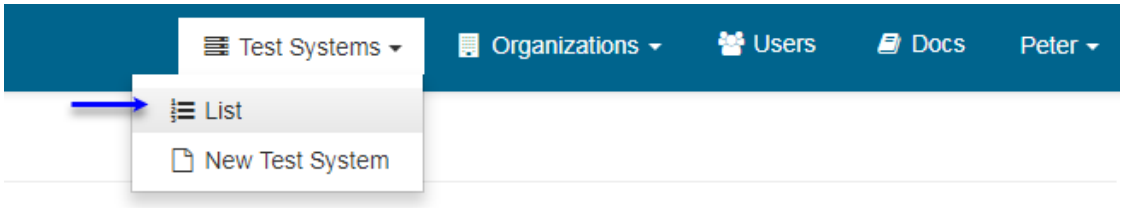
- to inform the end-user of the interactions that are unsupported by the test system during test execution.
- to conditionally evaluate test result assertions during test execution.
- to measure conformance of test systems against the specification. These measurements feed the Conformance Results.
- in Test Result Publishing.

Touchstone uses the Base URL of your test system to download the Capability Statement. If the Base URL is `https://testsystem1.initech34.com/fhir`, then Touchstone will make a GET request to: `https://testsystem1.initech34.com/fhir/metadata` using the appropriate Accept header. Refer to the capabilities section for more details. A Capability Statement can also be uploaded to Touchstone when editing a Test System.

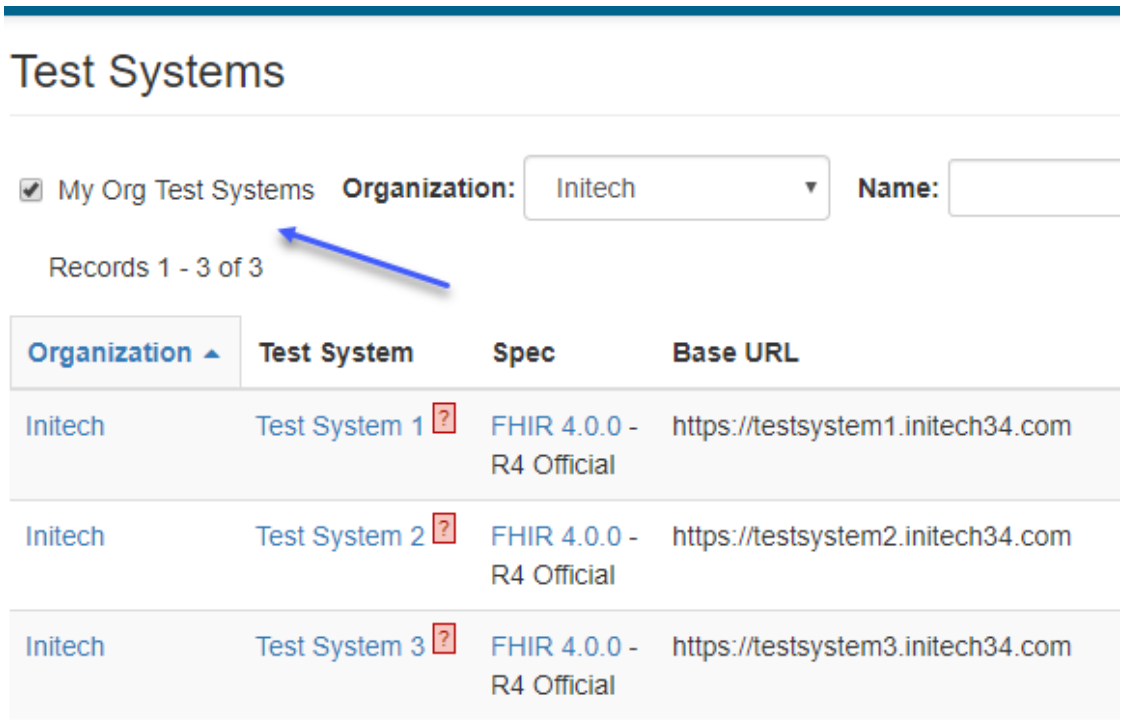
Note: Test Systems that are `HL7-V2` or `HL7-V3` do not use a Capability Statement, so those Test Systems will not have the option to download or upload the Capability Statement.

3.3 Test System List

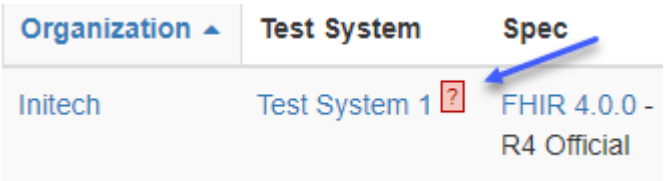
You can get a list of all test systems by clicking on the Test Systems List link in the top menu:



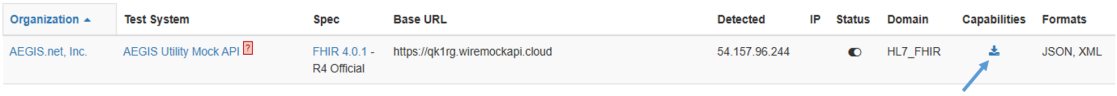
To view only the test systems that belong to your organization, you can check the `My Org Test Systems Only` option:



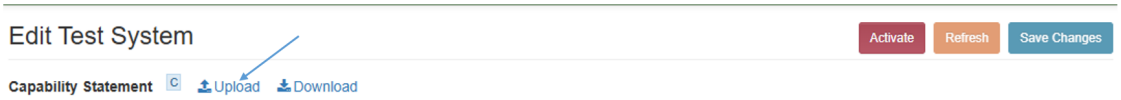
Red question marks appear for test systems that do not yet have a Capability Statement downloaded or uploaded into Touchstone:



You can get rid of that by downloading the Capability Statement here:



Or you can upload the Capability Statement on the Edit Test System screen here:



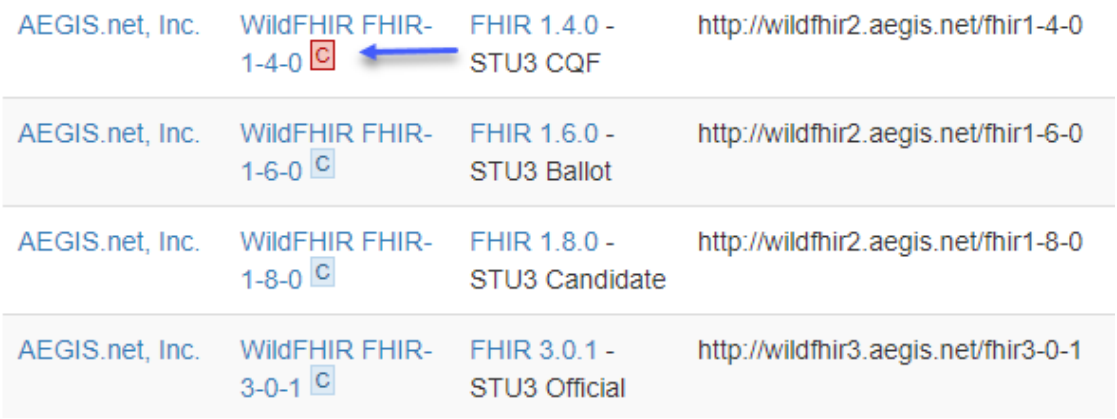
If your test system host is not publicly accessible, then you will see the following error:

If your test system's `/metadata` endpoint is not working, then you will see the following error:

> **Warning:** Getting past these errors is critical for future testing in Touchstone. You can use any REST client to ensure proper retrieval of your Capability Statement. Make sure you're able to do so even when you're not connected to your company's network.

For an example of how to define a FHIR capability statement, you can refer to http://build.fhir.org/capabilitystatement-example.json.html.

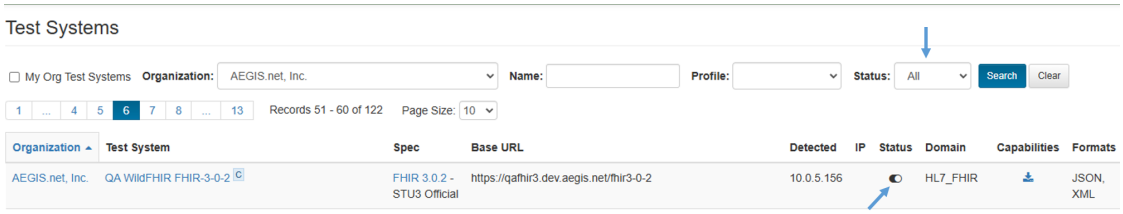
Red [C] marks appear for test systems whose Capability Statement is invalid. You can click on the [C] to view the list of validation errors.



> **Warning:** It is important to get past all validation errors reported by FHIR Validator. Validations are triggered automatically by Touchstone when a Capability Statement is downloaded or uploaded into Touchstone.

Activate or Deactivate Test Systems:

The Status column indicates if a Test System is Active (toggled to the right) or Inactive (toggle to the left). A Test System can be activated or inactivated by toggling the status button if the logged in user has permission to edit the Test System. If the user does not have permission the toggle will still display the status of the Test System but the user will not be able to toggle the status. Test Systems can be filtered by their status or all Test Systems can be viewed.



Test system status can also be changed on the Edit Test system page:

Note: Inactive Test Systems will remain available for selection when viewing Test Execution Results but will not be available for selection when executing tests.

3.4 FAQ

1. When registering Touchstone as a client for an Oauth Test System I need to register the redirect_uri, where can I find that? The redirect_uri for Touchstone is a static value: "`https://touchstone.aegis.net/touchstone/oauth2/authcode/redirect`"
2. Why were the endpoints of my Test System updated unexpectedly? When the Capability Statement is downloaded/pulled from the Test System configuration page, Touchstone will automatically pull the well-known/smart-configuration. If the values in your smart-config are incorrect you may edit them on the test system configuration and save.
3. What is the list of Touchstone IP addresses that would need to be added to a whitelist if my Test System is behind a firewall? The following IP ranges should be added to a whitelist to grant Touchstone access to a Test System that is behind a firewall:

   1) Site 1 (Primary) - 192.196.182.192/26 (which covers our entire IP range - 192.196.182.192-255)

   2) Site 2 (Secondary) - 4.59.159.64/26 (which covers our entire IP range - 4.59.159.64-127)
4. Why does the IP address field show an email address on a Create Test System or Edit Test System page when there is no IP address provided and why does Touchstone produce an error on save of the page? Browsers are inconsistent in their handling of a blank field. Some browsers identify the IP Address field as an email type of field and when blank will prepopulate it with a browser side stored email address. In the event that the browser fills the IP Address field with an email please delete it prior to saving to avoid the Touchstone error.

CHAPTER

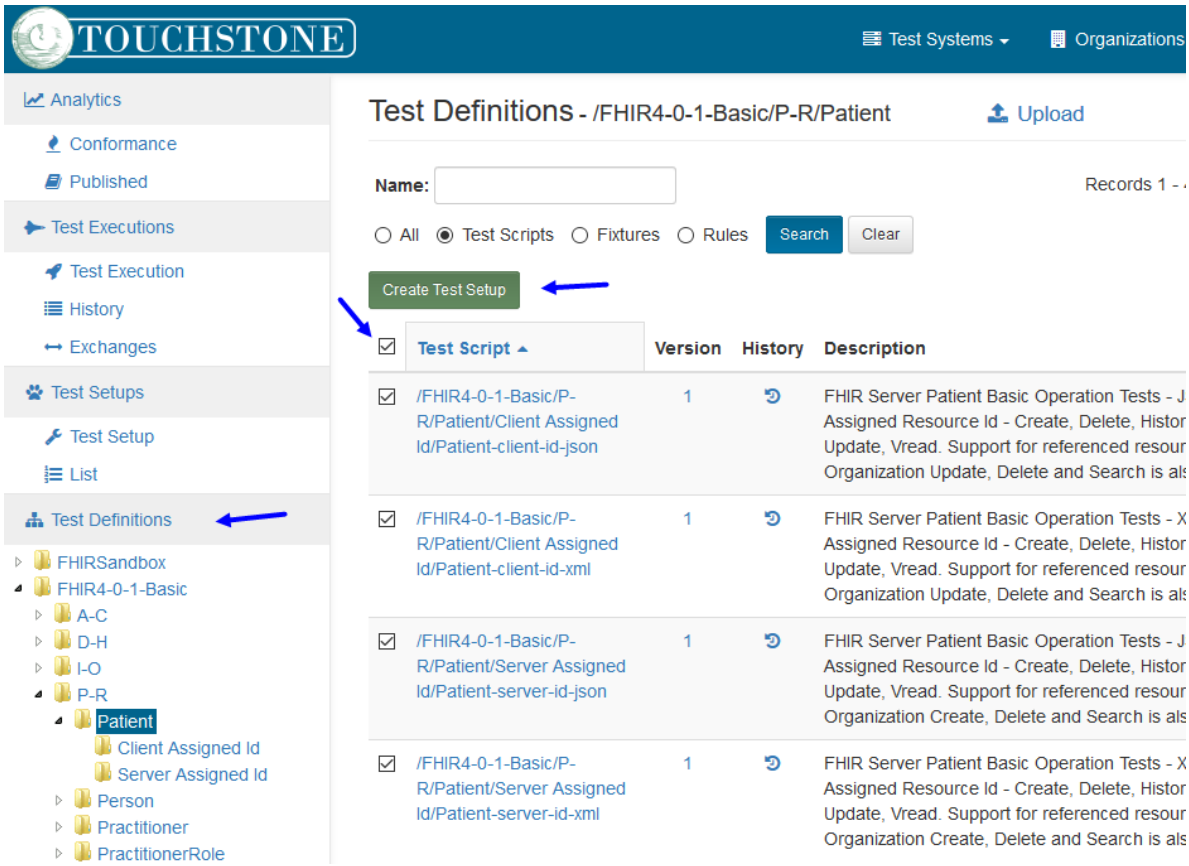
FOUR

EXECUTING TESTS

For both Touchstone-initiated testing and Client-side (Peer-to-Peer) testing, you need to create a Test Setup first. Both Org Rep and Tester users can create test systems, test setups, and execute tests.

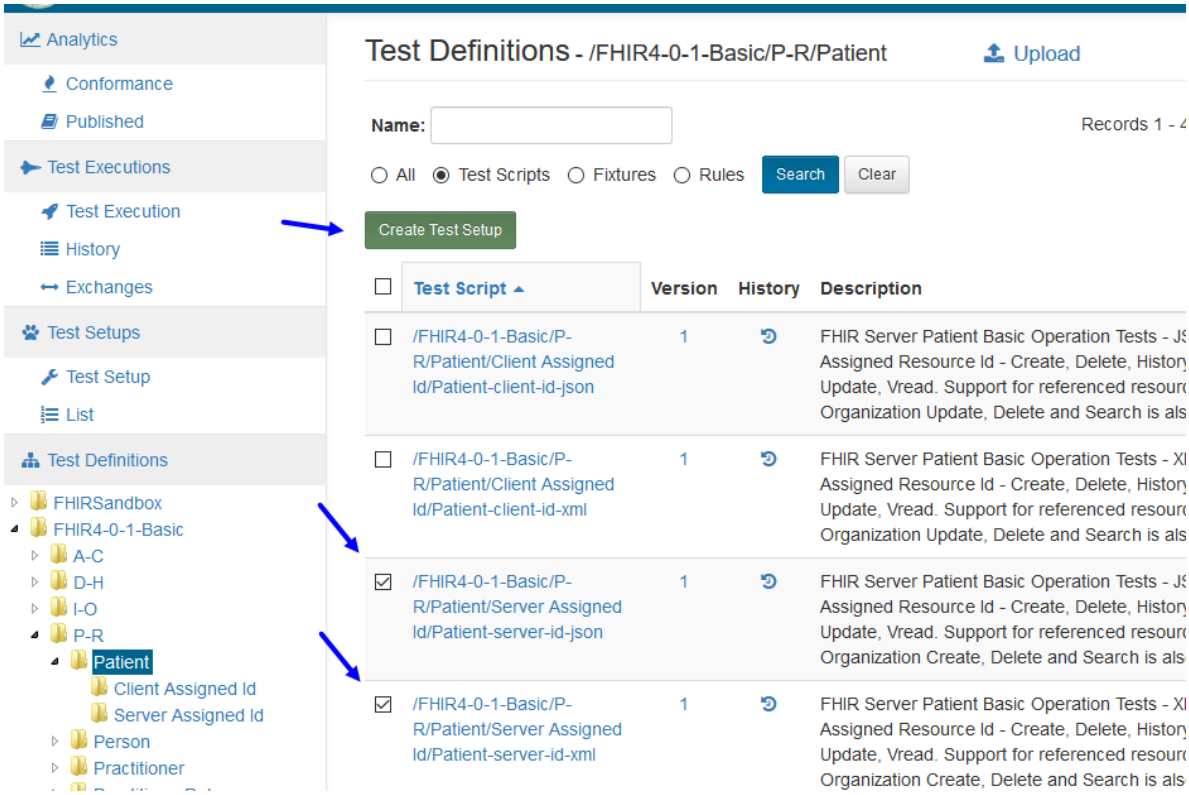
4.1 Creating Test Setup

1. Under Test Definitions in the left menu, select the set of test scripts you're interested in and click on the `Create Test Setup` button:



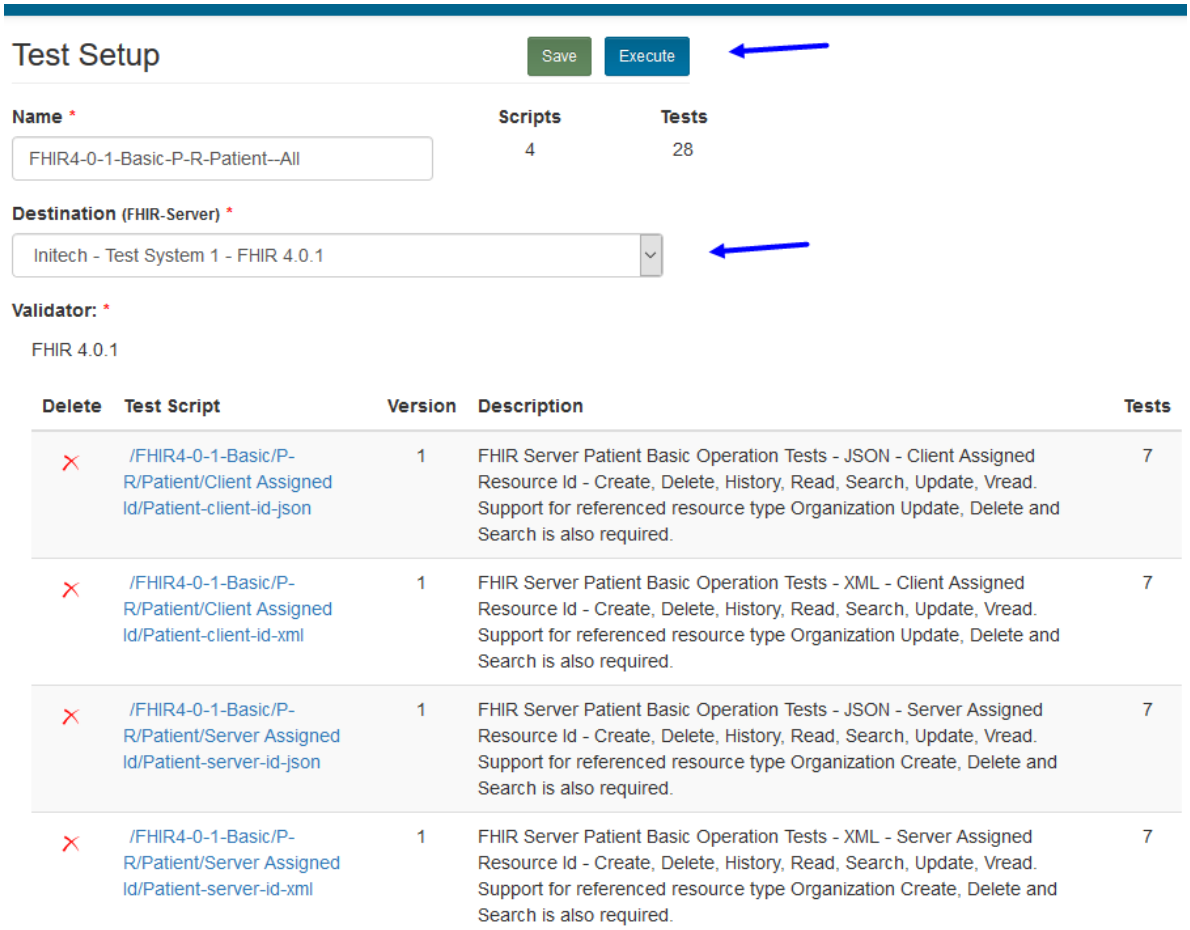
Note that the "All" checkbox (high-lighted in red above) was selected in this case. If you do this, Touchstone will remember this setting and will automatically pull test scripts that get added by the author to `/FHIR3-3-0-Basic/P-R/Patient` test group when you re-execute this Test Setup. Touchstone will also automatically remove any test scripts from this test setup that were removed from the test group by the author.

If you'd like to avoid Touchstone synchronizing the test setup with the test group, you can instead select individual test scripts separately when creating a particular test setup:



Note that in either case, the latest version of the test scripts will be pulled automatically by Touchstone when you re-execute a Test Setup (or a Test Execution). You do **not** need to go through Test Setup configuration again and again to get the latest test scripts (content).

2. After clicking on the `Create Test Setup` button, you'll land on the `Test Setup` screen where you can configure the Test System you'd like to execute tests against:



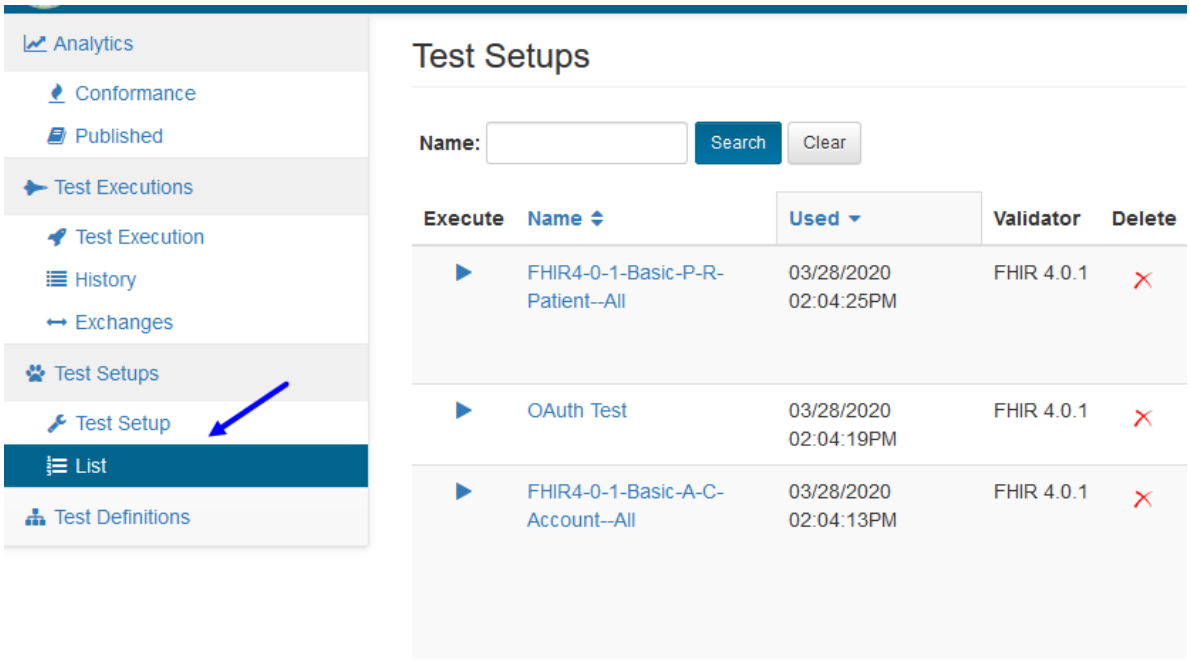
If you click on `Execute`, the system will both save your configuration settings and start the test execution.

If you click on `Save`, the system will only save your configuration settings.

You can override the system-generated name for your test setup in the `Name` input-box above.

If your TestScripts have Validator options set you can choose to use one of those instead of the default in the `Validator` drop-down above. For more information on how to Validators are set and maintained, refer to Validators.

You can access all your test setups by clicking on Test Setups / List on the left menu:



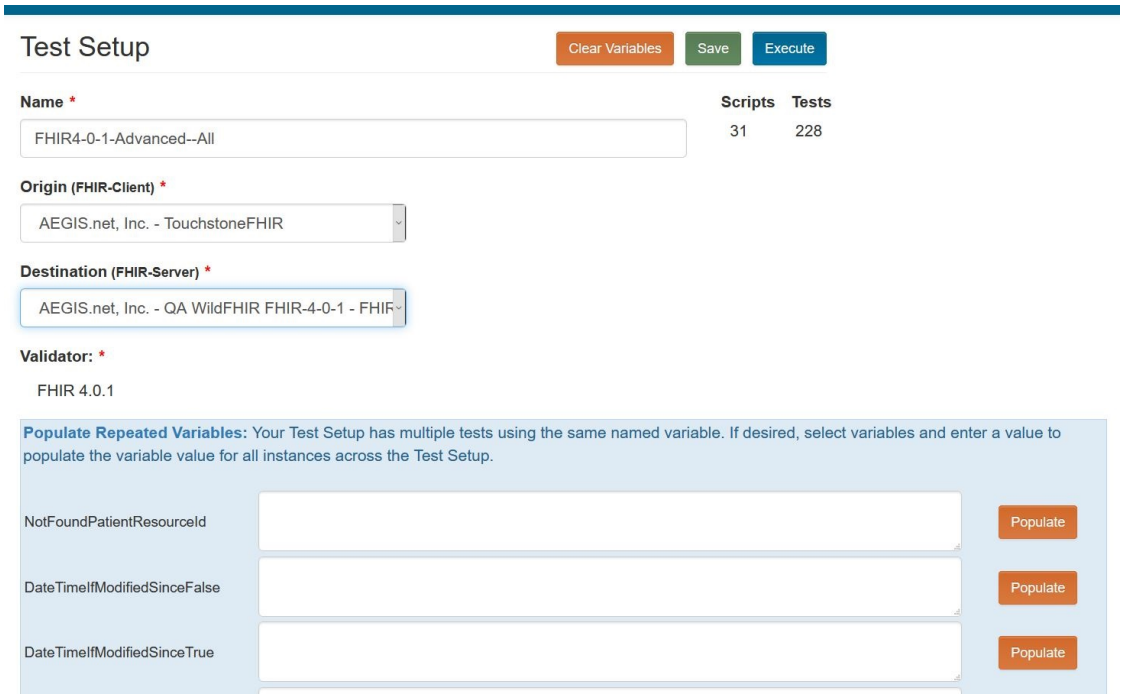
4.1.1 Repeated Variables

Often, when creating a Test Setup, there are several tests using identically named variables. Touchstone allows the user to input these variables once for all occurrences by populating the "Repeated Variable" field. Once the "Repeated Variable" field is filled, hit the "Populate" button. Doing so will place the desired variable value into each occurrence of that variable in the Test Setup. If you would like to clear all values for a particular variable, simply leave that "Repeat Variable" blank and hit "Populate".

Note:

- "Repeat Variables" are completely optional. All values can be entered manually if preferred.

You will also notice the "Clear Variables" button at the top of the Test Setup Page. This may be useful if you would like to manually enter values into a Test Setup without having to worry about out-of-date or invalid variables that were automatically populated. This button clears ALL variable values - not just the "Repeated Variable" values.



Fixtures in a TestScript define a set of resource instances that will be used as part of the setup, test, and teardown sections during TestScript execution. In Touchstone, those fixtures can be defined statically within the TestScript or they can be defined dynamically and the resource instance for the fixture is provided to the testscript as part of the test setup. More information about Fixtures can be found in the FHIR Testing Implementation Guide.

4.2 Dynamic Fixtures

Static fixtures are those that the testscript author uploads to Touchstone. The tester cannot change the content of such fixtures. In testscripts, they are defined using the **fixture** element:

```
<fixture id="resource-create">
    <autocreate value="false"/>
    <autodelete value="false"/>
    <resource>
        <reference value="../_reference/resources/Patient-create-server-id.json"/>
    </resource>
</fixture>
```

Dynamic Fixtures, on the other hand, are not uploaded by the testscript author. The tester has to specify the content of such fixtures before launching test executions. In testscripts, they are defined using a FHIR extension:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-dynamic-fixture">
    <extension url="id">
        <valueId value="resource-create"/>
    </extension>
    <extension url="resourceType">
        <valueString value="Patient"/>
    </extension>
    <extension url="contentType">
        <valueString value="application/fhir+json"/>
    </extension>
    <extension url="description">
        <valueString value="Patient create fixture. Enter the FHIR Patient resource that is to be created on the target test system."/>
    </extension>
    <extension url="hint">
        <valueString value="Patient create fixture"/>
    </extension>
</extension>
```

- id – This is similar to static fixture **id**. It identifies the fixture and can be referenced in testscript variables and operations.
- resourceType – Used to convey the FHIR resource type to the tester on Test Script Execution and Conformance screens. The resourceType specified in the fixtures contents is **not** validated against the extension resourceType when tester saves or executes the test setup. Testscript author can define a request assertion for resource to ensure tester has submitted the right resource type.
- contentType – Used to validate the contents of the fixture for proper JSON/XML format when tester saves or executes the test setup. Valid values are JSON and XML. Basic JSON/XML validation is performed. If the content is invalid JSON/XML, then tester is prevented from saving the test setup.
- description – Used to inform tester about the purpose of the fixture.
- hint – Used to inform tester about the purpose of the fixture.

You can learn more about this extension by visiting AEGIS Touchstone Testing Implementation Guide.

The dynamic fixture can be referenced in testscript variables and operations just like static fixtures:

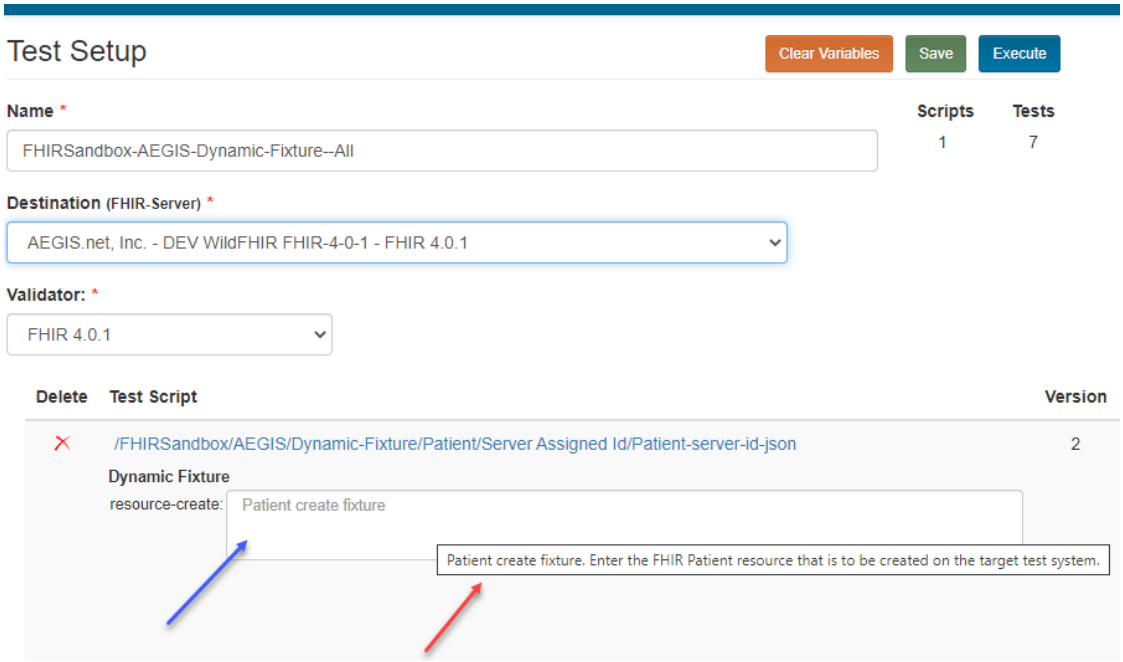
```
<variable>
    <name value="searchParamIdentifier"/>
    <path value=".identifier[0].value" />
    <sourceId value="resource-create" />
</variable>
```

```
<test id="Step1-CreateNewPatient">
    <name value="Step1-CreateNewPatient"/>
    <description value="Create a new Patient in JSON format where the server assigns the resource id. The expected respon

    <action>
        <operation>
            <type>
                <system value="http://terminology.hl7.org/CodeSystem/testscript-operation-codes"/>
                <code value="create"/>
            </type>
            <description value="Patient create operation with HTTP Header Accept and Content-Type set to JSON format."/>
            <accept value="json"/>
            <contentType value="json"/>
            <encodeRequestUrl value="true"/>
            <sourceId value="resource-create"/>
        </operation>
    </action>
```

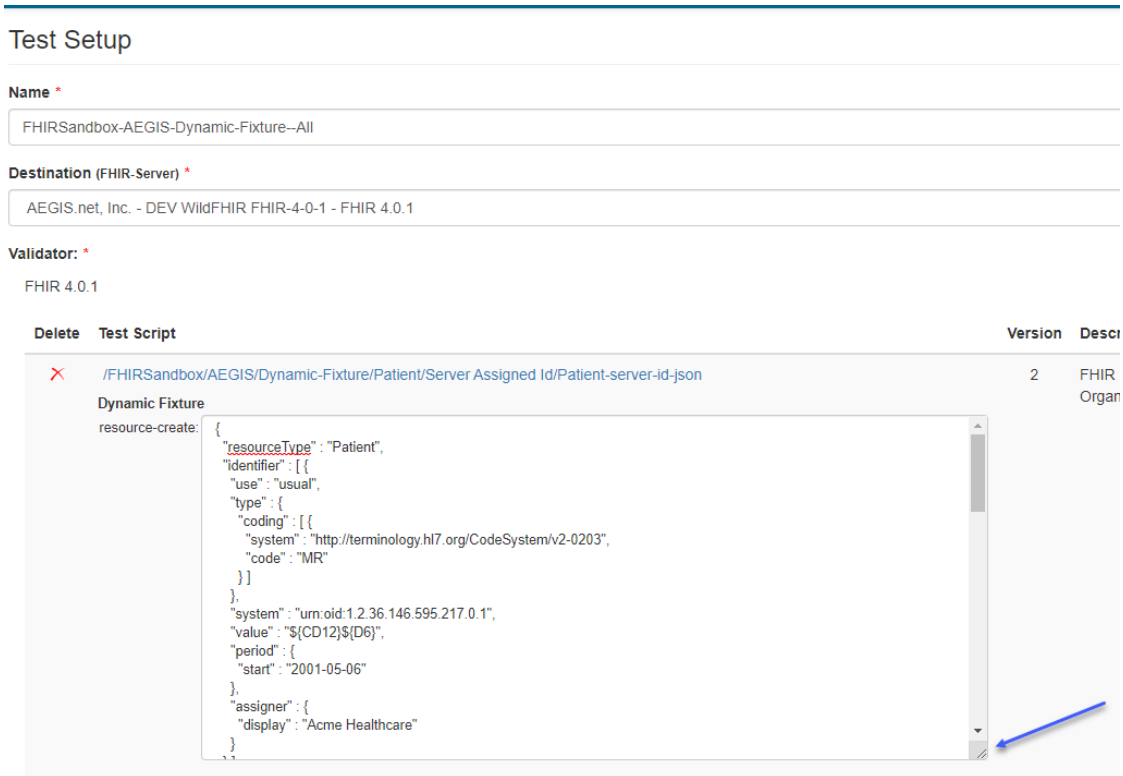
The next sections cover how the tester can specify the content of dynamic fixtures.

4.2.1 Test Setup

When a test setup is created that involves a dynamic fixture, the tester is prompted to provide the fixture contents. The fixture's **hint** (defined in the testscript) is displayed within the input box (blue arrow below). The fixture's **description** is displayed when user hovers over the input box (red arrow below).



Tester can enter the contents of the fixture and expand the input box using the drag handle (blue arrow below):



The dynamic fixtures used by a testscript execution are listed below the static fixtures on TestScript Execution screen:

Static Fixtures

Id	Resource	Version	Latest	Type	Contents
resource-update	Patient	1	1	JSON	Raw Resolved
reference-organization-create	Organization	1	1	JSON	Raw Resolved

Dynamic Fixtures

Id	Resource	Type	Contents
resource-create	Patient	JSON	Raw Resolved

They can be used in variables and operations just like static fixtures:

Variables

Name	Definition
createResourceId	**sourceId:** create-search-response **path:** .entry[0].resource.id
createResourceId	**sourceId:** create-search-response **path:** .entry[0].resource.id
createVersionId	**sourceId:** create-search-response **path:** .entry[0].resource.meta.versionId
searchParamIdentifier	**sourceId:** resource-create **path:** .identifier[0].value

Tests

Test Name	Description
Test: Step1-CreateNewPatient	Create a new Patient in JSON format where the server assigns the resource id. The expected response code is 201 (Create

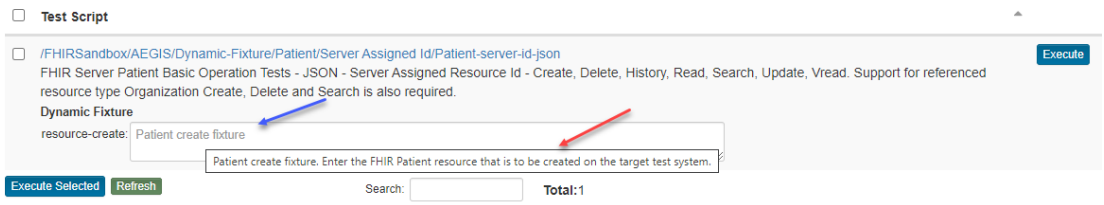
Action	Description	Status	Duration	Details
Operation	**create** - Patient **Origin:** TouchstoneFHIR **Destination:** AEGIS.net, Inc. - DEV WildFHIR FHIR-4-0-1 http://devfhir4.aegis.net:8080/fhir4-0-1	201 Created	0.129s	**Description:** Patient create operation with HTTP Header Accept and Content-Type set to JSON format. **Type:** create **Resource:** Patient **URL:** http://devfhir4.aegis.net:8080/fhir4-0-1/Patient **Definition:** { "type" : "create", "accept" : "json", "contentType" : "json", "description" : "Patient create operation with HTTP Header Accept and Content-Type set to JSON format.", "encodeRequestUrl" : true, "sysGen" : false, "sourceId" : "resource-create" } **Request:** **Method:** POST **Path:** http://devfhir4.aegis.net:8080/fhir4-0-1/Patient **Headers:** Accept application/fhir+json;charset=UTF-8 Content-Type application/fhir+json;charset=UTF-8 **Request:** Body **Response:** **Status:** HTTP/1.1 201 Created

Note: Testscript authors should define a request assertion for the fixture's resource-type after testscript operations that utilize a dynamic fixture. Touchstone does not validate the resource-type of a dynamic fixture when tester saves the test setup.

```
<action>
   <assert>
      <description value="Confirm that the submitted resource type is Patient.
↪"/>
      <direction value="request"/>
      <resource value="Patient"/>
      <warningOnly value="false"/>
   </assert>
</action>
```

4.2.2 Conformance Testing

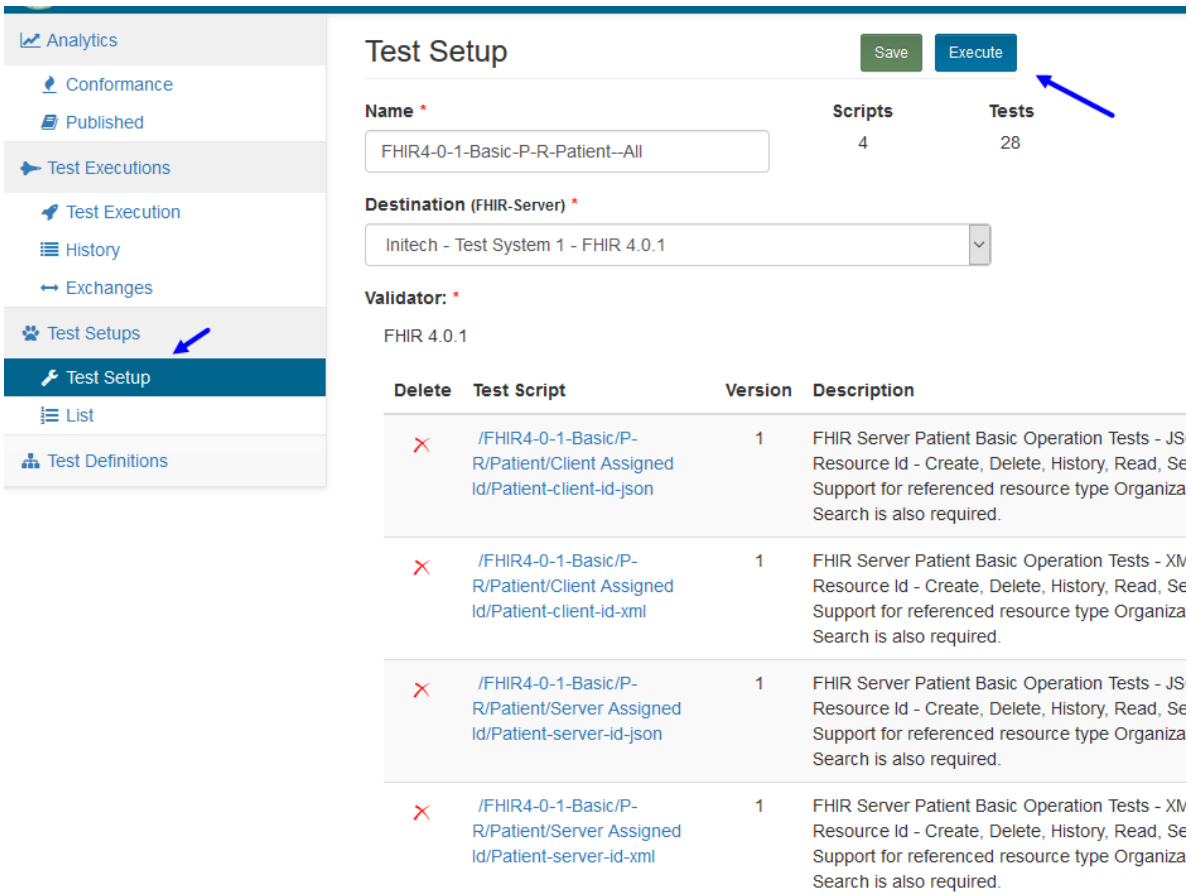
On the Conformance Current page, the tester is prompted for input the same way as on the Test Setup screen:

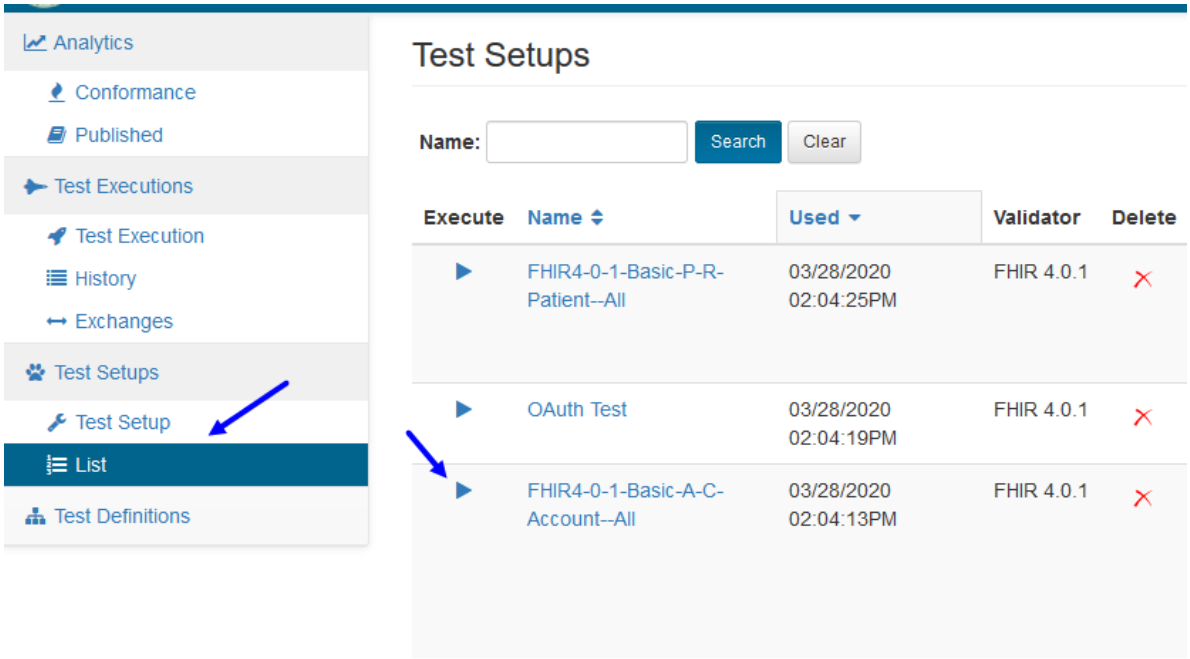


4.3 Test Executions

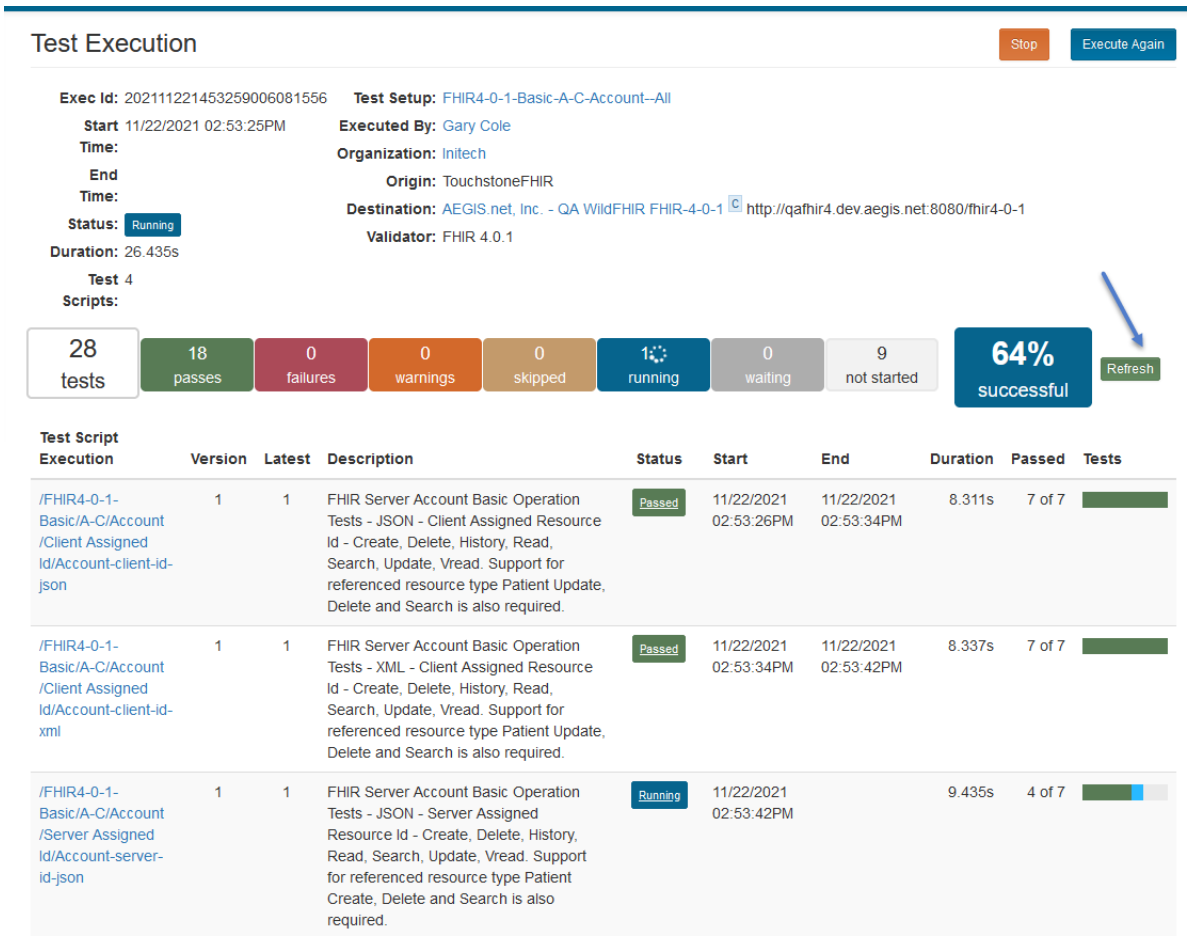
To launch a test execution, you can take the following steps:

1. Click the `Execute` button on the Test Setup screen or the Test Setups / List screen:





2. After clicking on `Execute` you'll be taken to the Test Execution screen where you can monitor the status of your overall test execution. The page will automatically refresh every few seconds, but you can also manually refresh the page by clicking the `Refresh` button:

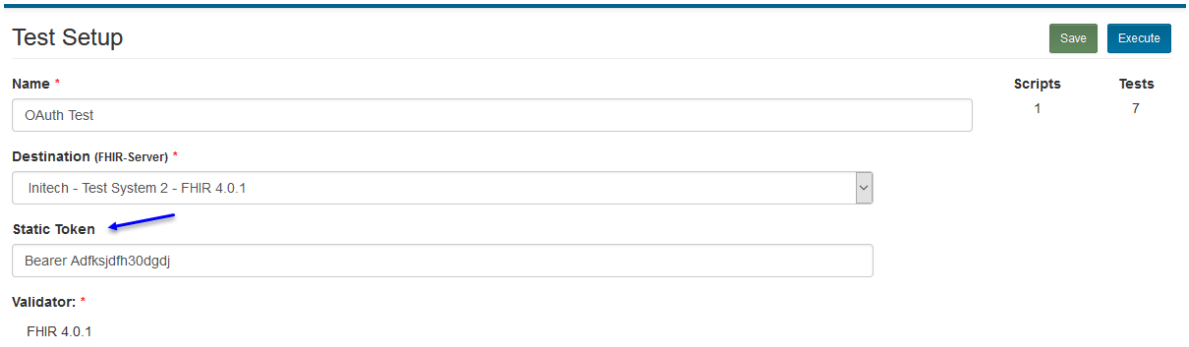


4.4 OAuth2 Test Executions

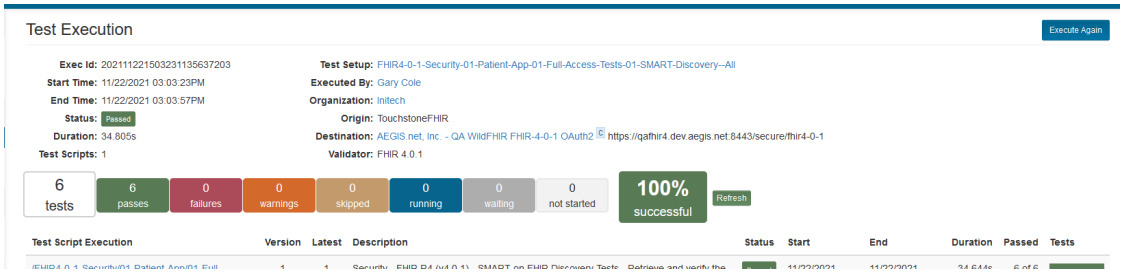
When running a Touchstone test execution against a FHIR server that is OAuth2 enabled, there are a few key changes that you will run into, depending on if the server is using a Static Token or a Dynamic Token.

4.4.1 Static Token

1. When a Test System is OAuth2 enabled and is set to use a Static Token, the Test Setup page will include the Static Token field when the Test System is selected as the Destination server. This field can be overwritten by the user if necessary:



2. The test execution will then begin, using the Static Token in the OAuth2 handshake with the FHIR server:

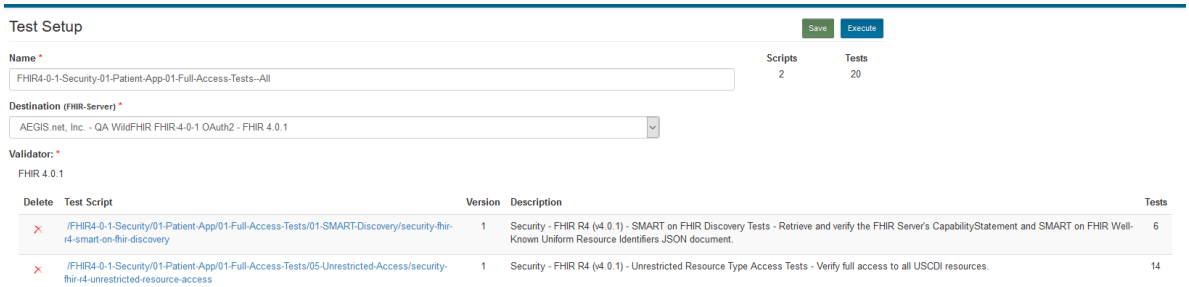


4.4.2 Dynamic Token

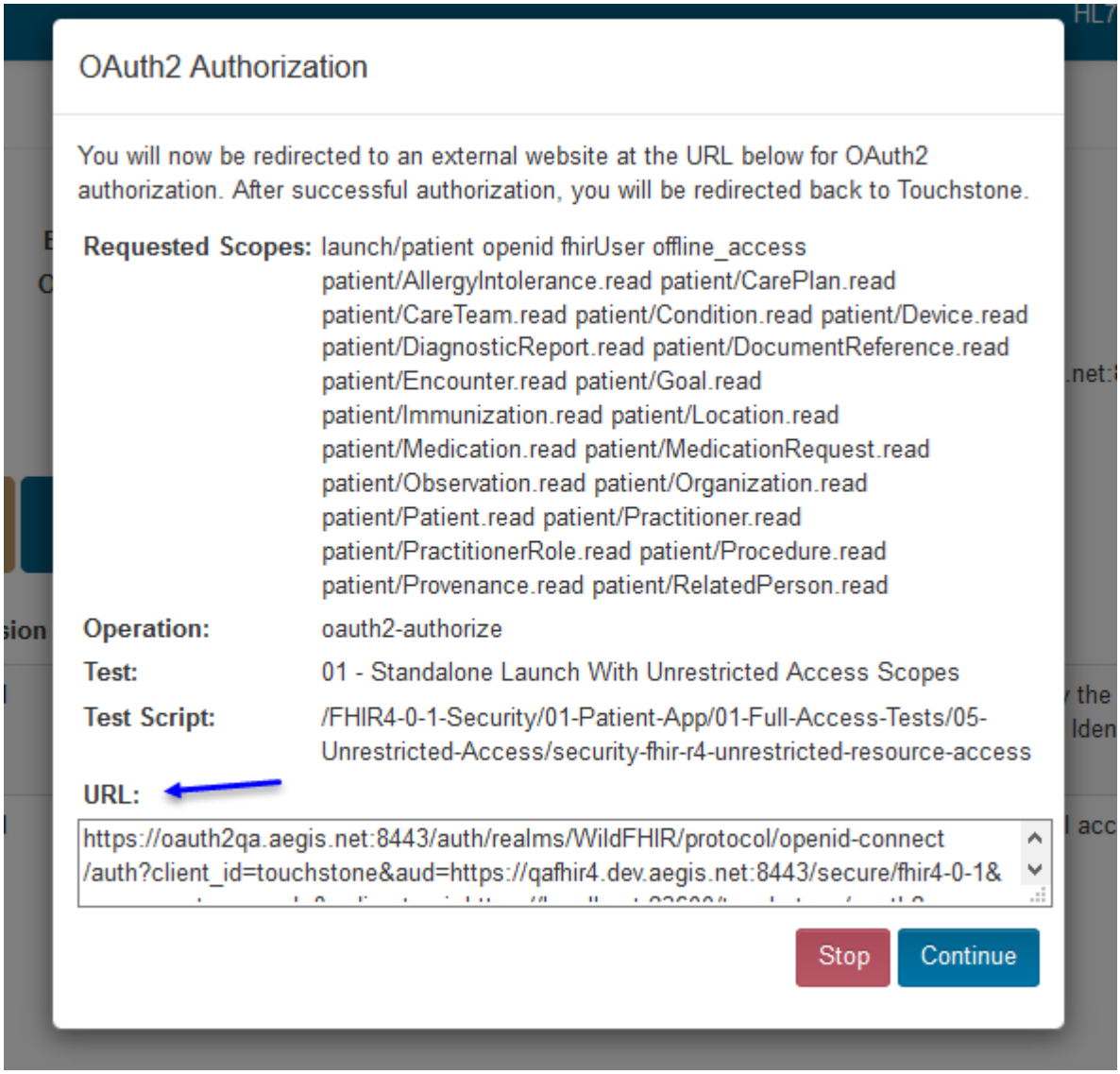
There are three different Grant Types that a Test System can use if it is using a Dynamic Token. One is Authorization Code, one is Client Credentials, and the other is JWT Assertion. They each have slightly different flow for the user when executing tests.

4.4.2.1 Authorization Code

1. When a Test System is OAuth2 enabled and is set to use a Dynamic Token with the Authorization Code grant type, the Test Setup page will look like the traditional Test Setup page:

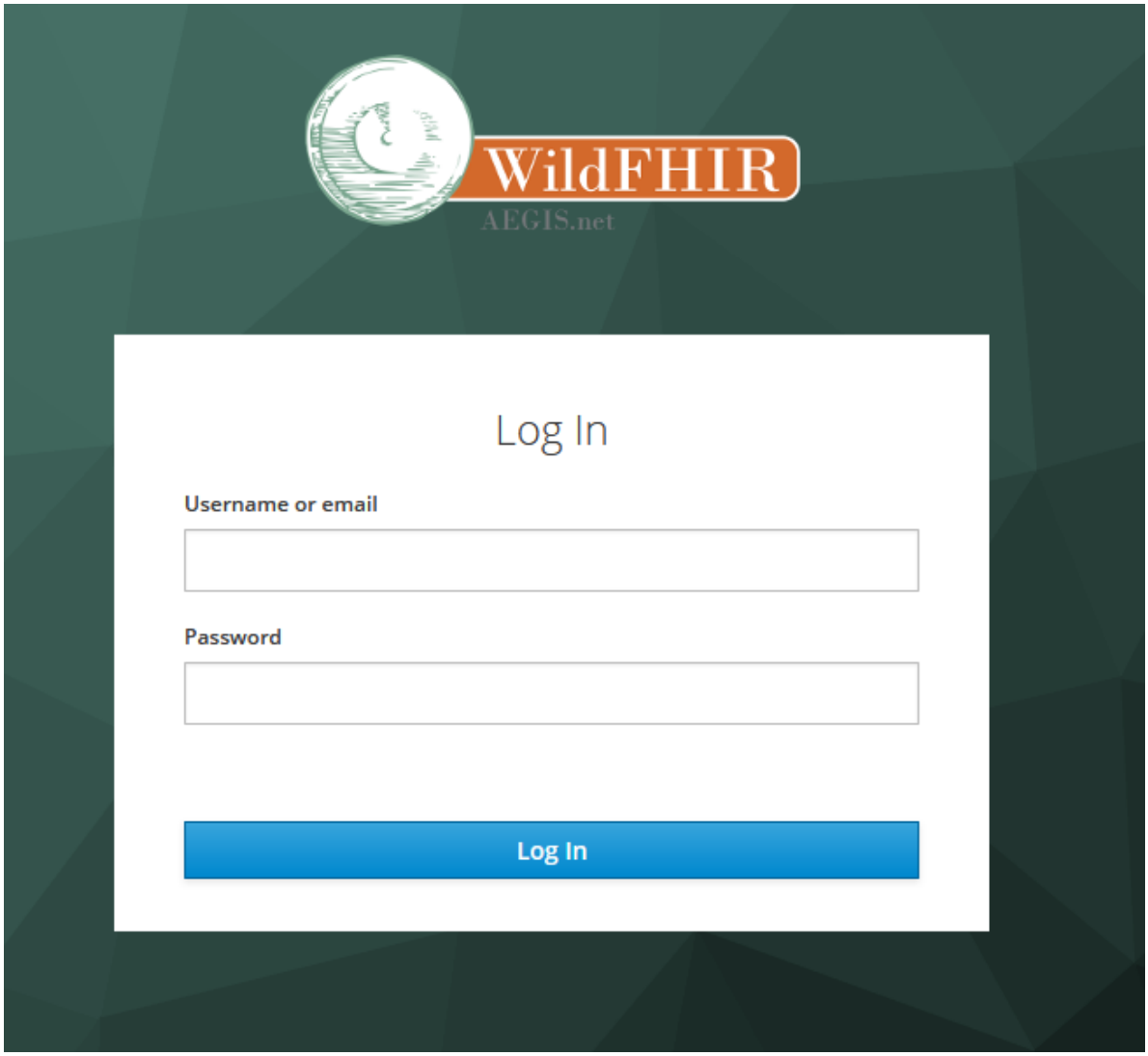


2. The test execution will begin, only this time, you will be prompted with a new window that will explain that you will be redirected to an external URL that will complete the OAuth2 Authorization step in the Test Execution process:



- Requested Scopes – Scopes requested by the user to the Authorization server.
- Operation – The operation being performed.
- Test – The particular test related to OAuth2 Authorization.
- Test Script – The overall Test Script related to OAuth2 Authorization.
- URL – External URL that the user will be taken to for login.

3. When you are redirected, you will be asked to use a login for the Authorization service that is in front of the FHIR server. The example below is the one for our WildFHIR OAuth2 enabled FHIR servers:



4. Once you are redirected back to Touchstone after the successful login to the Authorization service, in the background, Touchstone will be taking the Authorization Code recieved by the Authorization service and sending it back to trade it in for a Token that it will be able to use to access the FHIR server. If there is another test in your Test Setup that requires a login to and Authorization service, then you will be prompted again with the OAuth2 Authorization window, and repeat the process. Once the execution completes, the Test execution screen will present you with the results of the tests:

Test Execution

Execute Again

Exec Id: 20211122150912064514129 8
Start Time: 11/22/2021 03:09:59PM
End Time: 11/22/2021 03:10:18PM
Status: Passed
Duration: 18.866s
Test Scripts: 2

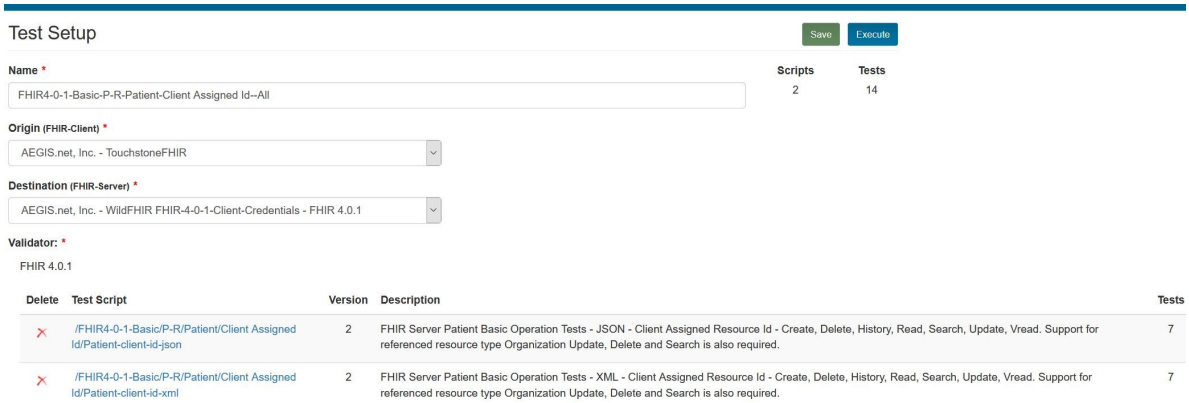
Test Setup: FHIR4-0-1-Security-01-Patient-App-01-Full-Access-Tests--tests
Executed By: Gary Cole
Organization: Initech
Origin: TouchstoneFHIR
Destination: AEGIS.net, Inc. - QA WildFHIR FHIR-4-0-1 OAuth2 https://qafhir4.dev.aegis.net:8443/secure/fhir4-0-1
Validator: FHIR 4.0.1

20 tests	20 passes	0 failures	0 warnings	0 skipped	0 running	0 waiting	0 not started	100% successful	Refresh

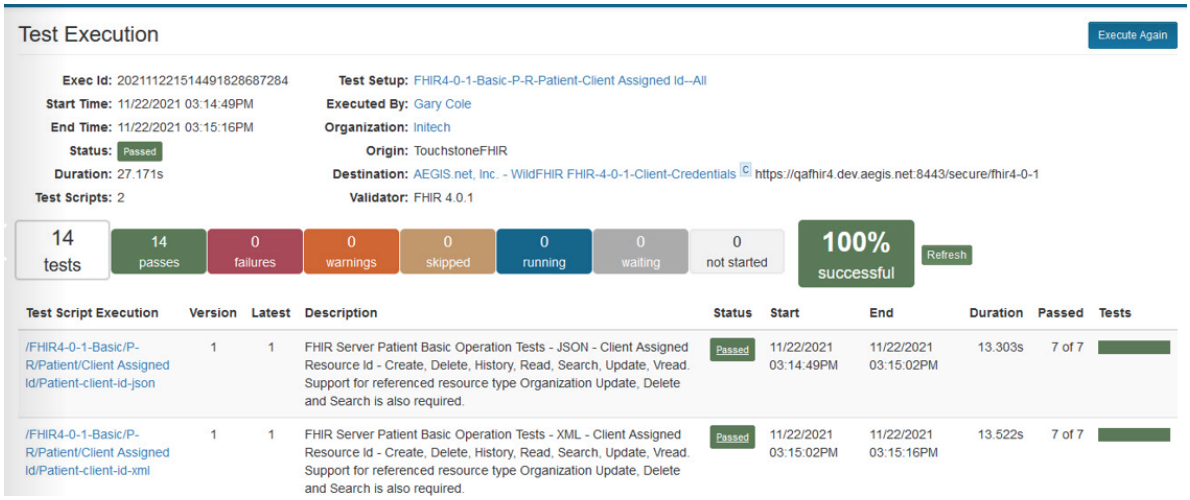
Test Script Execution	Version	Latest	Description	Status	Start	End	Duration	Passed	Tests
/FHIR4-0-1-Security/01-Patient-App/01-Full-Access-Tests/01-SMART-Discovery/security-fhir-r4-smart-on-fhir-discovery	1	1	Security - FHIR R4 (v4.0.1) - SMART on FHIR Discovery Tests - Retrieve and verify the FHIR Server's CapabilityStatement and SMART on FHIR Well-Known Uniform Resource Identifiers JSON document.	Passed	11/22/2021 03:09:12PM	11/22/2021 03:09:48PM	36.494s	6 of 6	
/FHIR4-0-1-Security/01-Patient-App/01-Full-Access-Tests/03-Unrestricted-Access/security-fhir-r4-unrestricted-resource-access	1	1	Security - FHIR R4 (v4.0.1) - Unrestricted Resource Type Access Tests - Verify full access to all USCDI resources.	Passed	11/22/2021 03:09:59PM	11/22/2021 03:10:18PM	18.863s	14 of 14	

4.4.2.2 Client Credentials

1. When a Test System is OAuth2 enabled and is set to use a Dynamic Token with the Client Credentials grant type, the Test Setup page will look like the traditional Test Setup page:

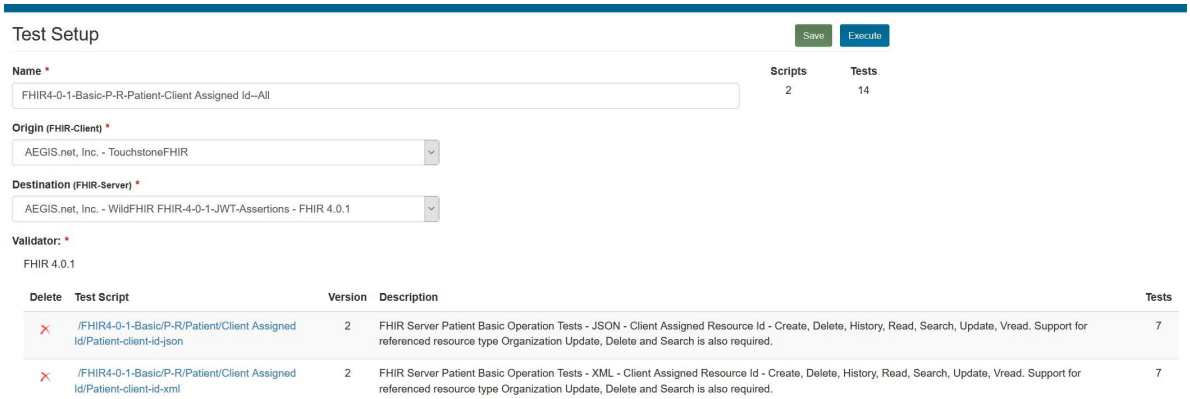


2. After starting the execution of the Test Setup, Touchstone will seem to run the Test Execution like it traditionally does. However, in the background, Touchstone is making a direct call to the OAuth2 server and retrieving the Token by using the Client ID and Client Secret that is saved to the Test System. The Token that is retrieved is then used to access the FHIR server right away. There is no extra step of going to login to an Authorization service because the Client ID and Secret are already trusted with the OAuth2 server, allowing a direct grant of the token. Once the execution completes, the Test execution screen will present you with the results of the tests:

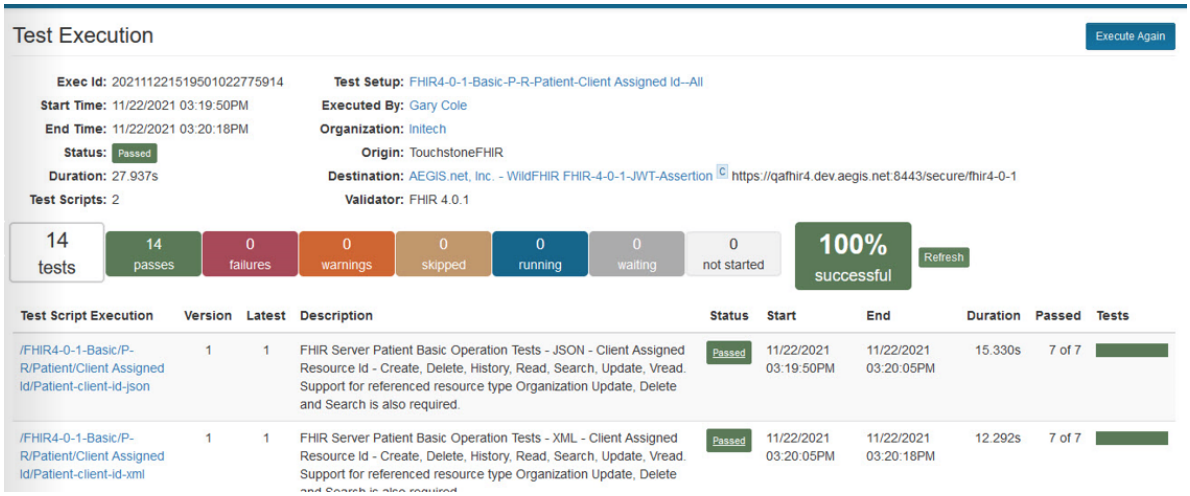


4.4.2.3 JWT Assertion

1. When a Test System is OAuth2 enabled and is set to use a Dynamic Token with the JWT Assertion grant type, the Test Setup page will look like the traditional Test Setup page:



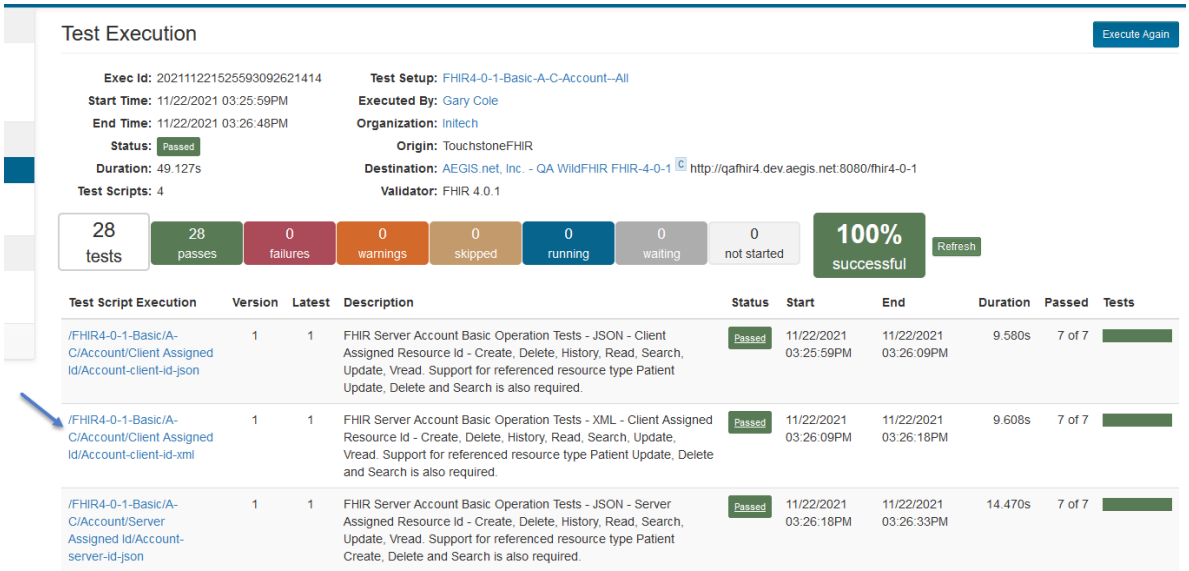
2. After starting the execution of the Test Setup, Touchstone will seem to run the Test Execution like it traditionally does. However, in the background, Touchstone is making a direct call to the OAuth2 server and retrieving the Token by using the Client ID and JWT Signing Algorithm chosen in the Test System Setup. The Token that is retrieved is then used to access the FHIR server:



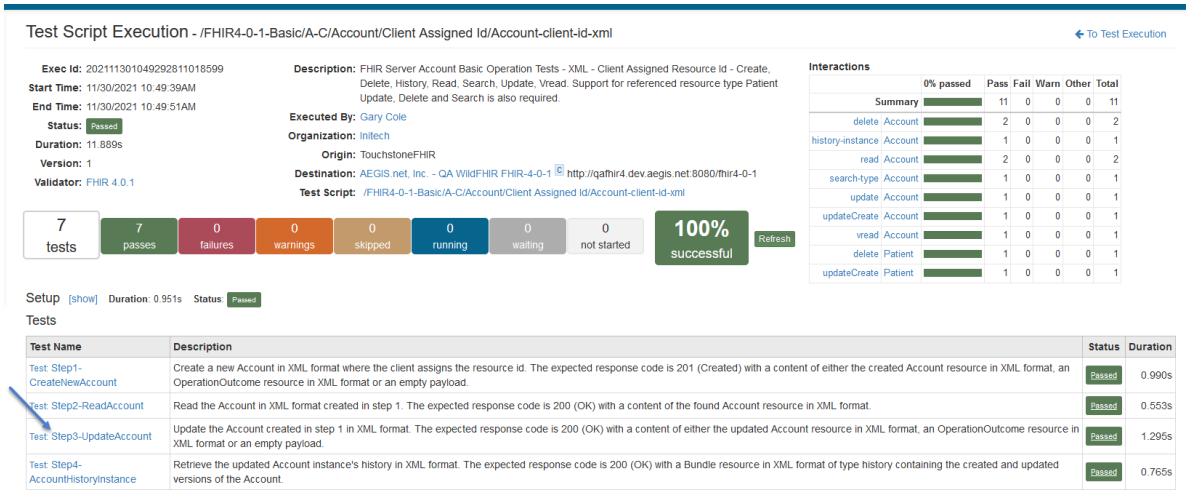
4.5 Test Execution Results

To get a detailed report on your test execution, you can take the following steps:

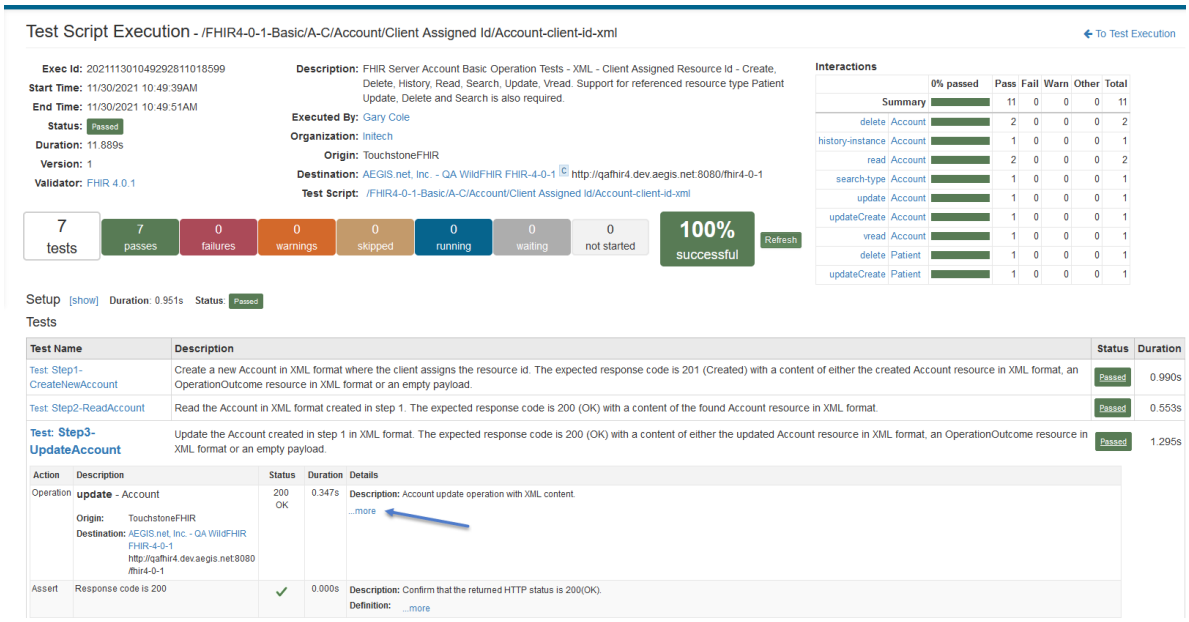
1. Click on the `Test Script Execution` link of interest on the Test Execution screen:



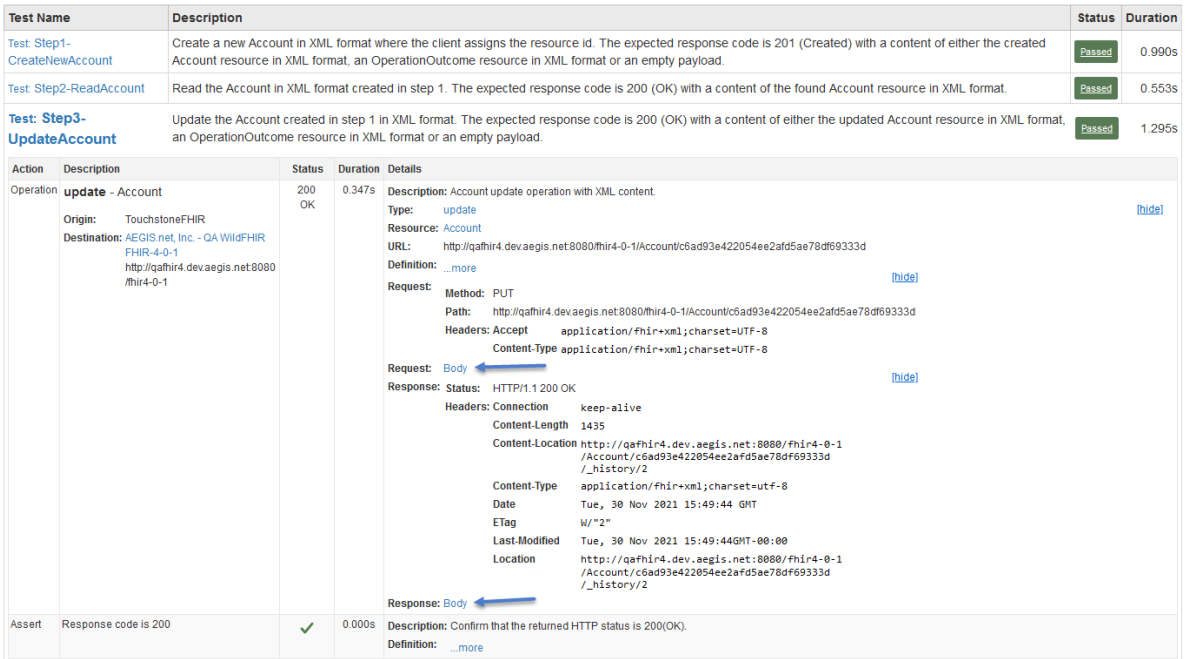
2. You can click on a Test of interest within the `Tests` section to get an expanded view of the test results:



3. You can click on the `...more` link to get detailed information on what took place during an operation:

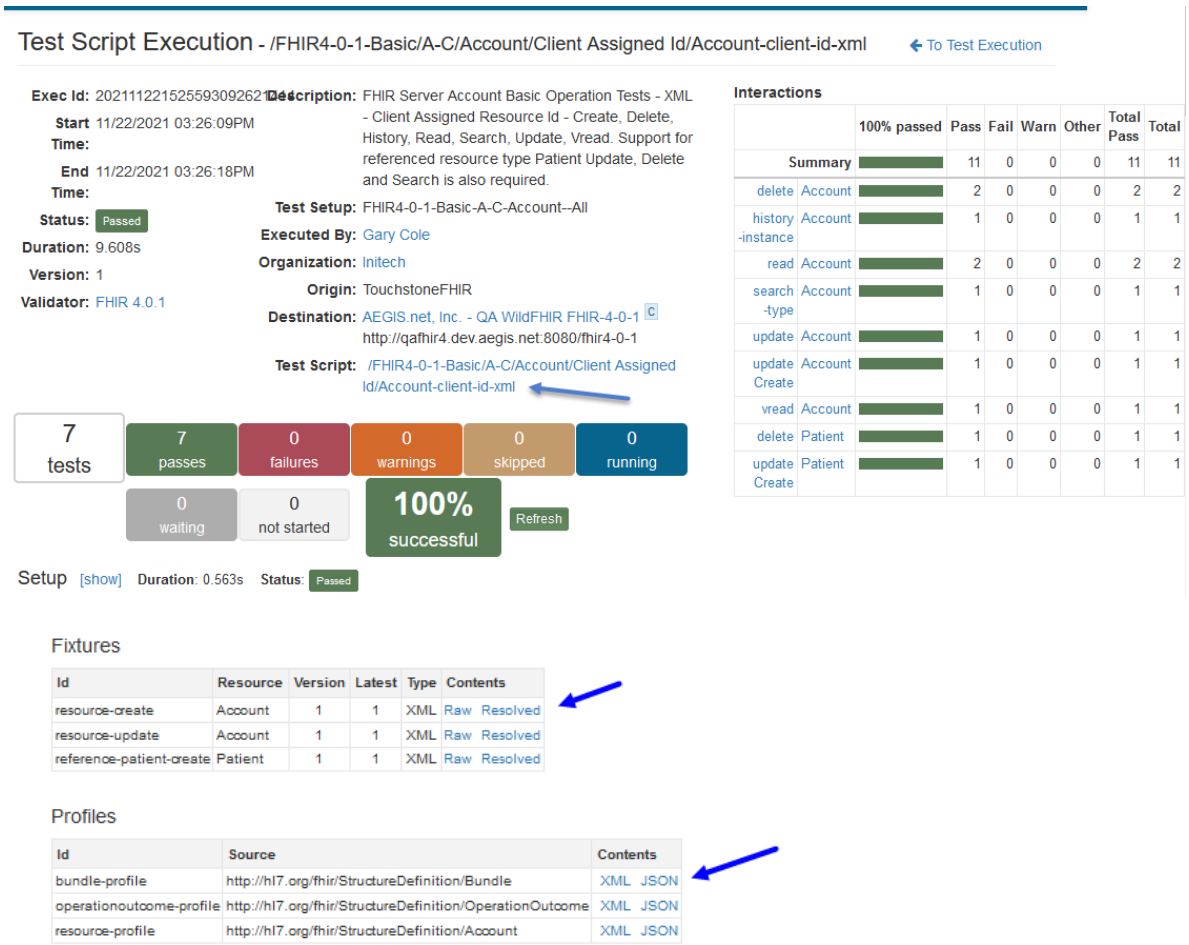


4. You can click on the `Request Body` and `Response Body` links to get the payloads in a separate window:



If you click on the `Test: Step3-UpdatePatient` link again (above), it will collapse this Test view and get you back to the initial view.

The `Test Script` link will take you to the actual Test Script that was executed for this test execution (and not the latest version that's there in the system). That's important for research. Similarly, the fixture and profile links will take you to the actual fixtures and profiles used during test execution:



For more information on fixtures, profiles, and variables please refer to the TestScript FHIR specification at http://build.fhir.org/testscript.html.

4.6 FAQ

1. I have registered into Touchstone but am unable to execute tests. To execute tests, you must either create a new organization or become a member of an existing organization.
2. When I try to re-execute a previous test execution, it says that the Test Setup could not be found. You probably deleted the Test Setup. Just recreate it in Test Definitions screen.
3. When I try to re-execute a previous test execution or click on a previous Test Setup, the test scripts section appears empty. The test scripts in your test setup have likely been removed from the system. This may not be obvious. If the path to the script changes, then the script is treated by the system as a new one. You can recreate your test setup in Test Definitions screen.
4. When executing a client-side test script, I have submitted what the system asked me to but my operation execution is still stuck on `Waiting for Request` Make sure you are sending your `USER_KEY` in the request header. The sections Test execution matching and Exchanges screen cover this in great detail.
5. When creating a Test Setup I get an error saying that the test scripts are not using the same validator. Touchstone will enforce one Validator per Test Setup. Either make sure that all the tests you want to run are using the same Validator or create seperate Test Setups for any Tests that require a different Validator.

6. Why is my Bearer Token different in the Test Execution than the one I provided. Touchstone is Base64Encoding the *operation.requestHeader.value*. More information on this can be found in OAuth Capabilites.
7. The server under test is returning a 401 error instead of good response. Verify the Test System setup. Is the Client ID/Client Secret correct for Oauth2 enabled systems? Are the scopes correct?
8. I cannot get a response from my server at all. Is your endpoint correct? Is your Test System accessible outside your firewall? Does your organization need to white-list Touchstone? Check with Touchstone_Support@aegis.net for full list of IP ranges.
9. My server sent a CapabilityStatement, but Touchstone indicated a problem. Touchstone will automatically validate a Test System CapabilityStatement when pulled, even when the system is not yet under test. Touchstone WILL still allow that system to continue testing, but will remind the tester of the issues.
10. My server doesn't have a CapabilityStatement Touchstone will still allow that system to continue testing, but will remind the tester at each test that it is required per the FHIR specification. Systems that are identified as SMART-on-FHIR systems will need to provide a SMART Well-Known Configuration for most testing in Touchstone.
11. How do fixtures get uploaded to Touchstone and referenced by TestScripts? Fixtures are uploaded along with the TestScripts and should be referenced by a relative/absolute file path of Touchstone's folder structure. Best practice is to include a _reference/resources folder and use "./_reference/resources/filename.json" as the reference value.
12. How do I view Test Executions and how do I stop one that is running? Test Executions can be viewed by clicking on the Test Execution History link on the left side of the page, under Test Executions. If you have a Test Execution that is currently running but you want to stop it, in the list of executions, you can click the square `STOP` icon on Test Execution you want to stop. If you are on the page of the Test Execution itself, you can click on the `STOP` button in the top right corner.

CHAPTER

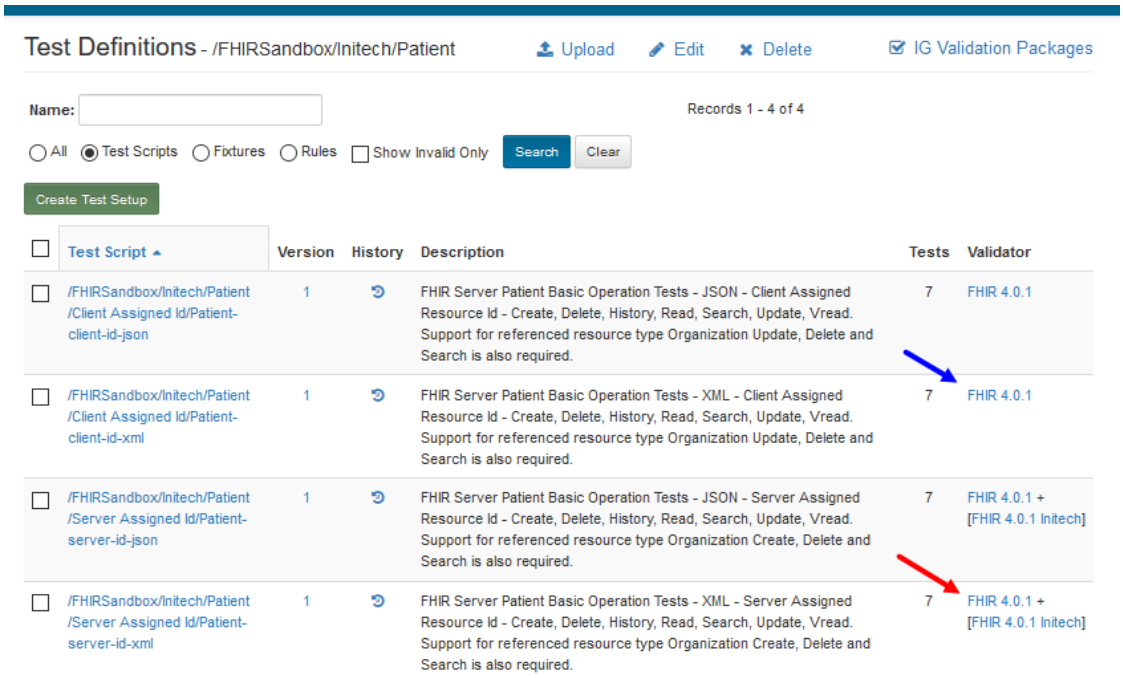
FIVE

MULTI-PROFILE TESTING

Touchstone supports multi-profile testing, where a single test can be run against multiple different validators. Test Scripts and fixtures are validated both upon upload and at test execution. Note, to associate a Test Script to multiple validators, those additional validators need to be set up in Touchstone. Please contact Touchstone_Support for more information.

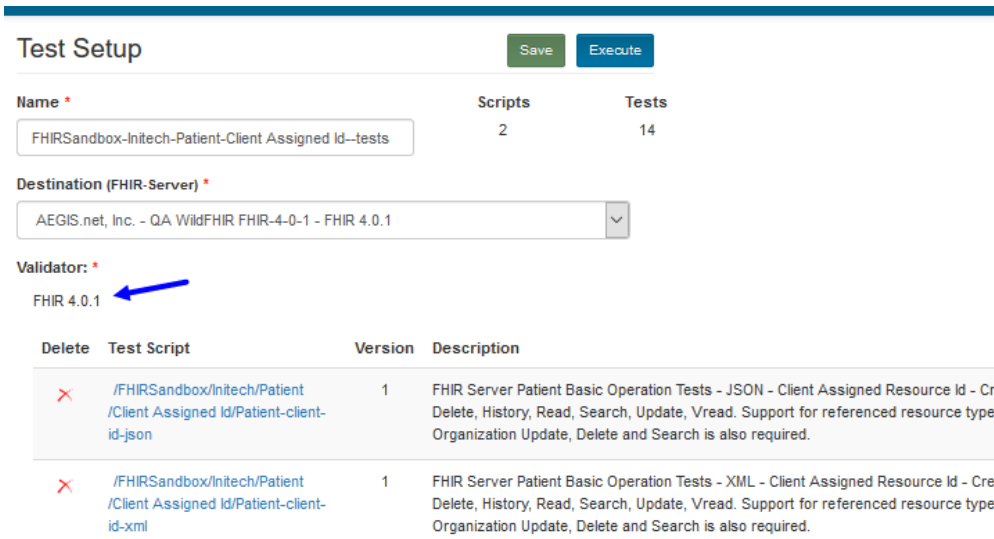
5.1 Validator Testing

The Test Definitions in Touchstone indicate to users which Validator(s) are associated to a Test Script. Validators are associated to a Test Script at the time the script is uploaded to Touchstone. The Validator column in Test Definitions will indicate if the Test Script is associated to a single (blue arrow) or multiple validators (red arrow).

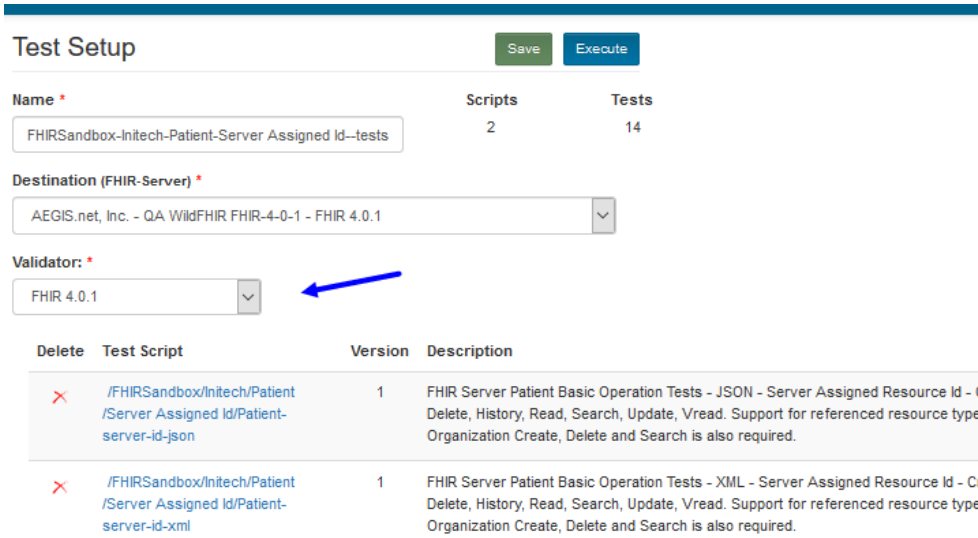


These differences can also be seen at test setup.

- **With just one Validator selected:**



- **With multiple Validators selected:**



Note: Test Scripts within a Test Setup must all have the same default Validator(s).

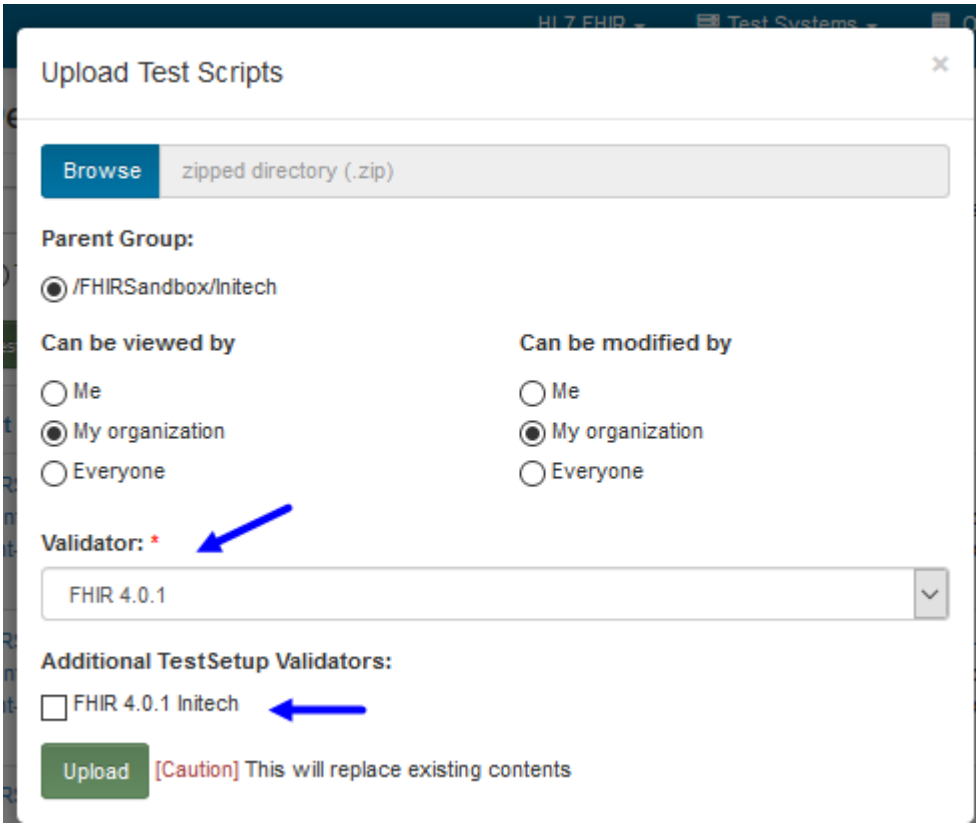
5.1.1 Uploading

While any Test Editor can set the Base Specification Validator when uploading Test Scripts, only users that have the IGV Package Uploader Role will have the ability to view and select a custom Validator.

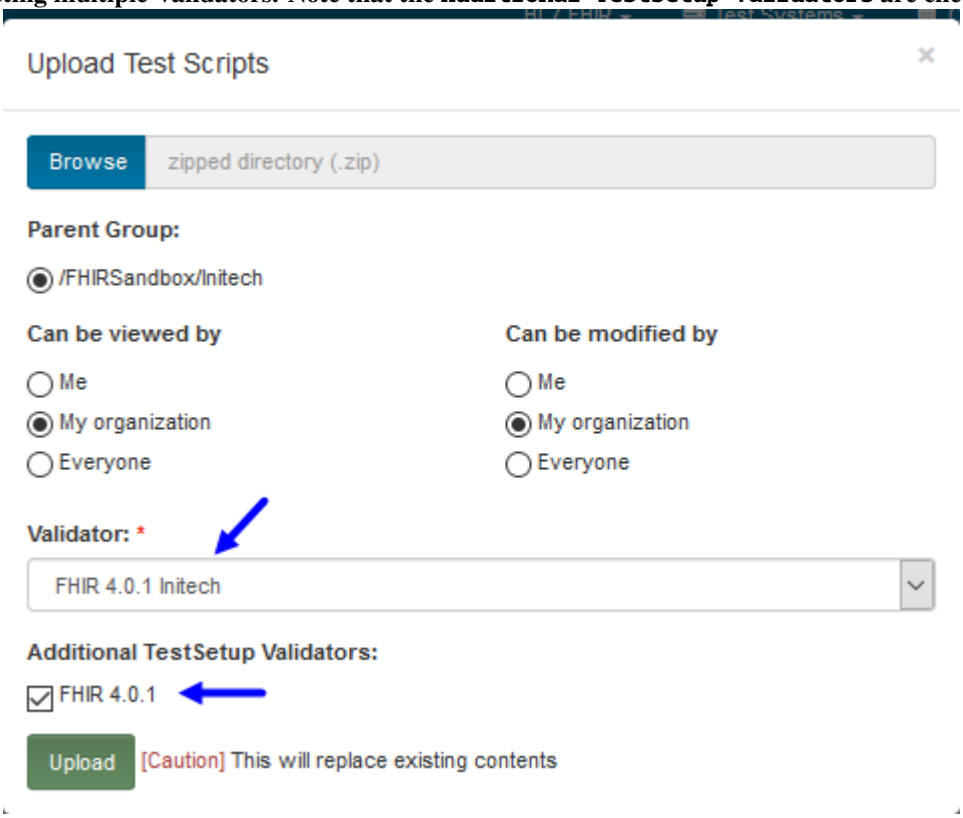
Validators for a Test Script or Test Script group can be set at either upload or edit. Regardless of whether a single or multiple validators are selected during upload, all artifacts within the upload package will have the selected Validator(s) associated to them.

Base Specification Validators are chosen from the Validator drop-down, and if any additional Validators are available for that Base Specification Validator, they will be listed below that selection with a checkbox to select.

- **Selecting a Base Spec Validator. Note that one Validator is selected in the drop down menu and none of the `Additional T`**



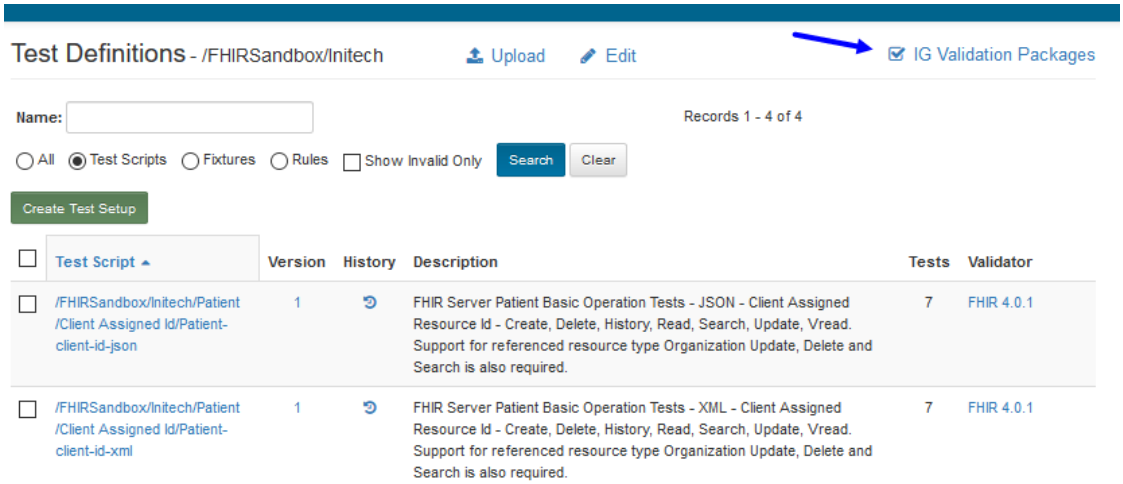
- **Selecting multiple Validators. Note that the `Additional TestSetup Validators` are checked to give the tester options at**



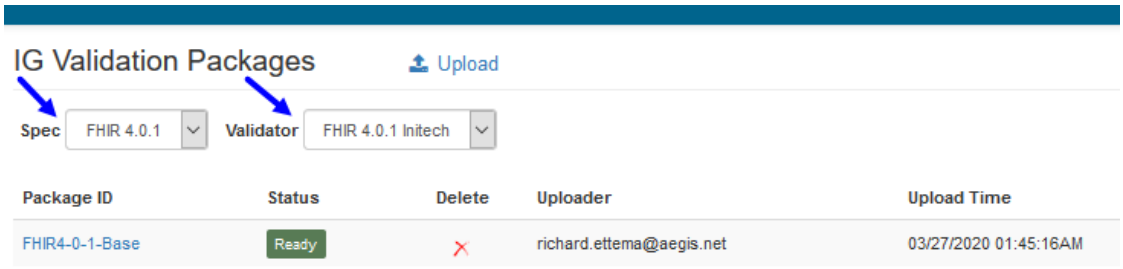
5.2 Updating Validators

Validators can be edited and updated by any user with the IGV Package Uploader Role. This role is limited to users in Orgs with an Enterprise Subscription. The original Role and Validator assignment will be handled by Touchstone Support, but any Org Rep with the IGV Package Uploader Role can assign the same IGV Package Uploader privileges to another User in the same organization.

Any Users with the appropriate role can navigate to the `IG Validation Packages` link on the Test Definitions screen.



At this page there are multiple drop down menus to select which validator you want to edit. The `Spec` menu filters the available validators by the base FHIR spec they depend on, and the `Validator` menu that lets you choose from your list of assigned Validators which one you want to edit.

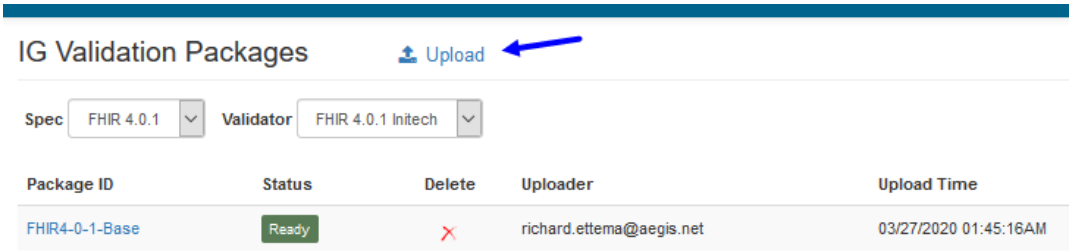


Once you have a Validator selected for editing, you can either upload a validation package (and replace an existing package of the same name) or delete a validation package.

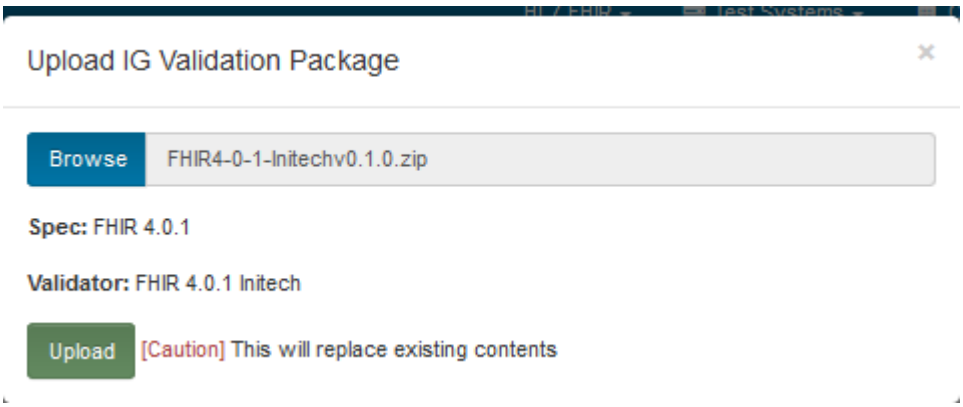
> **Warning:** Either deleting or replacing a Validation Package are permanent. Make sure that you have a back-up of your validation packages in your own repository before removing or replacing one within Touchstone.

To upload a Validation Package:

1. Click the `Upload` link at the top of the page and



2. Browse to the file containing the Validation Package you want to upload.

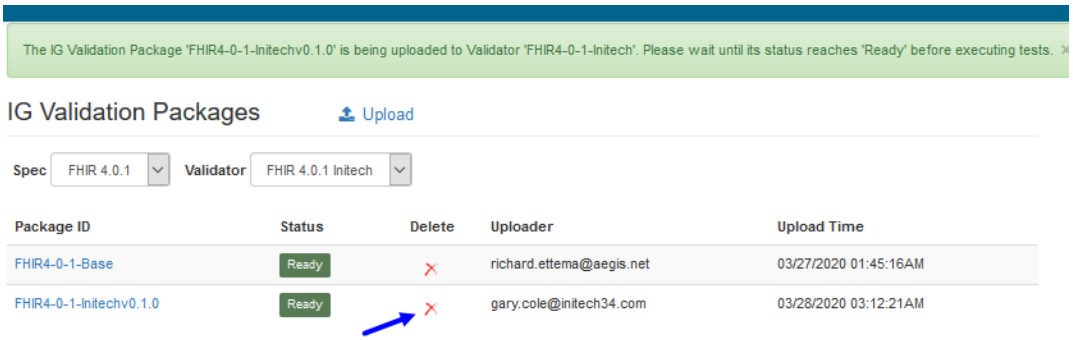


- **Validation Package – The collection of resources from a FHIR® Implementation Guide to be used by a FHIR® Validation**
  - ZIP package format - a ZIP archive of the validation resources in a flat folder structure; i.e. no root or sub-folders present.
  - TGZ package format - an NPM (Node Package Manager) gzipped tarball containing the validation resources in FHIR® directory structure defined via a package.json file.

If the package being uploaded is too large to process quickly, then Touchstone will handle the upload asynchronously and an `Uploading` prompt will display to indicate that the upload is still in-progress. During the Validation package upload, any Test Script that is attempted to run against that Validator will fail.

To delete a Validation Package:

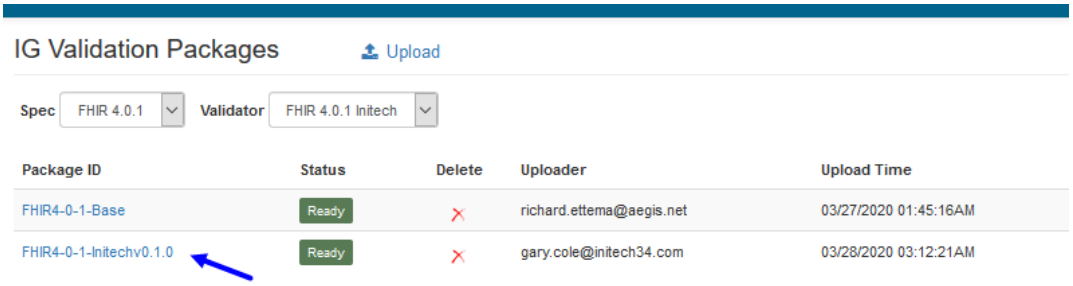
1. Click on the red `Delete` X for the row you want to delete.



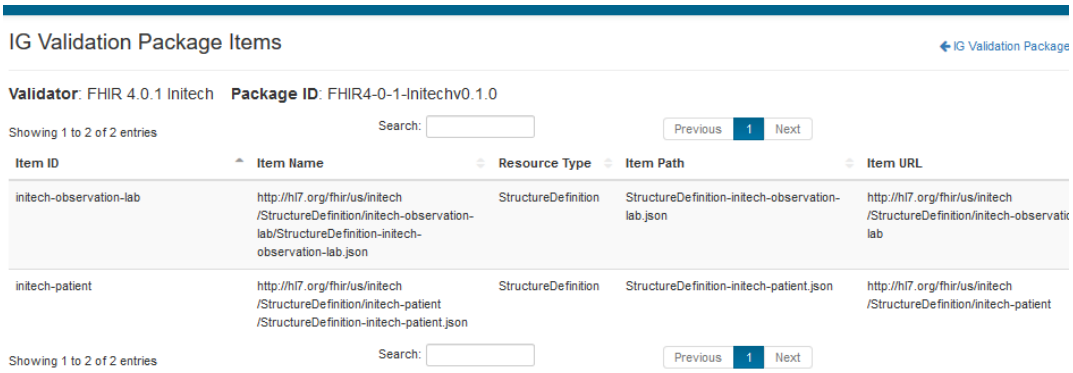
Inspecting Validation Packages:

Any Validation Package in one of your Validators can be inspected to see what resources it contains.

1. Click on the Package ID of the Validation Package you want to inspected



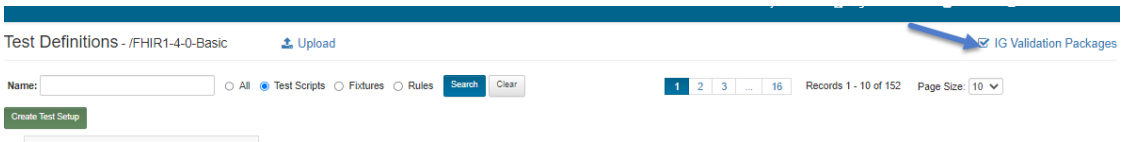
2. If the Validation Package is very large, this screen may take a few additional moments to load fully.



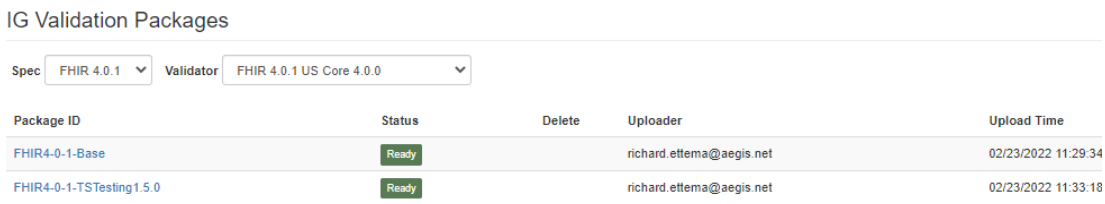
- Item ID – Logical id of the structure definition
- Item Name – Name for the structure definition (computer friendly)
- Resource Type – Type defined or constrained by the structure
- Item Path – The file structure inside the upload file that points to the resource
- Item URL – Canonical identifier for the structure definition, represented as a URI (globally unique)

5.3 Viewing Validators

Users can view the Implementation Guide (IG) packages loaded in a particular Validator by clicking the "IG Validation Packages" hyperlink on the Test Definitions page.



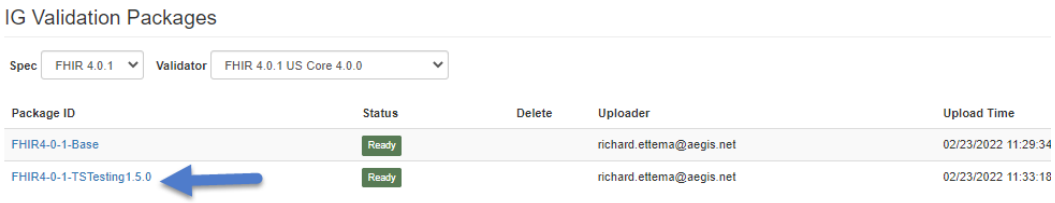
On the IG Validations Package page select the FHIR version from the Spec drop down. The Validator drop down will populate with the list of Validators for the selected Spec version. Select a Validator from the drop down to view the IG Validation Packages loaded in the selected Validator



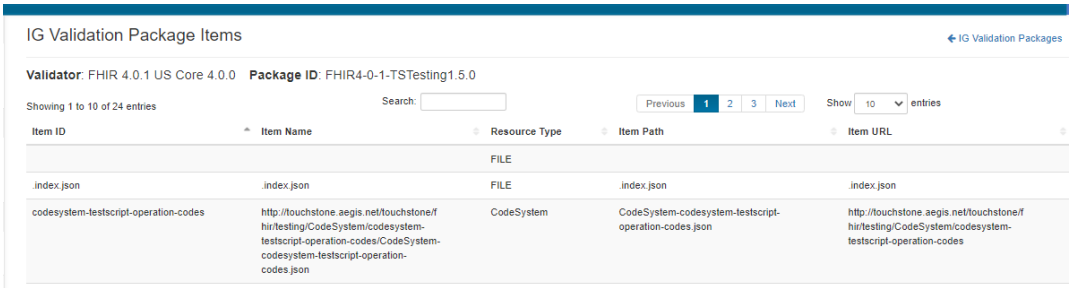
5.3.1 To inspect the contents of a Validation Package

Any Validation Package in one of your Validators can be inspected to see what resources it contains.

1. Click on the Package ID of the Validation Package you want to inspected



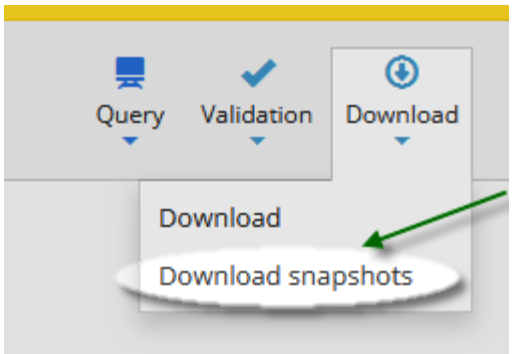
2. If the Validation Package is very large, this screen may take a few additional moments to load fully.



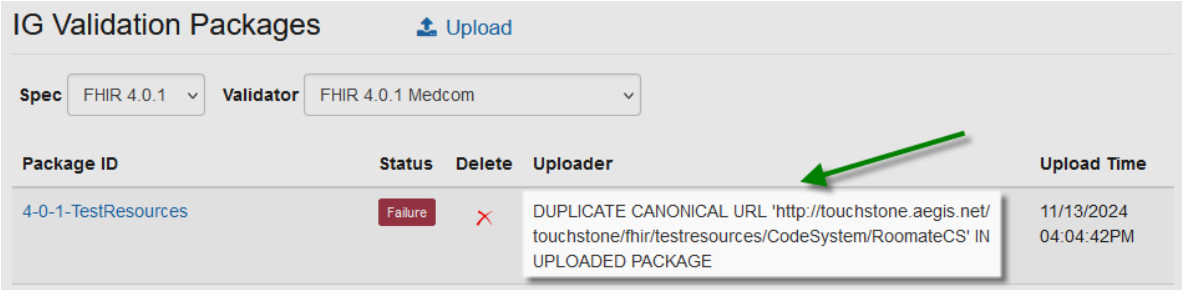
- Item ID – Logical id of the structure definition
- Item Name – Name for the structure definition (computer friendly)
- Resource Type – Type defined or constrained by the structure
- Item Path – The file structure inside the upload file that points to the resource
- Item URL – Canonical identifier for the structure definition, represented as a URI (globally unique)

5.4 FAQ

1. IG validation packages downloaded from Simplifier do not always upload successfully to my Touchstone Validator. Why is that? When downloading an IG validation package from Simplifier, the package with generated snapshots must be downloaded for subsequent upload to the Touchstone Validators as shown in this screen shot:



2. My IG validation package upload failed. Where can I find the cause? IG validation package upload failure message(s) are displayed in the Uploader column for the corresponding Package ID:



CHAPTER

SIX

ORG GROUPS

Organizations can come together to form Org Groups. Access to test systems, test execution results, and test scripts can be controlled at the Org Group level.

6.1 Creating Org Group

Org Groups can be created by Touchstone Admininstrators. Please contact us at touchstone_support@aegis.net if you need an Org Group to be created. We will analyze your needs and see if Org Group will be a right fit. If so, we will create the Org Group and assign an Org Group Rep who will be responsible for managing organization membership into the Org Group.

To explain the concepts revolving Org Groups, we will use the following two Org Groups for the examples in this section:

Org Groups

Name: Search Clear Records 1 - 2 of 2

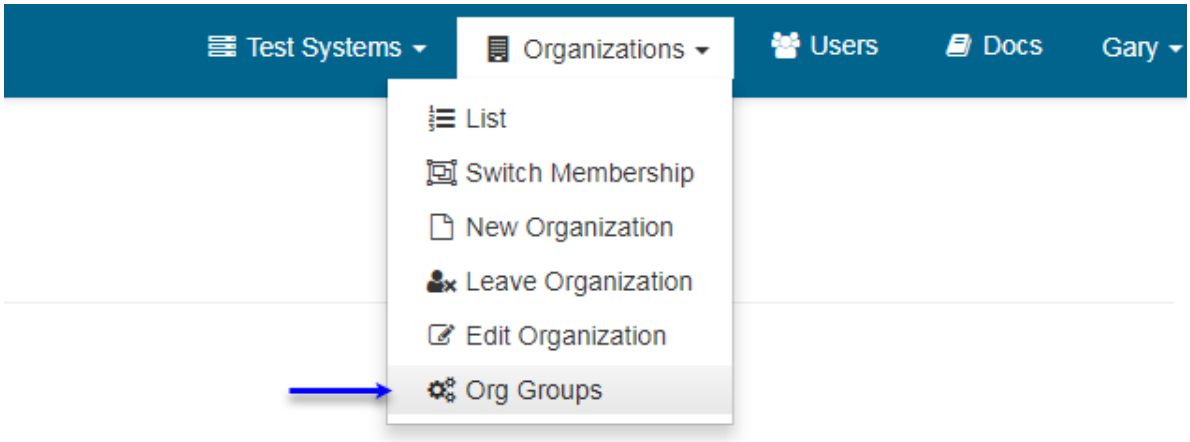
Name	Exec Visibility	Action	Description	Org Group Reps	Pending Orgs	Approved Orgs
Org Group A	Org Group (Open)	Join	Organizations that become part of this Open org group will be able to see each others' test executions.	Org1Rep		Organization0001,Organization0002,Organization0003
Org Group B	Org (Private)	Join	Organizations that become part of this Private org group will not be able to see each others' test executions.	Org4Rep		Organization0004,Organization0005,Organization0006

- Org Group A has three approved organizations as its members: Organization0001, Organization0002, and Organization0003.
- Org Group B has three approved organizations as its members: Organization0004, Organization0005, and Organization0006.
- Org Group A has Org1Rep as its Org Group Rep
- Org Group B has Org4Rep as its Org Group Rep

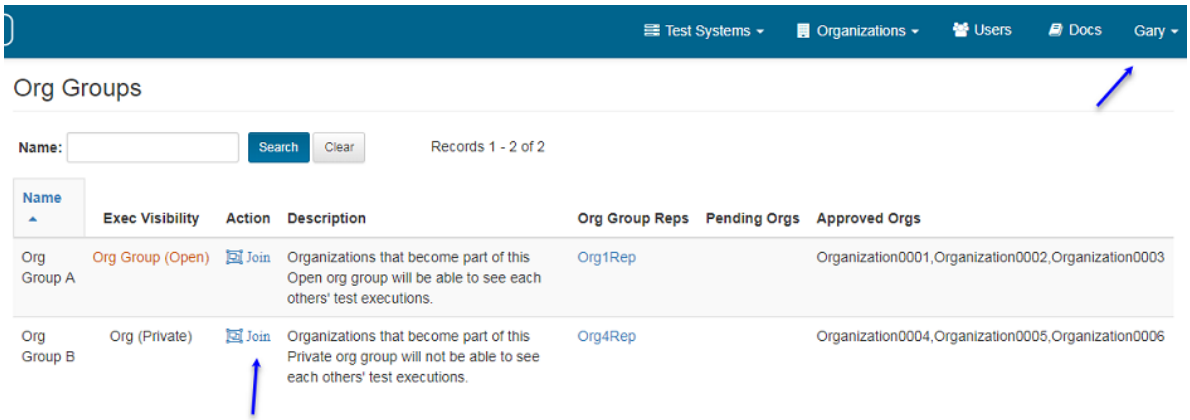
6.2 Joining Org Group

To have your organization join an existing Org Group, you can take the following steps:

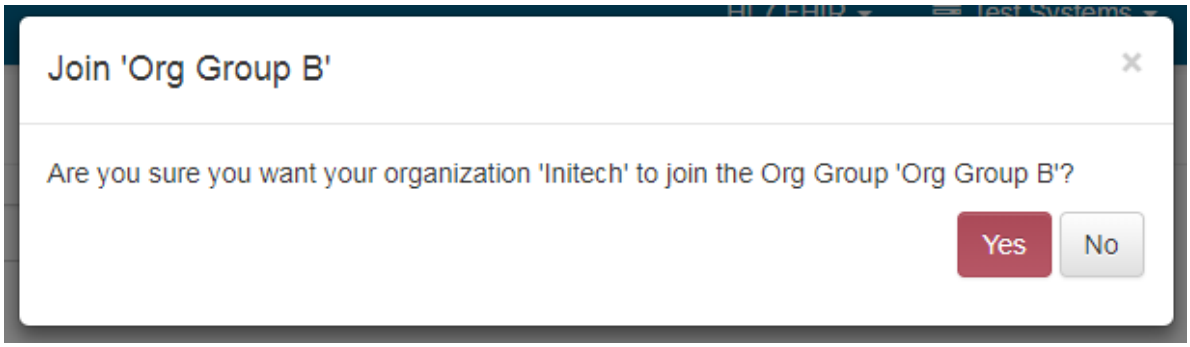
1. Sign In in as the Org Rep of your organization.
2. Click on Organization / Org Groups menu link



3. Click the `Join` link of the Org Group you wish you join. Gary as the Org Rep of Initech decides to have Initech become a member of Org Group B.

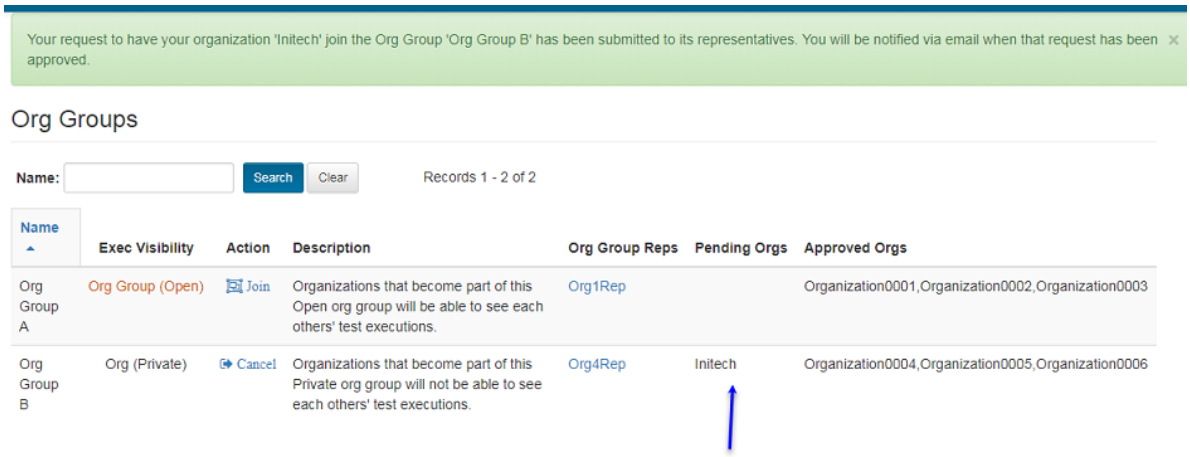


4. After you confirm, your request will be submitted to the Org Group Rep of that Org Group. In this case it will be submitted to Org4Rep:



5. After your request has been submitted, you will need to wait for approval by the Org Group Rep. You will be notified of approval (or rejection) via email. Check your Spam folder in your email system in case the emails get

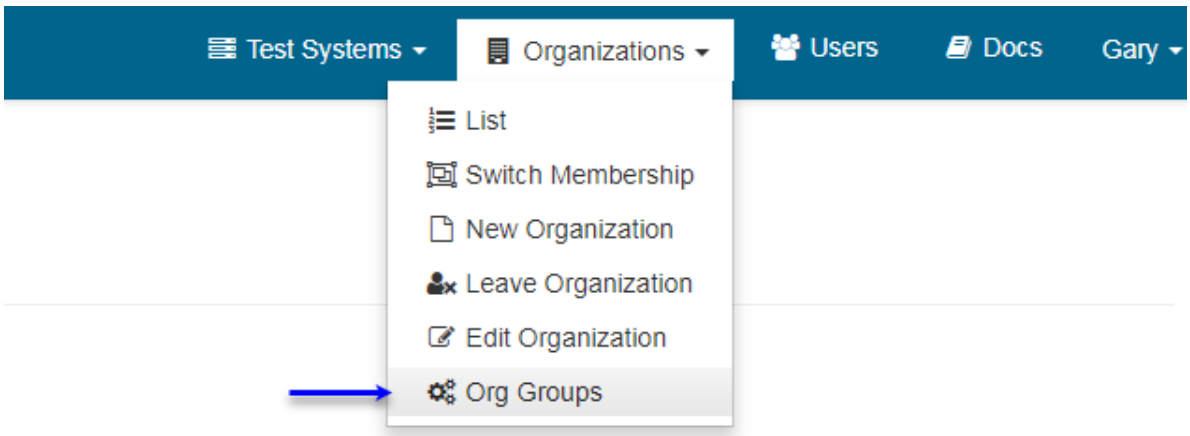
directed there.



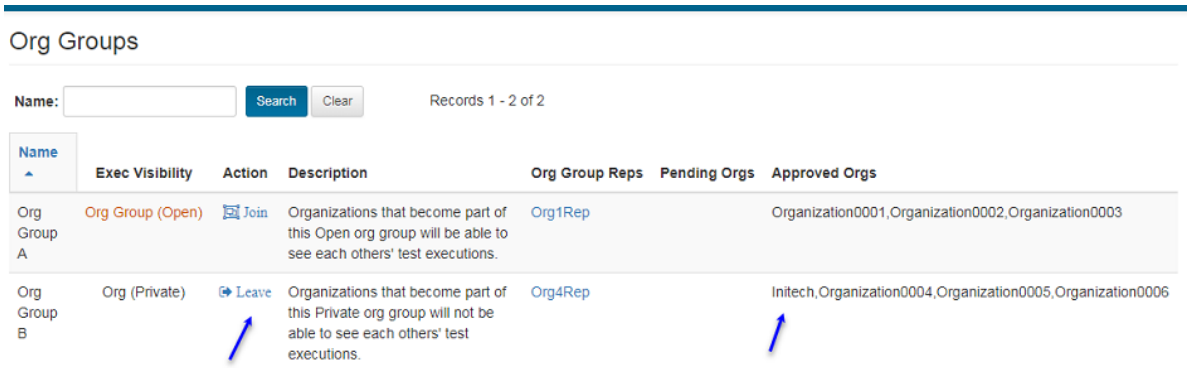
6.3 Leaving Org Group

If you wish for your organization to leave an Org Group, you can take the following steps:

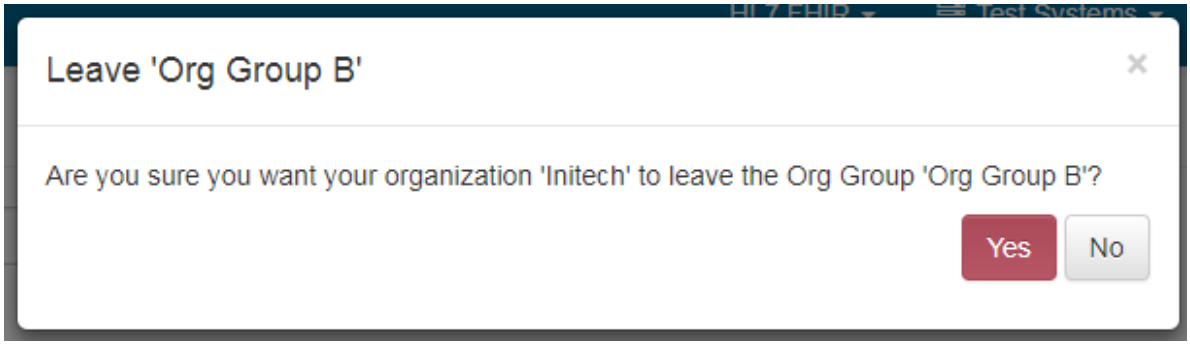
1. Sign In as the Org Rep of your organization
2. Click on Organization / Org Groups menu link



3. Click the `Leave` link of the Org Group you wish you join:



4. After you confirm, your organization will no longer be part of that Org Group:

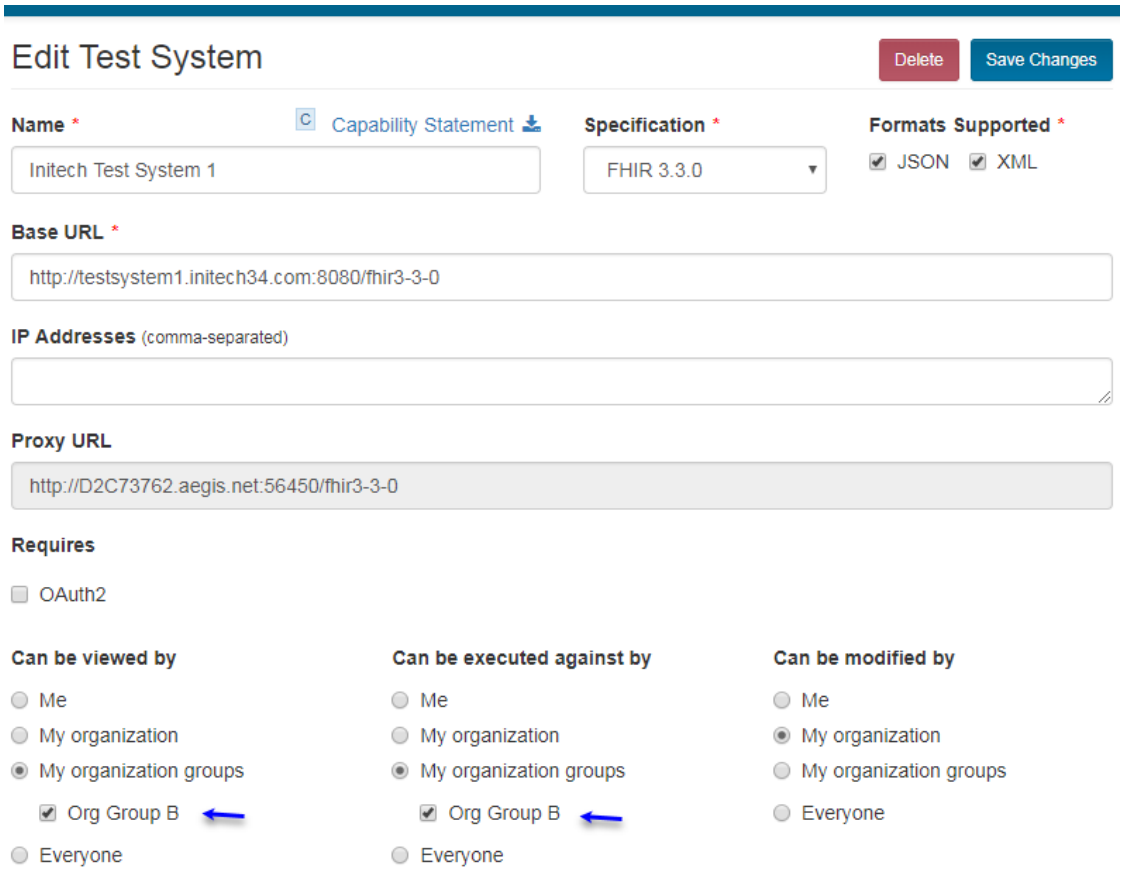


Gary clicks on Cancel and stays in Org Group B.

6.4 Access

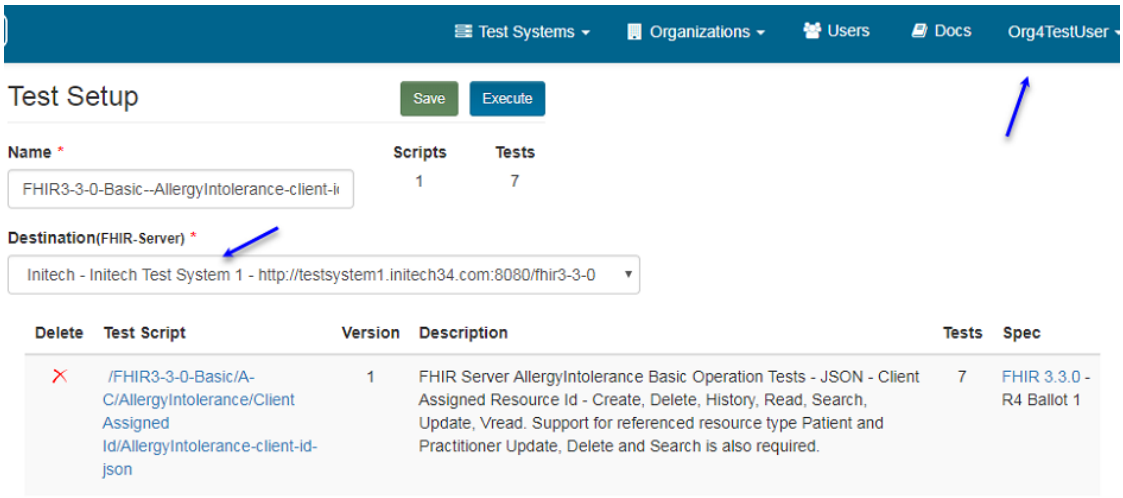
6.4.1 Test System access

If your organization belongs to an Org Group, you can widen the access rights for viewing and execution against your test system to the Org Group level:

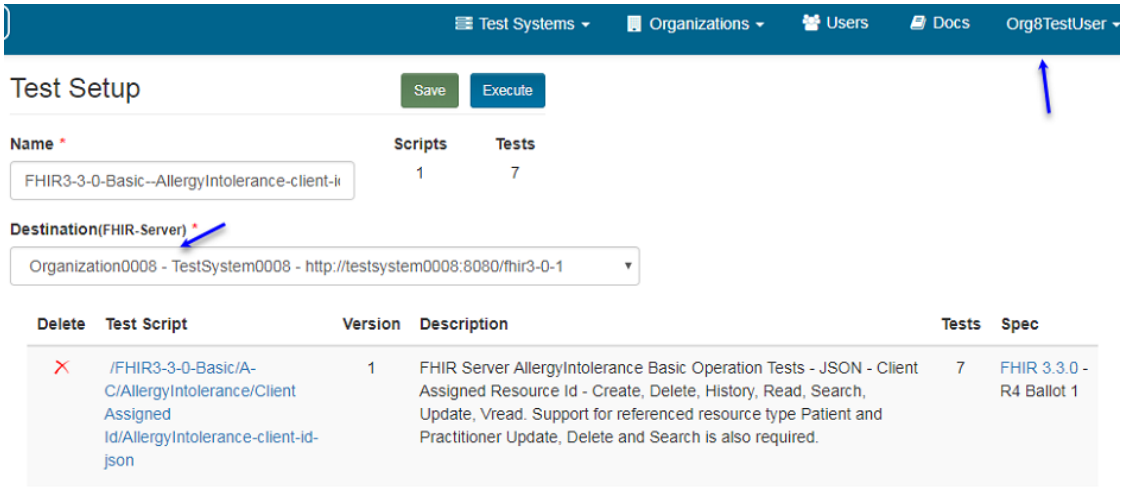


In the case above, the user is permitting all organizations that belong to `Org Group B` to execute test scripts against the test system.

User from Organization0004 (that belongs to Org Group B) can execute test scripts against `Initech Test System 1`:



On the other hand, user from Organization0008 (which does **not** belong to Org Group B) will **not** be able to execute test scripts against `Initech Test System 1`. The test system `Initech Test System 1` will not appear in the `Destination` drop-down:

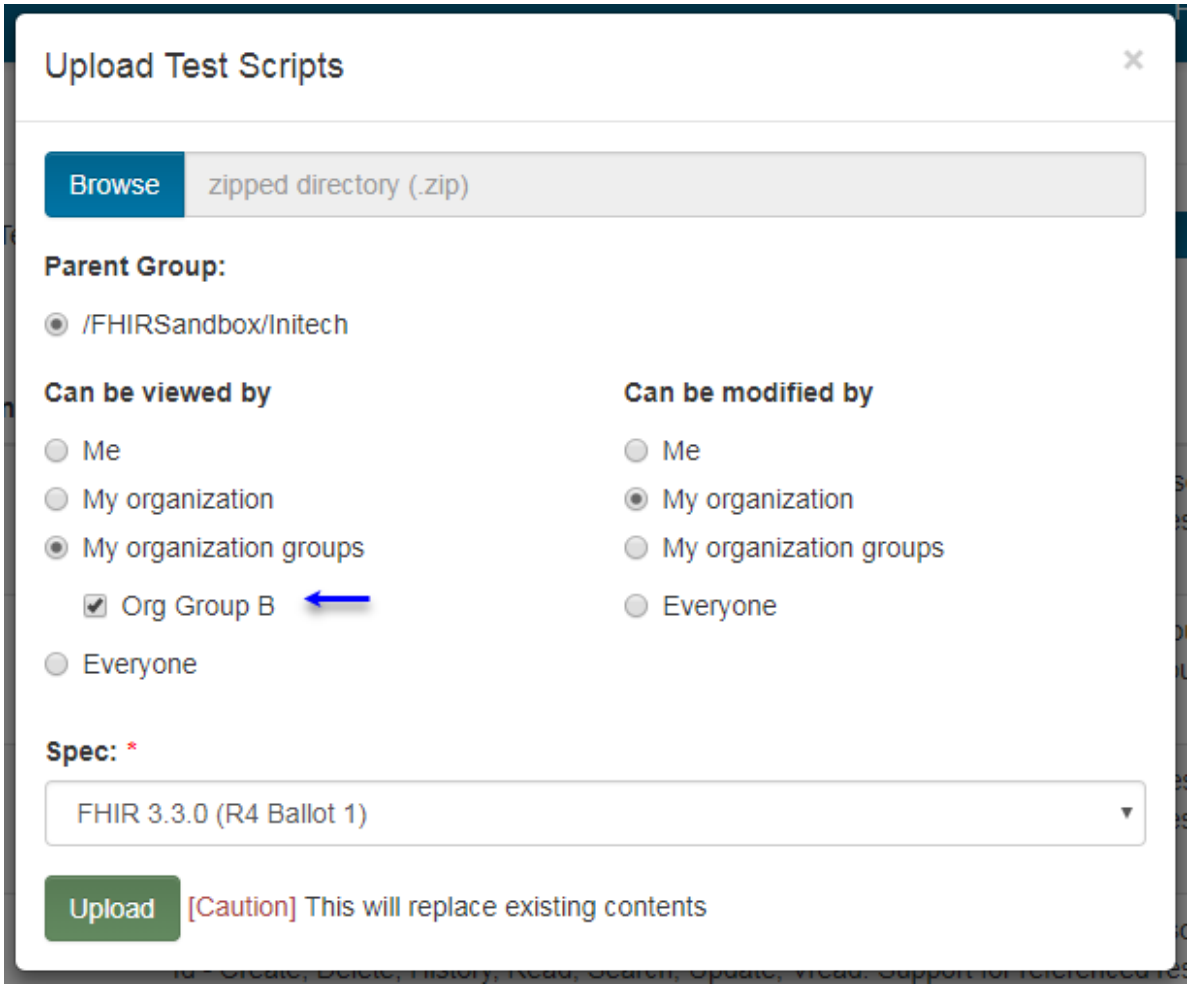


6.4.2 Test Definition access

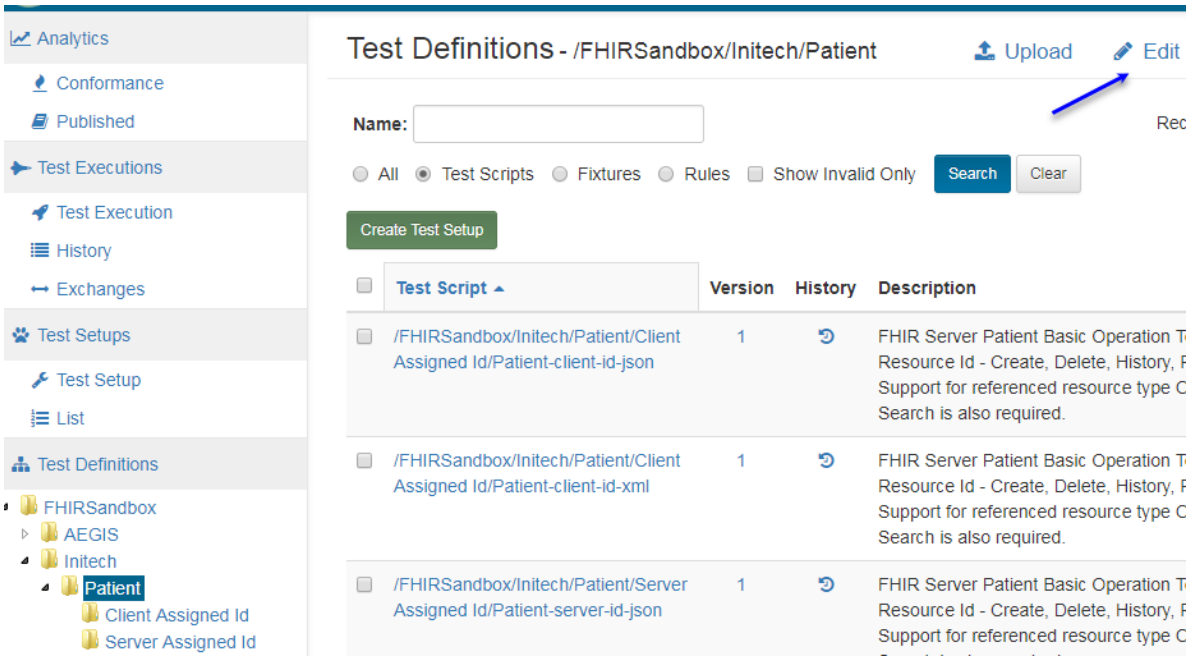
If your organization belongs to an Org Group, you can widen the access rights for viewing of Test Groups and Test Definitions to the Org Group level.

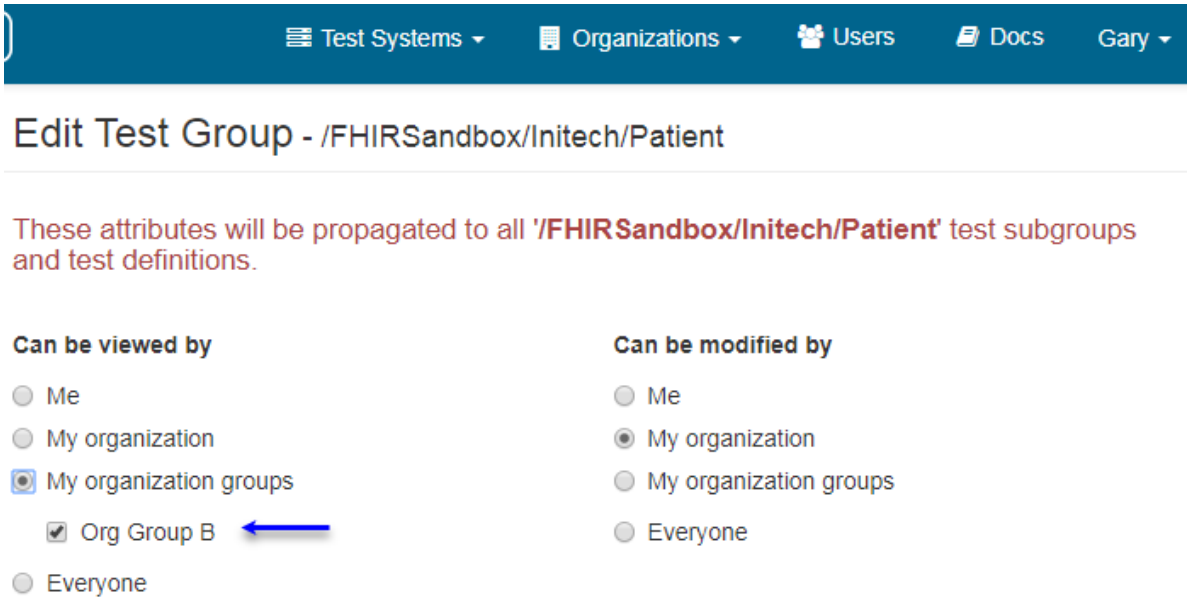
There are two ways to do that:

1. You can assign Org Group access during upload time:

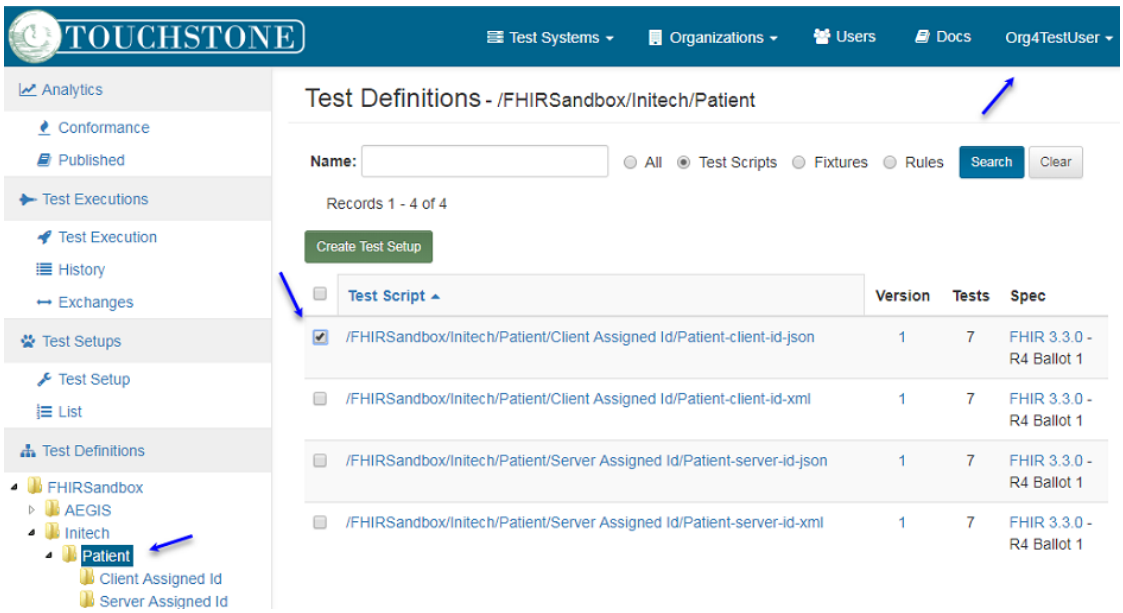


2. You can edit an existing test group and modify its access:

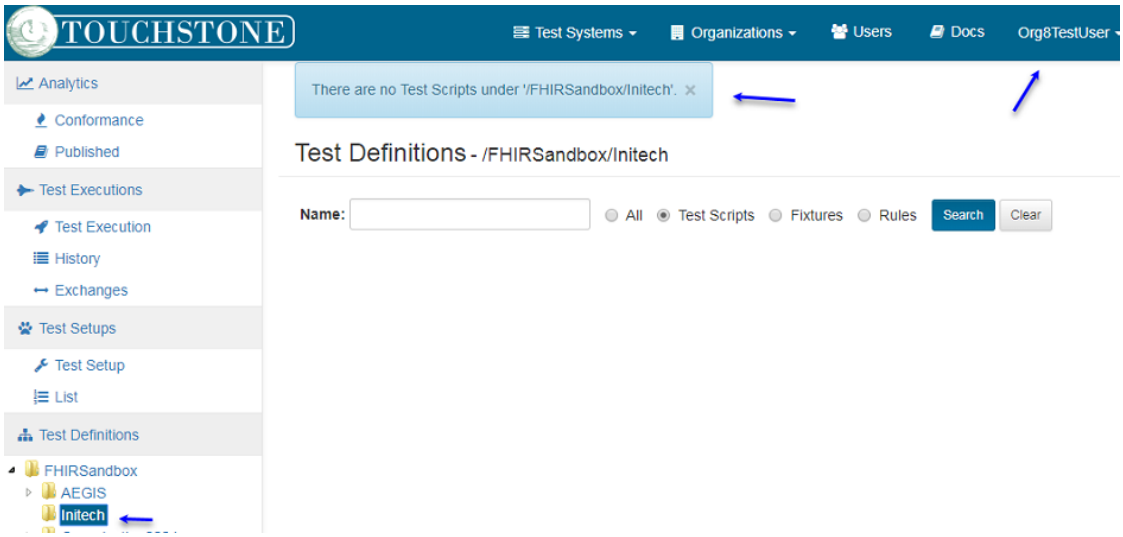




After Initech widens the access to `Org Group B`, user from Organization0004 (that belongs to Org Group B) will be able to access the `Patient` test group in Initech and execute test scripts in that test group:



But user from Organization0008 (which does **not** belong to Org Group B) will **not** be able to access the `Patient` test group in Initech and its test definitions:

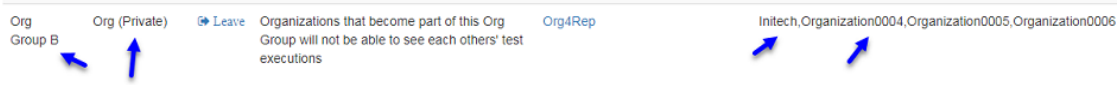


6.4.3 Test Results access

If your organization belongs to an Org Group, viewing of test results are automatically widened to the Org Group level if the Org Group is Open.

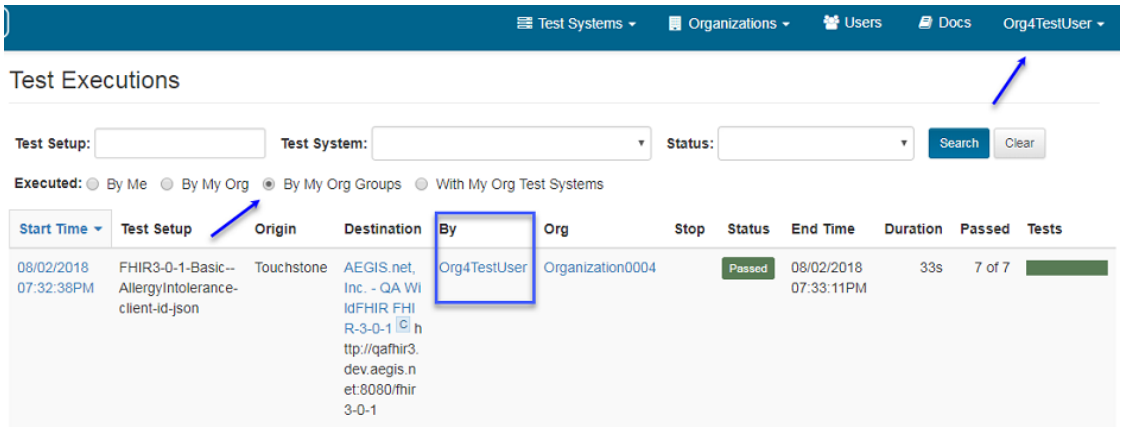
Org Group Private

In the case below, Initech and Organization0004 are members of a Private group.

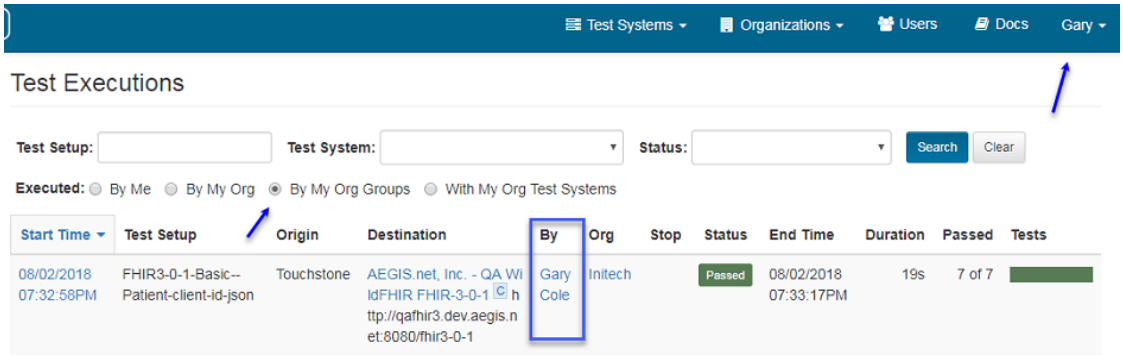


That means members **cannot** view each others' test executions.

Notice that Gary from Initech can view only Initech's test executions:

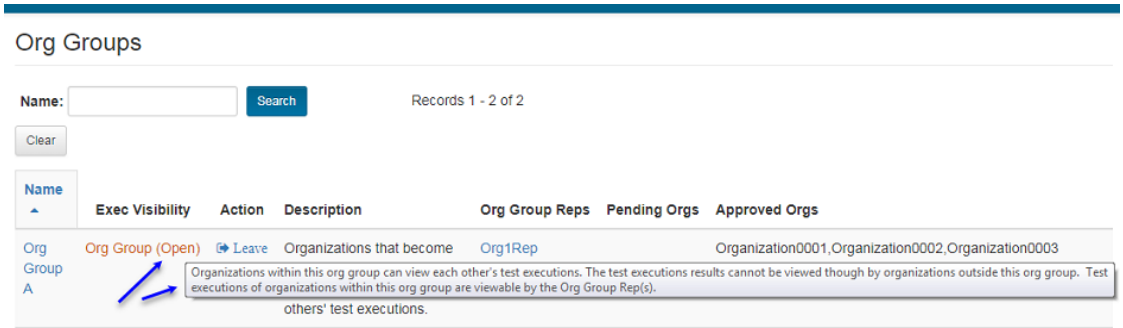


And Org4Rep from Organization0004 can view only Organization0004's test executions:



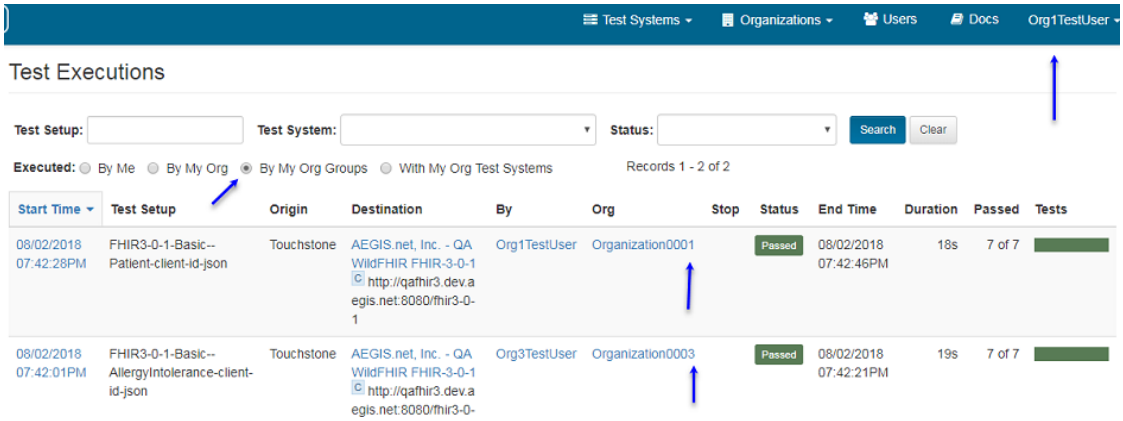
Org Group Open

In the case below, Organization0001 and Organization0003 are members of an Open group.

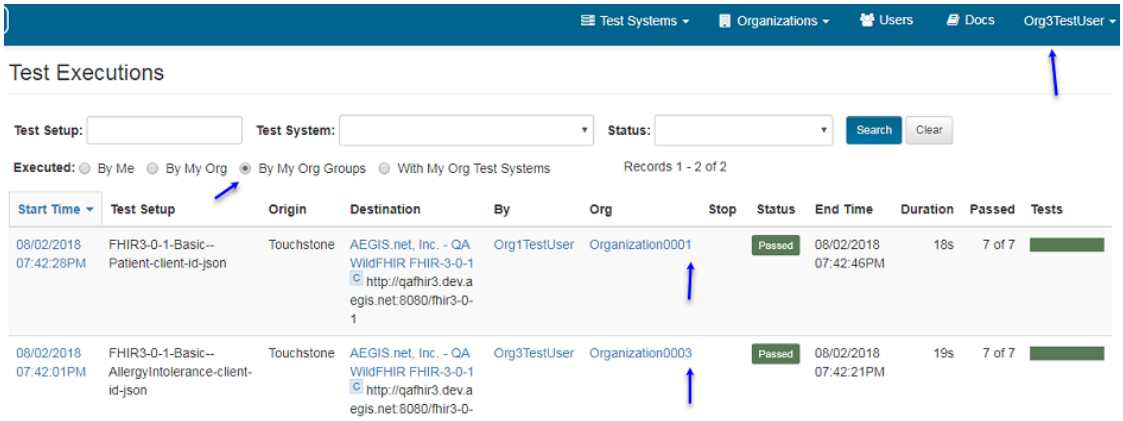


That means members **can** view each others' test executions.

Notice that Org1Rep from Organization0001 can view test executions by both Organization0001 and Organization0003:



Similary, Org3Rep from Organization0003 can view test executions by both Organization0001 and Organization0003:


Test Systems
Organizations
Users
Docs
Org3TestUser
Test Executions
Test Setup:
Test System:
Status:
Search
Clear
Executed: By Me By My Org By My Org Groups With My Org Test Systems
Records 1 - 2 of 2
Start Time
Test Setup
Origin
Destination
By
Org
Stop
Status
End Time
Duration
Passed
Tests
08/02/2018 07:42:28PM
FHIR3-0-1-Basic--Patient-client-id-json
Touchstone
AEGIS.net, Inc. - QA WildFHIR FHIR-3-0-1 http://qafhir3.dev.aegis.net:8080/fhir3-0-1
Org1TestUser
Organization0001
Passed
08/02/2018 07:42:46PM
18s
7 of 7
08/02/2018 07:42:01PM
FHIR3-0-1-Basic--AllergyIntolerance-client-id-json
Touchstone
AEGIS.net, Inc. - QA WildFHIR FHIR-3-0-1 http://qafhir3.dev.aegis.net:8080/fhir3-0-
Org3TestUser
Organization0003
Passed
08/02/2018 07:42:21PM
19s
7 of 7

CHAPTER

SEVEN

CONFORMANCE TESTING

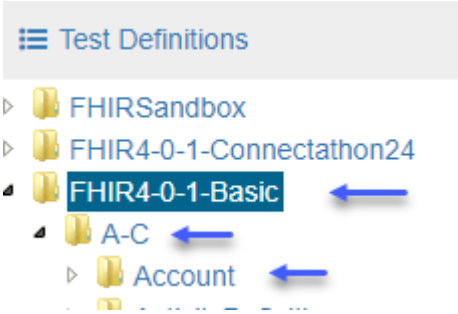
Users can monitor the conformance of tests systems to a specification based on **Conformance Suites** available in Touchstone.

Prior to Touchstone 5.0.0, users were limited to pre-configured conformance tests in Touchstone. The following Conformance-testing capabilities have been added in Touchstone 5.0.0:

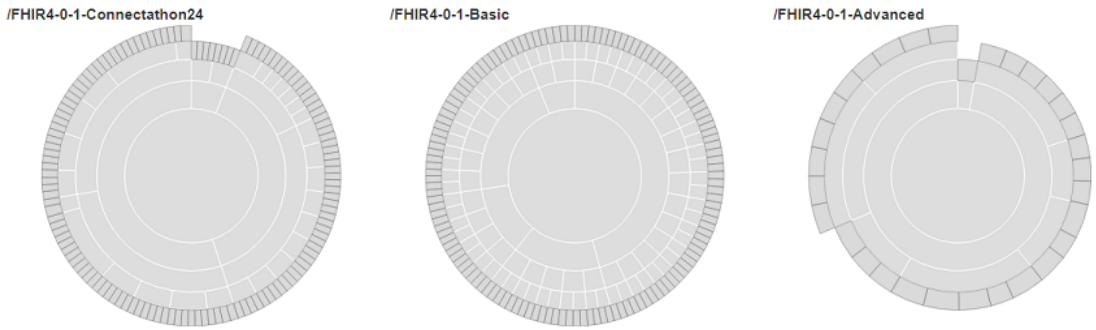
- Organizations (with the appropriate subscription level) can create their own Conformance Suites with custom test groups and test scripts.
- Client (Peer-to-Peer) interactions can be monitored as part of Conformance (in addition to Server interactions).
- Organizations can optionally make their Conformance Suites accessible to members of their Organization only or to members of their Org Group or to everyone.
- Conformance Suites and Results are versioned. This allows organizations to publish their results at any time and continue their testing without affecting published results.
- Suite authors can allows users to publish conformance results or prevent them from doing so.
- Suite authors can have conformance results tracked separately for XML and JSON formats.
- Suite authors can have conformance results tracked separately for supported interactions (in the test system's capability statement) or disregard the capability statement and include all interactions.

7.1 Conformance Suites

Test Groups allow one or more test scripts to form a unit that can be executed together as a batch.

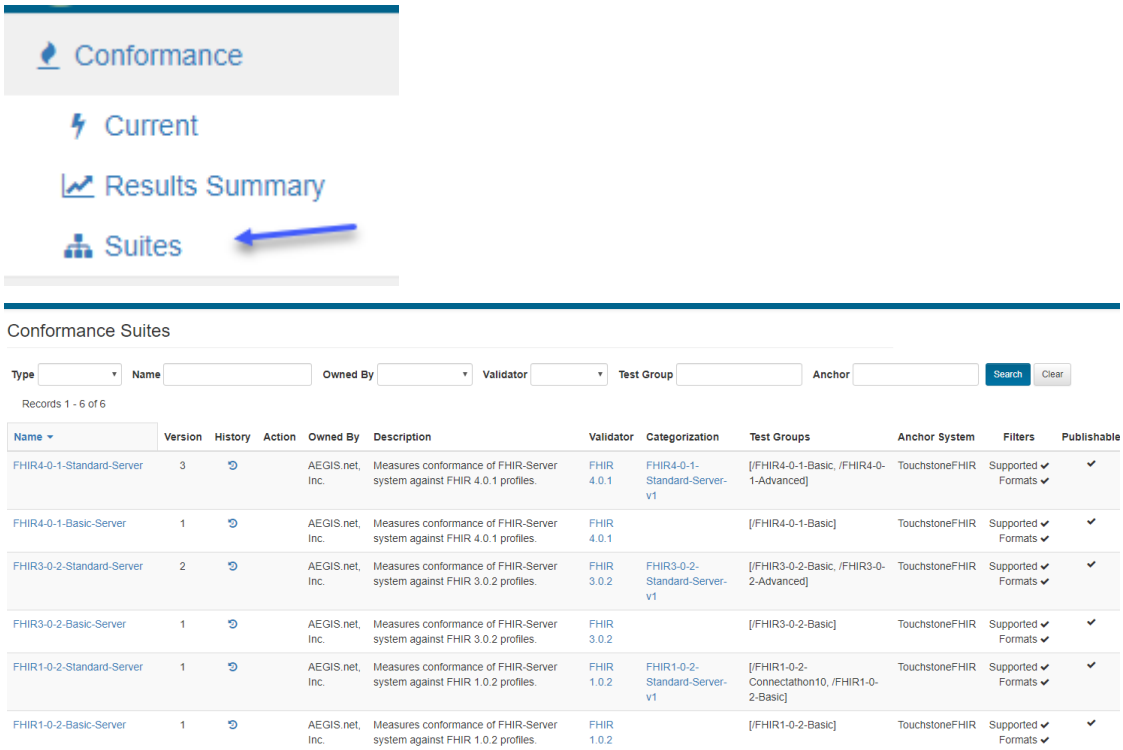


Conformance Suites allow one or more of those test groups to form a unit for tracking conformance of test systems to a specification.



7.1.1 Listing

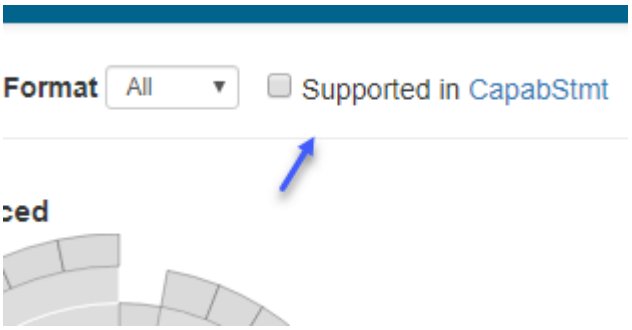
Conformance Suites can be accessed here or by navigating to Conformance / Suites menu link:



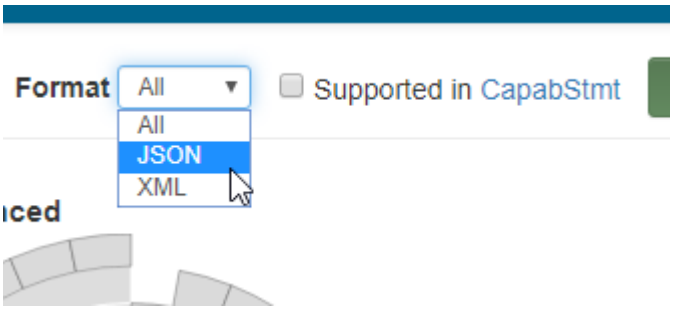
- Owned By – The organization that is maintaining the Conformance Suite definition.
- Validator – The AEGIS Validator that is used to validate request and response payloads in test script executions within the suite.
- Categorization – If a Conformance Suite is composed of more than one test group, then Categorization is required and interaction counts will be aggregated based on the Categorization definition.
- Test Groups – The list of test groups that form the Conformance Suite.
- **Anchor System**
  - If the suite is a **Server** suite, then the Anchor System will be a **Client** system that all **Server** SUTs will use for test executions so conformance results would be consistent in the suite.

  - If the suite is a **Client** suite, then the Anchor System will be a **Server** system that all **Client** SUTs will use for test executions so conformance results would be consistent in the suite.

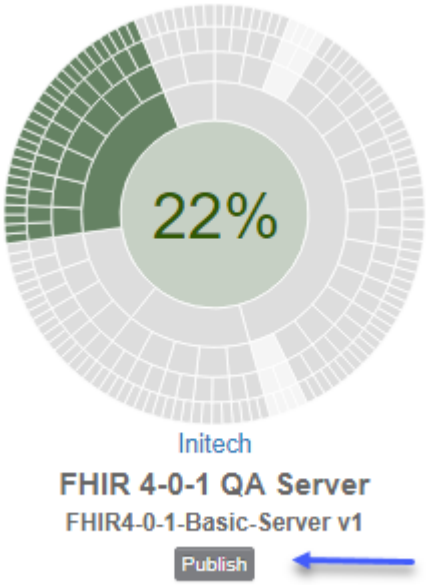
- Filters: Supported – If checked, users will be offered the **Supported by CapabStmt** checkbox on conformance results screens. They would be able to filter results by supported interactions in the capability statement in addition to the default (All):



- Filters: Formats – If checked, users will be offered the Format dropdown on conformance results screens. They would be able to filter results by individual formats (JSON / XML) in addition to the default (All):

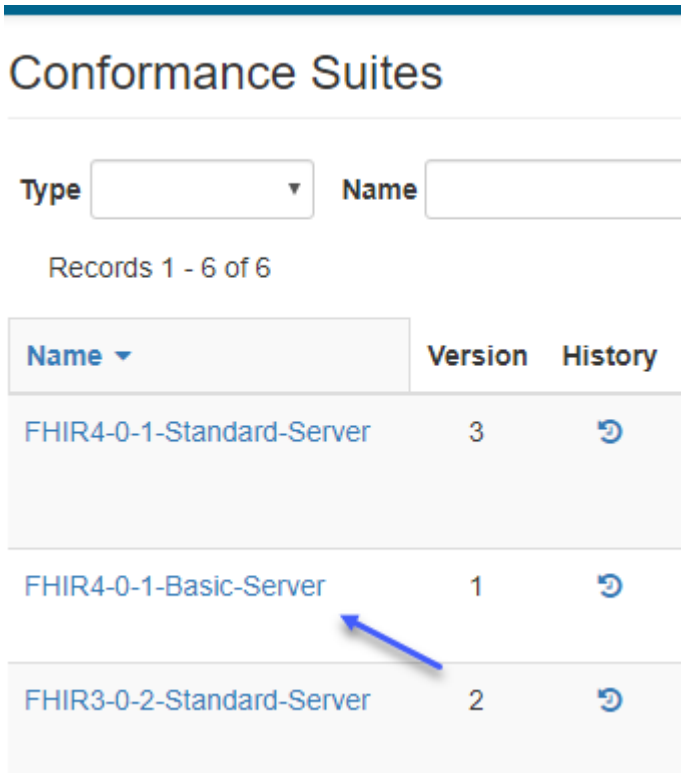


- Publishable – If checked, users will be offered the option of publishing conformance results. Those results along with the associated test executions would become publicly accessible if published.



7.1.2 Launching Executions

To select a given Conformance Suite (for execution), you can click on the name of the suite on the Conformance Suites page:



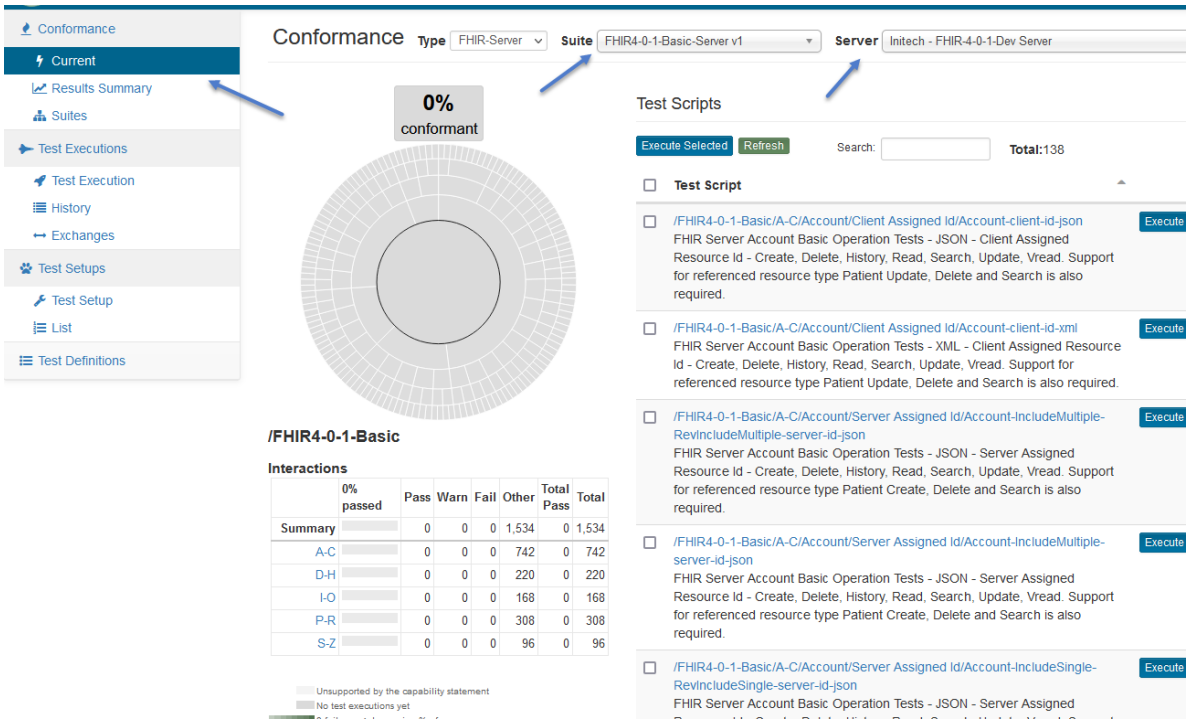
That takes you to the Current (Conformance) screen which is covered in the next section.

7.2 Current

7.2.1 Single Test Group

The Current (Conformance) page represents the latest Conformance metrics for the selected Conformance Suite and Test System.

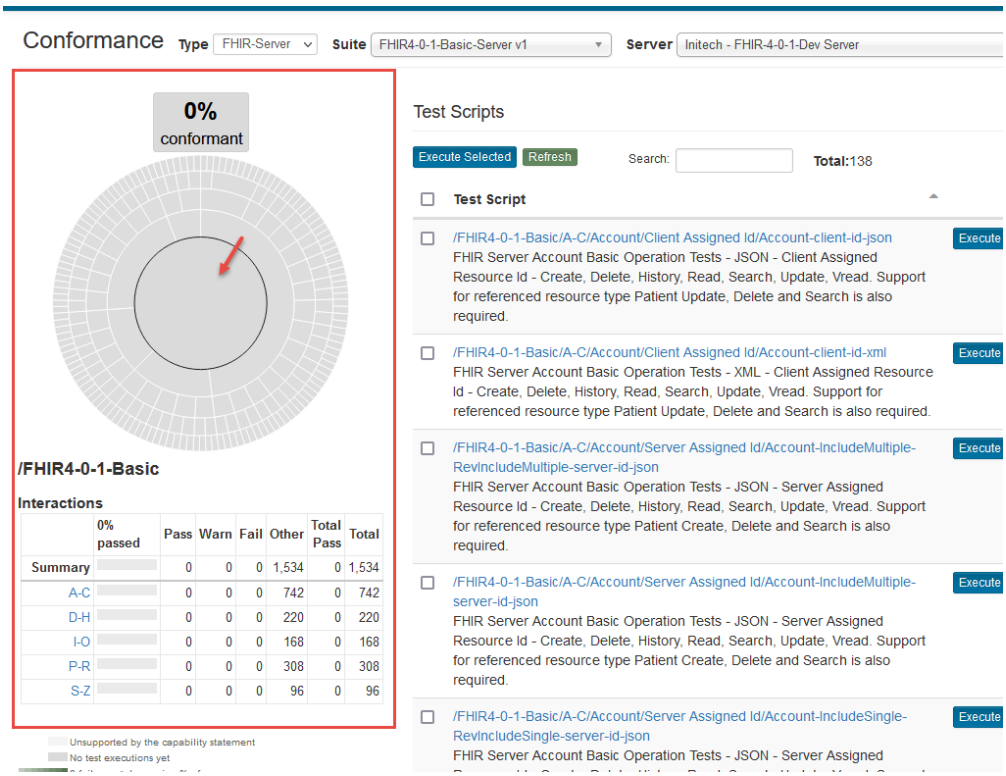
1. You can get to the Conformance page by clicking on the Conformance / Current link on the left menu. The Conformance Suite below contains a single test group. Initially, assuming no test executions, you should see all grey on the Results Summary chart:



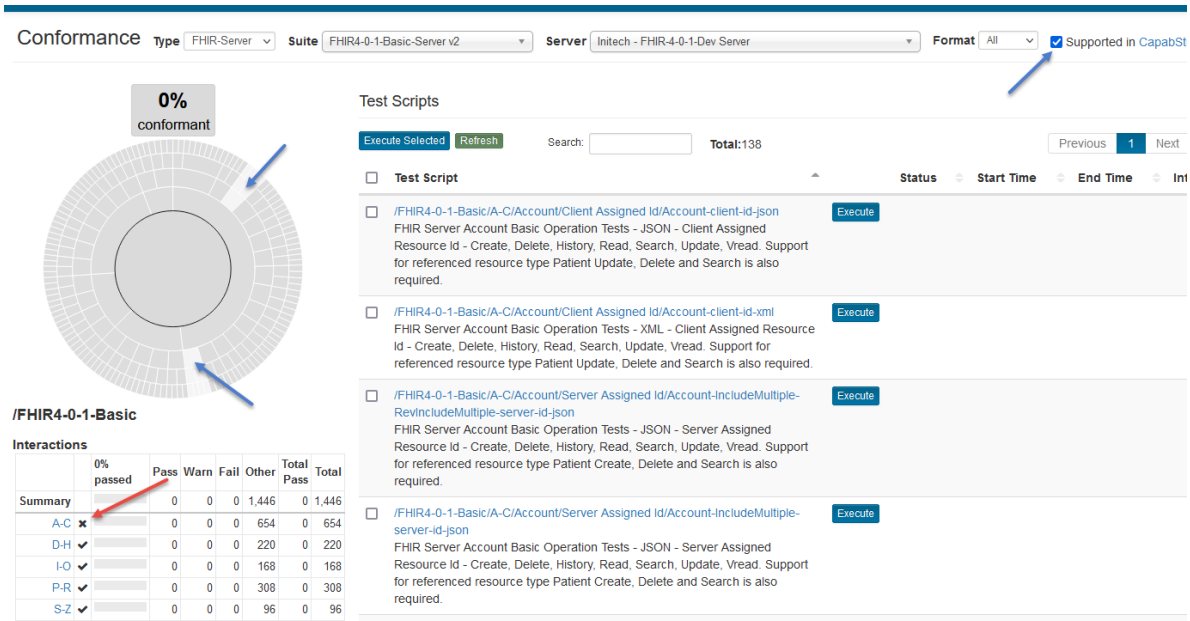
The dropdowns for Suite and Server are filtered to those that you have access to.

Note: When a Conformance Suite contains a single test group, the test group chart and the Results Summary chart (red rectangle below) are the same.

The center-most band (red arrow) that's selected by default is the root of the Results Summary chart (red rectangle below). That band represents the root of the **/FHIR 4.0.1-Basic** single Test Group that composes this Conformance Suite.



2. If you'd like to focus only on interactions that your capability statement supports, click on the **Supported Only** checkbox:



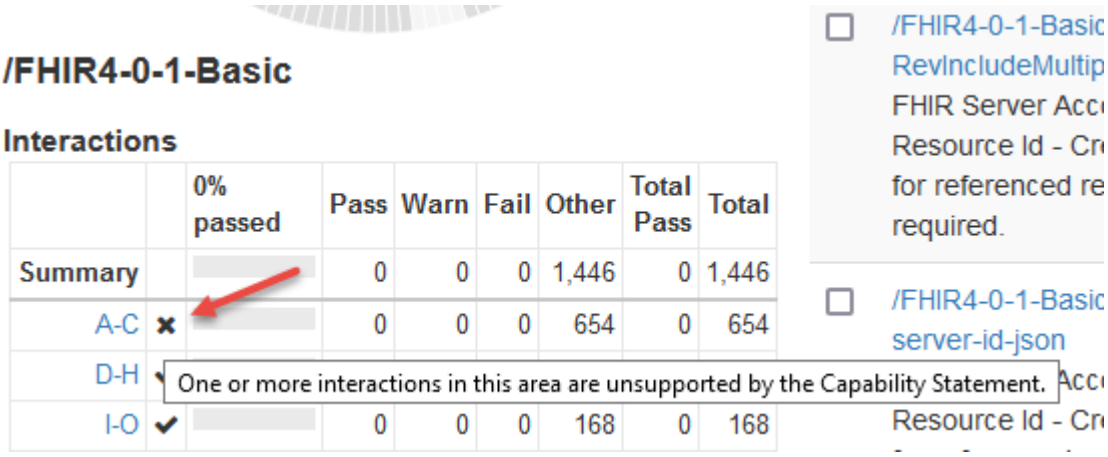
Notice that some bands have changed to silver (blue arrows above). Those are the areas that are **not** supported by this test system. Only the interactions that are supported by the Capabilities statement are contributing towards

'**% Conformance**'.

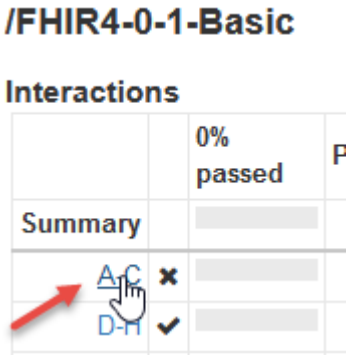
Note that the Supported checkbox may not be available for all Conformance Suites. The Suite owner can choose to make the checkbox unavailable for a suite in which case all interactions will contribute towards '**% Conformance**' regardless of support in Capability Statement.

Unsupported Interactions

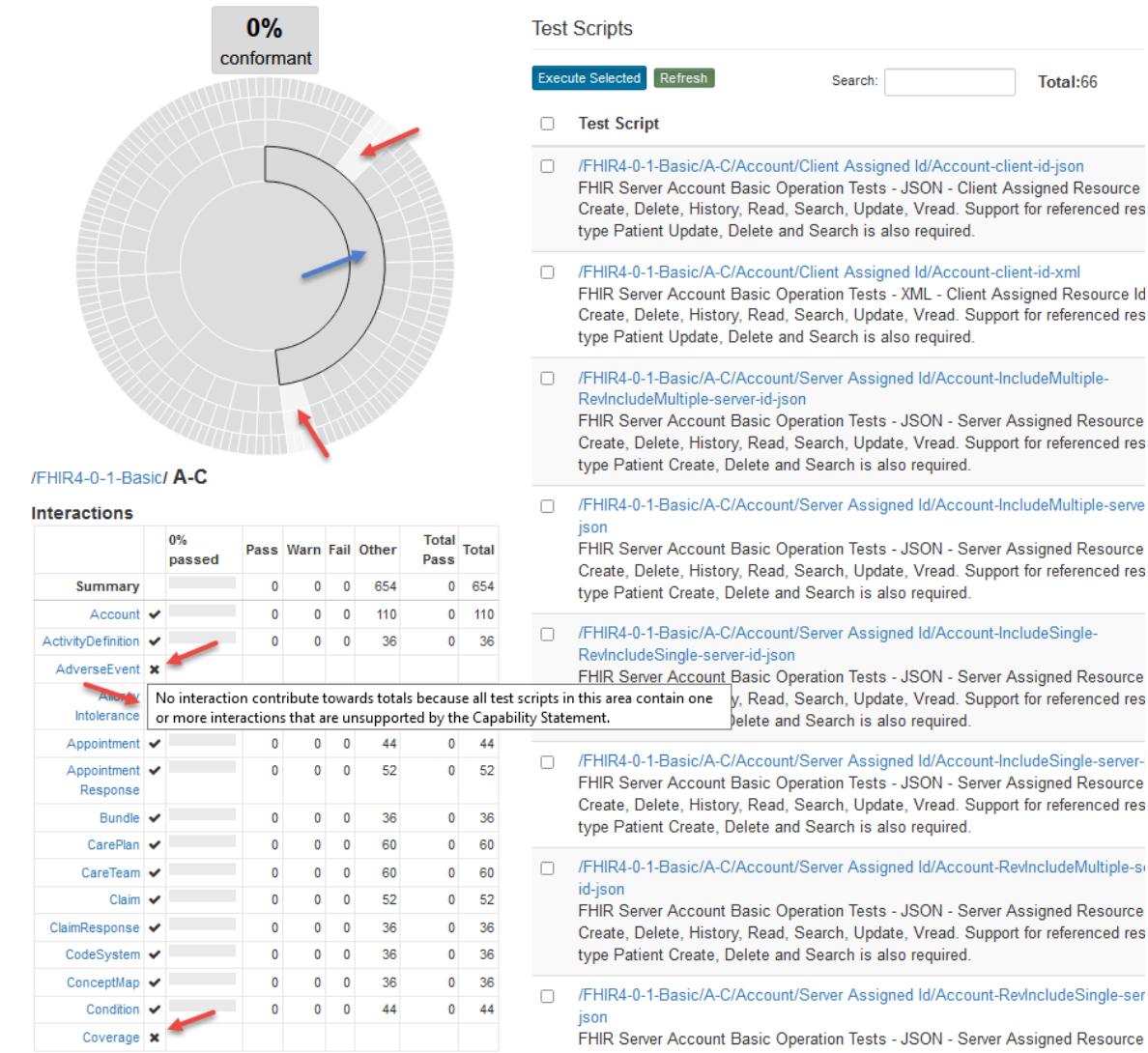
Notice that the A-C group is not supported yet by the Capability Statement:



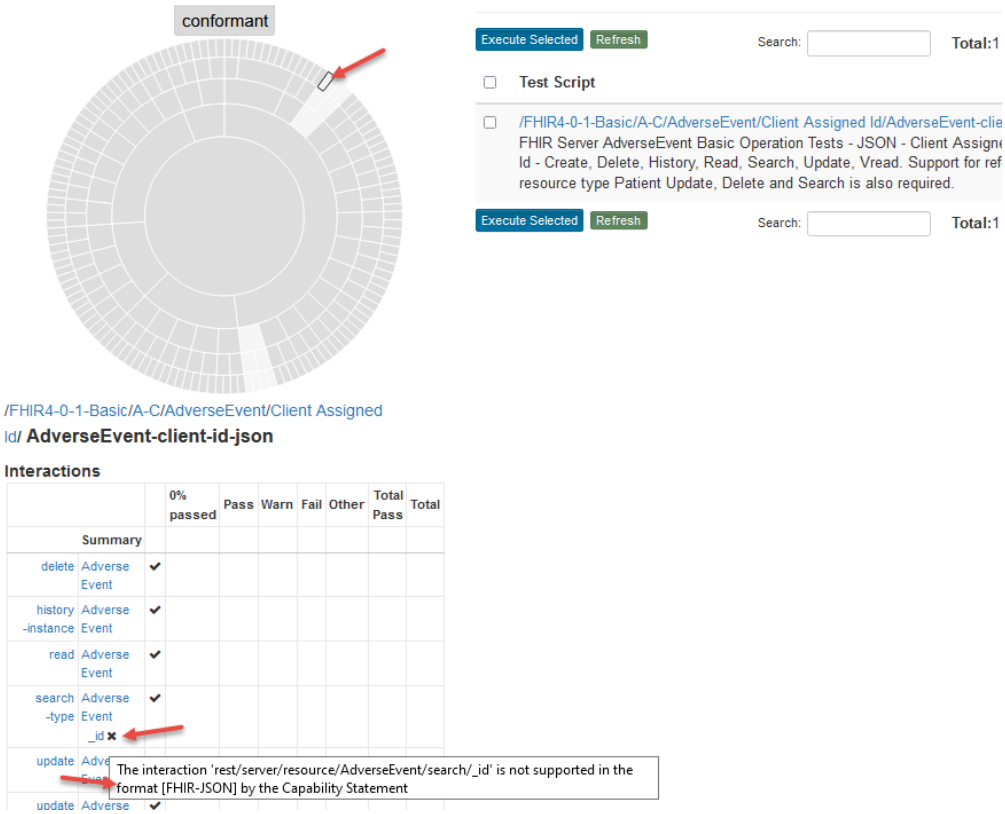
You can drill down by clicking on the **A-C** link within the table:



The Results Summary chart changes focus to a deeper level (blue arrow below). The unsupported groups have an **X** next to them (red arrow below):

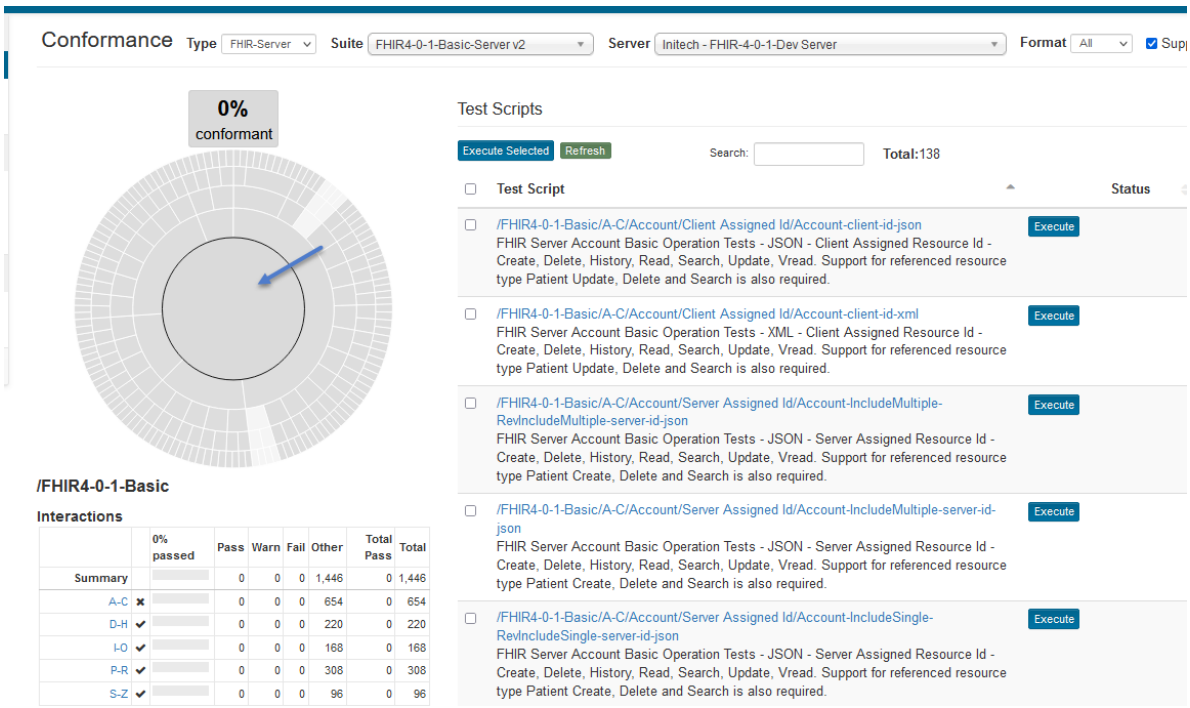


You can continue drilling down by clicking on the link within the table with an **X** until you get to the leaf node:

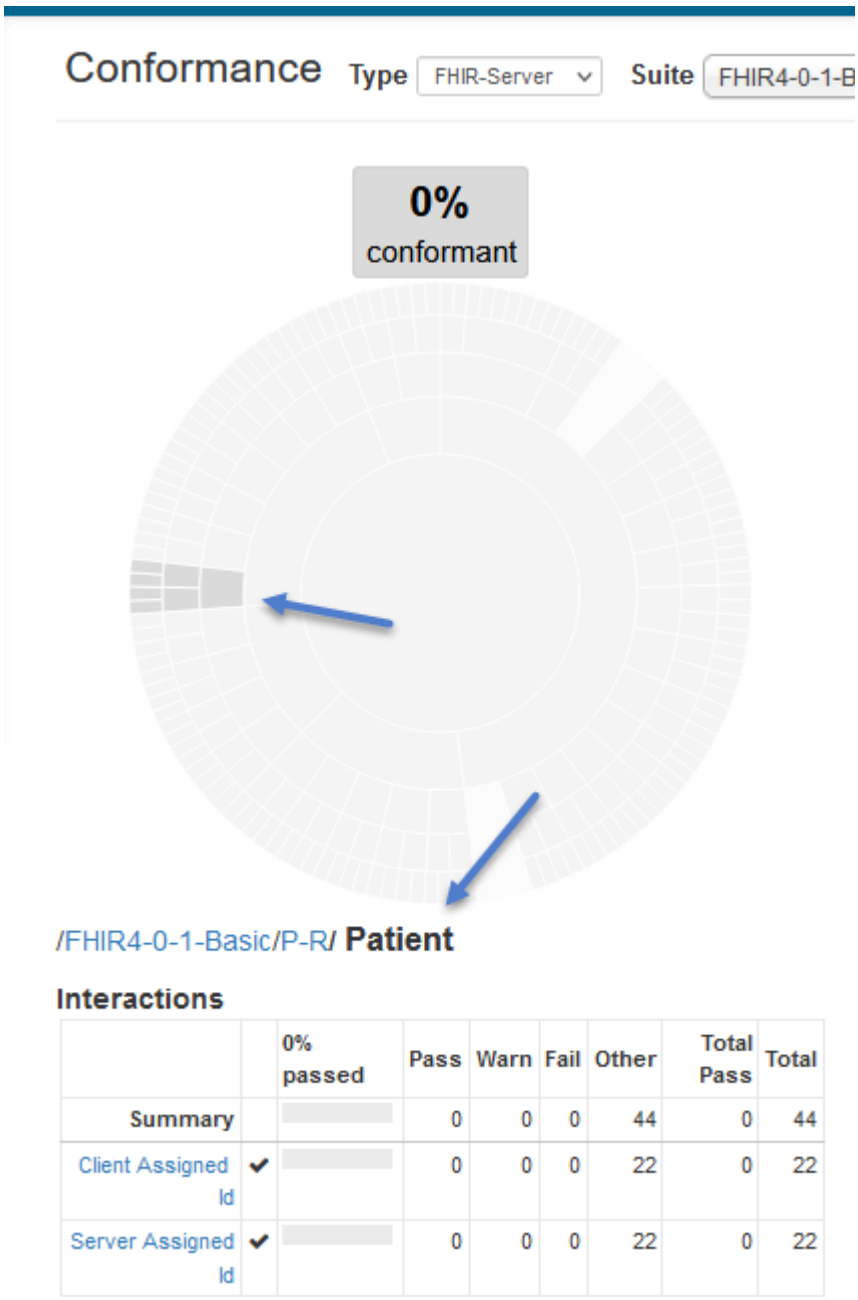


You can see the explanation (above) for why the test script along with its parent nodes were excluded from contributing towards the Results Summary '**% Conformance**'.

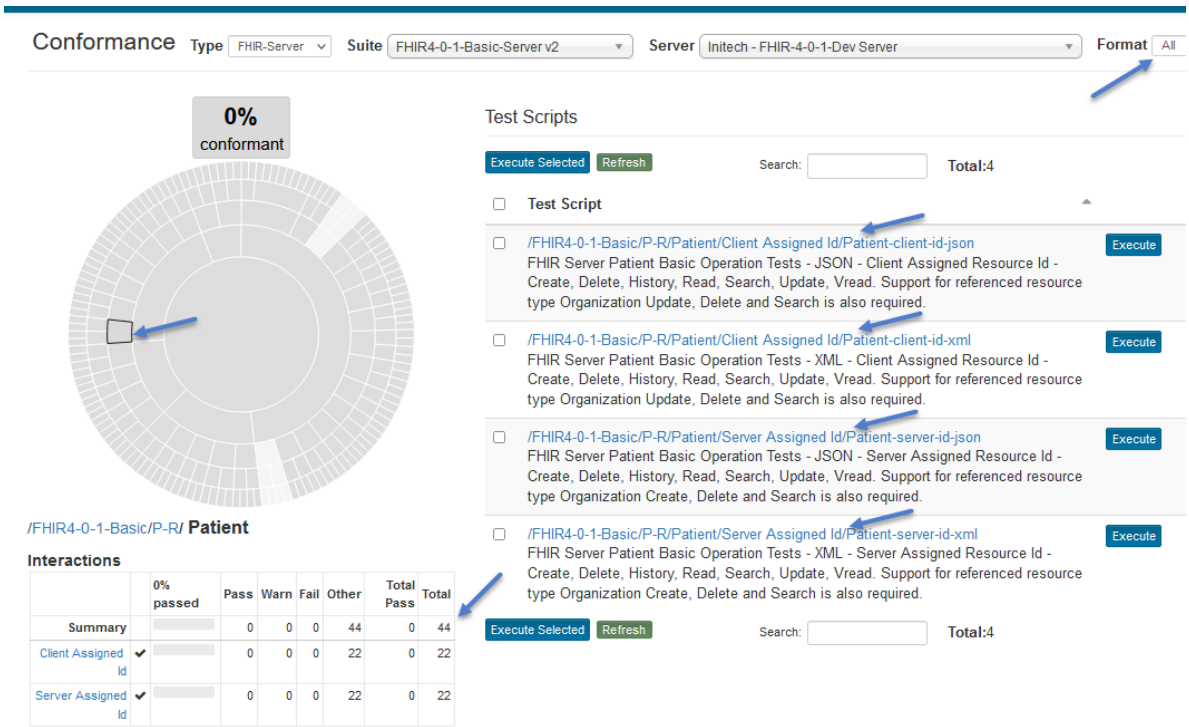
3. Click on the root node of the Results Summary chart so the center-most band is highlighted:



4. Suppose you're interested in executing tests for the Patient resource. You can hover over the Results Summary chart to find the Patient resource:

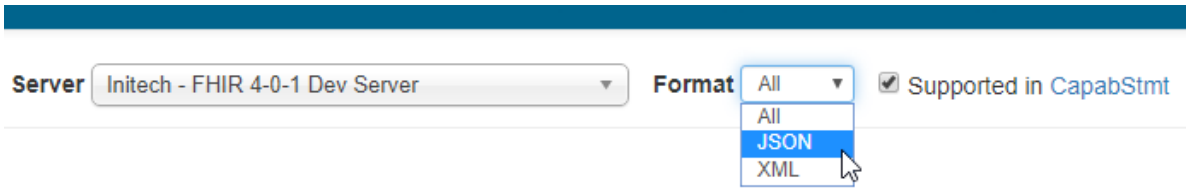


5. Click on the /FHIR4-0-1-Basic/P-R/**Patient** band (below arrow below) on the Results Summary chart:

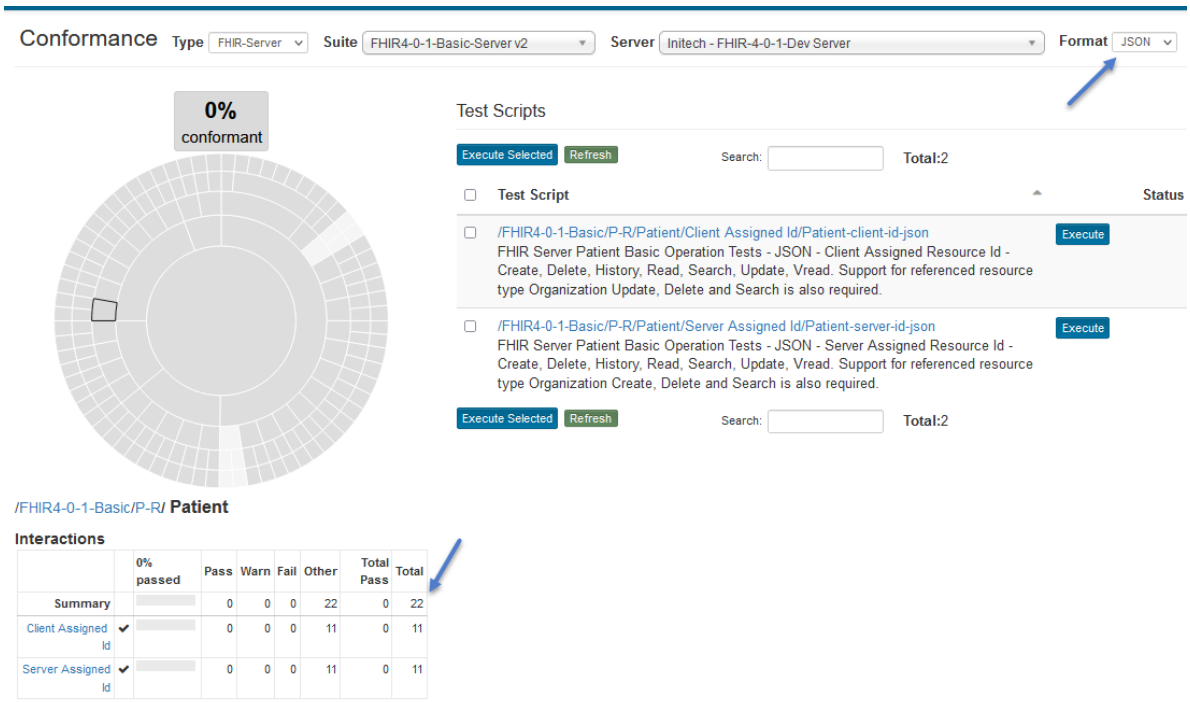


You can see there are 4 test scripts (on the right) with a total of 44 Patient interactions (below arrow above).

6. This particular suite offers filtering by formats (All, XML, JSON). Perhaps you're interested in testing out the JSON format first. You can select **JSON** in the Formats dropdown:

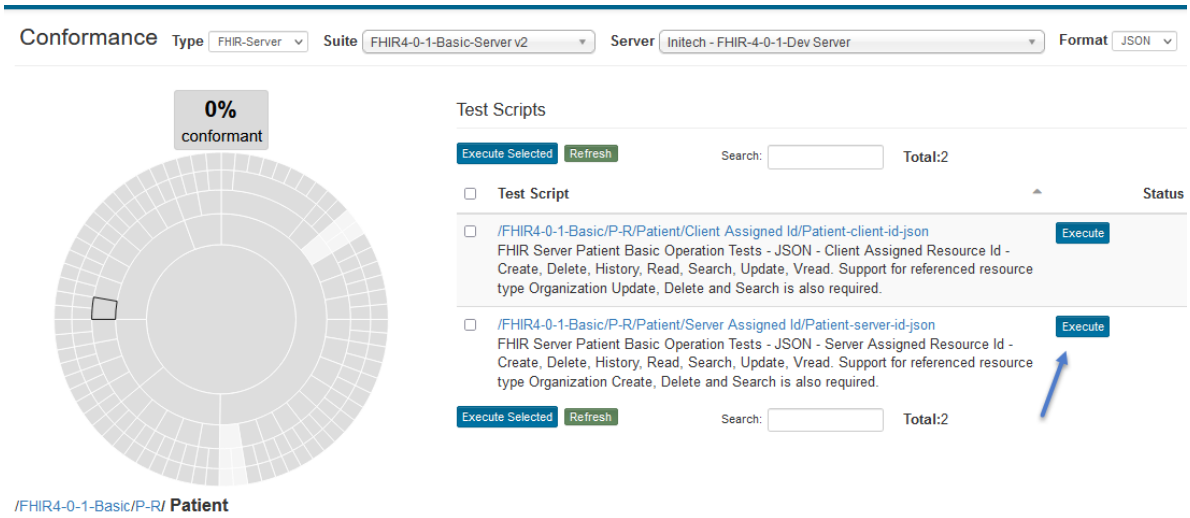


The test scripts and counts change to reflect the JSON interactions available in the suite:



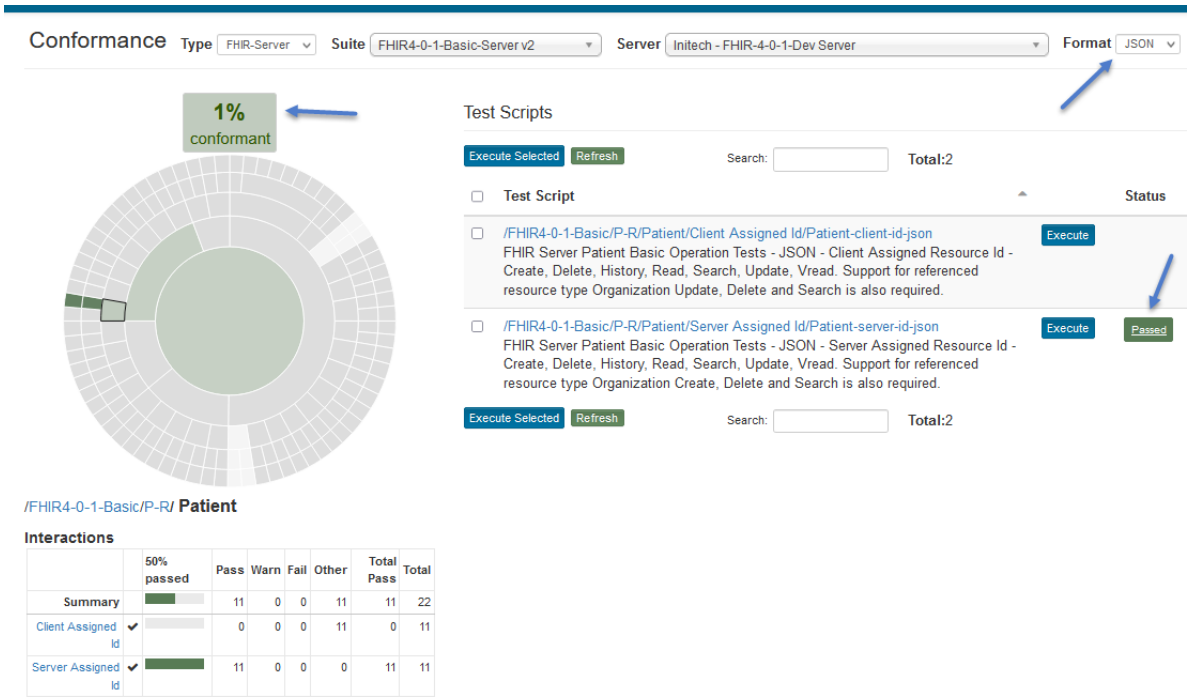
There are a total of 22 **JSON** Patient interactions (below arrow above) available in 2 test scripts in this suite.

6. Click on the **Execute** button if you want to launch one test script:

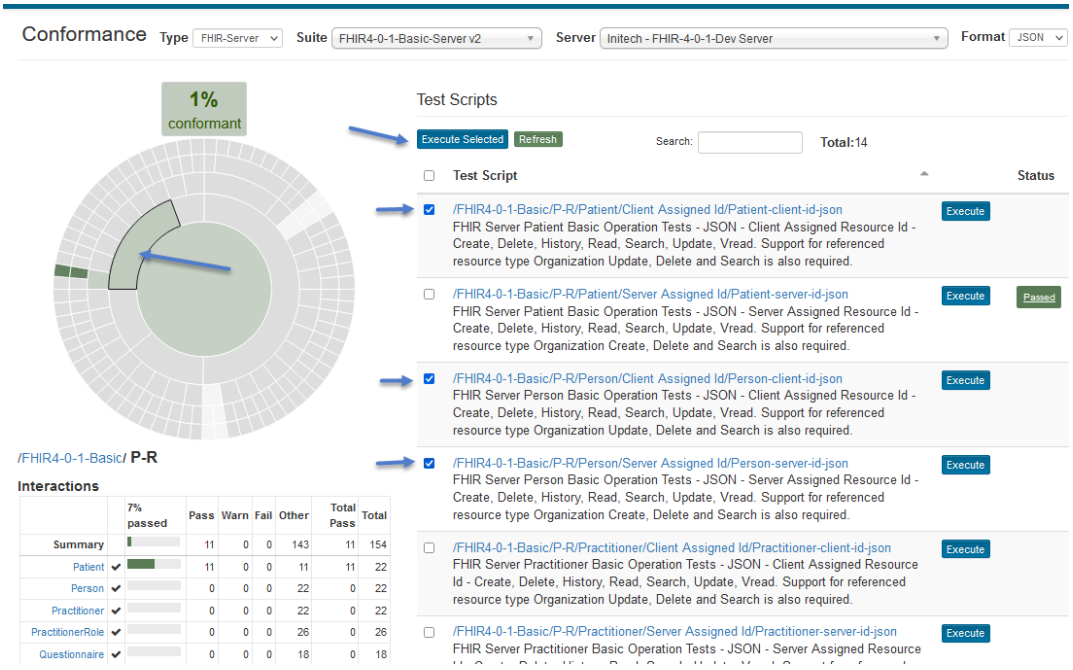


7. The test will get launched in the background. You might need to wait for a few seconds before refreshing the screen to see the status change to **Running** and then to either **Pass** or **Fail**:

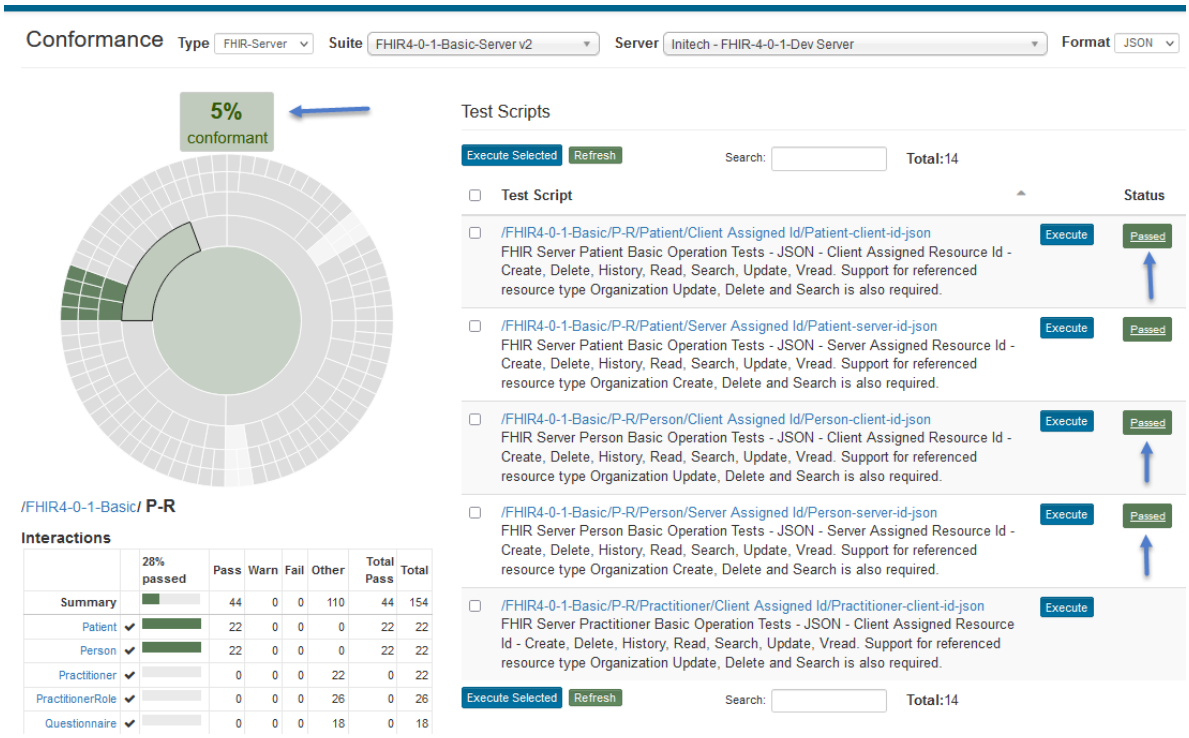
Notice that the interaction counts in the summary tables have changed and so have the percentages. The overall "% conformant" is now at 1%.



8. Select the parent band in the Results Summary chart and a bunch of test scripts and click on **Execute Selected**:



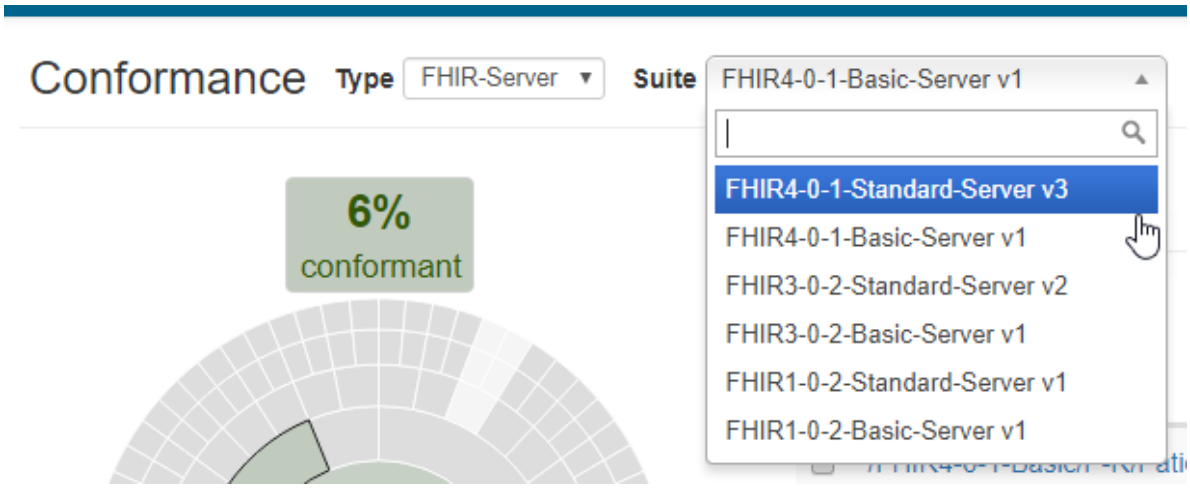
Notice that '**% Conformance**' has gone up to 4%. The "% conformant" does take into account all the interactions that have not been executed yet. It's at 4% because we have only executed 4 test scripts.



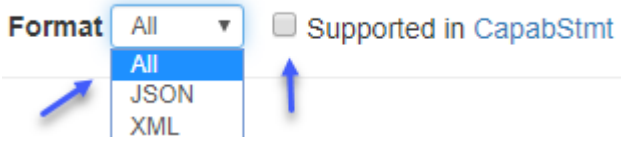
7.2.2 Multi Test Group

The Current (Conformance) page represents the latest Conformance metrics for the selected Conformance Suite and Test System.

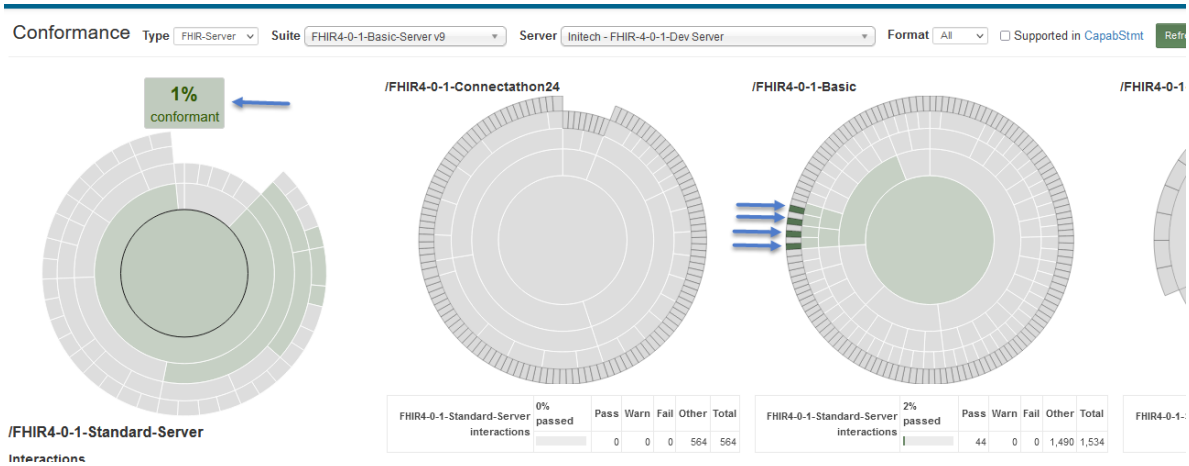
1. Change the Conformance Suite to one that's composed of more than 1 test group:



Also change the **Format** to All and **Supported** to unchecked :



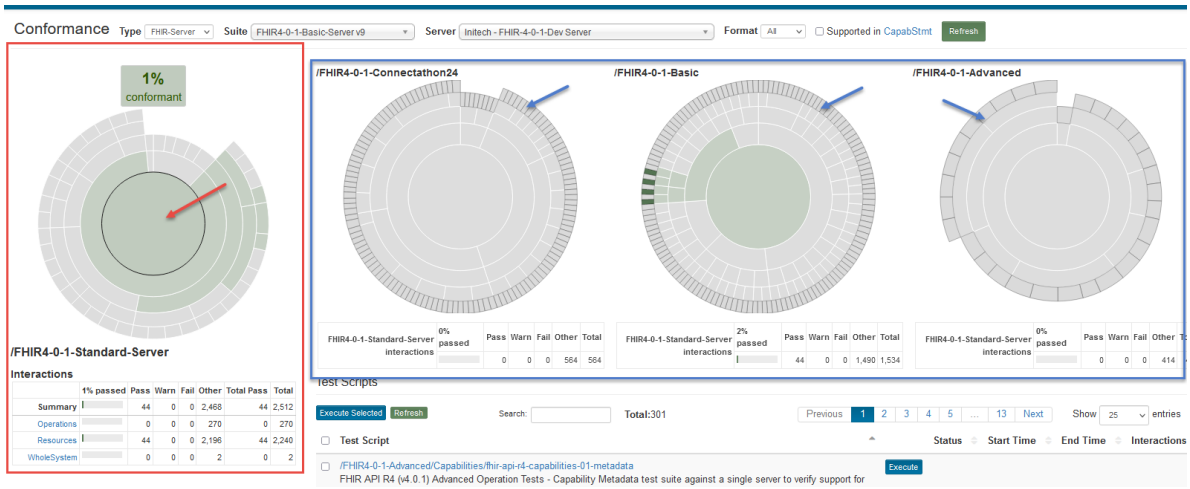
2. Notice that even though, this is the first time you've used this Suite, the '**% Conformance**' is already at 1%:



This is because the **/FHIR4-0-1-Basic** test group is also part of **FHIR4-0-1-Basic-Server** conformance suite and we had executed the 4 test scripts in green above earlier against the same test system. See *this FAQ* for more details.

The Conformance Suite contains 3 Test Groups (blue rectangle below).

The center-most band (red arrow) that's selected by default is the root of the Categorization chart (red rectangle below). It's **FHIR 4.0.1-Standard-Server** in this case. When a Conformance Suite has more than one Test Group, the interactions are grouped and summarized into the Categorization chart.



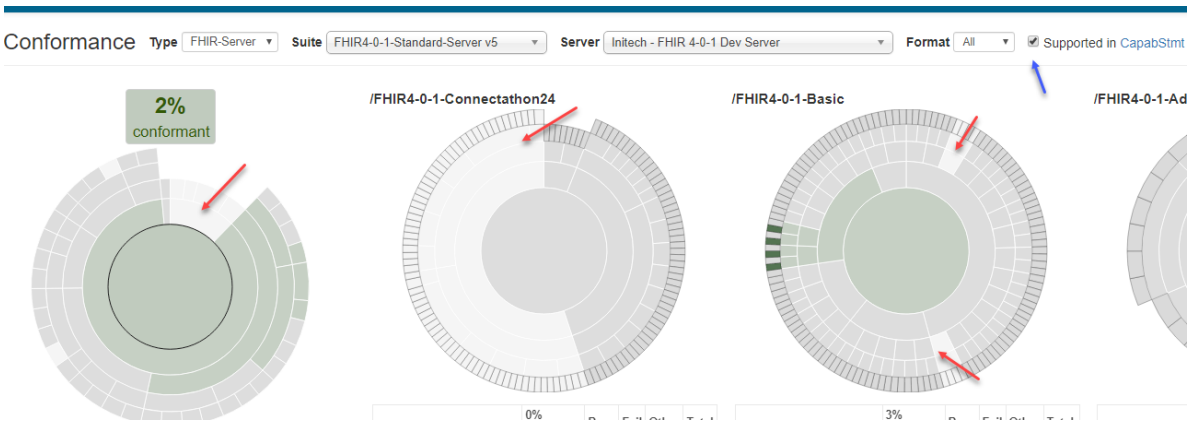
The charts on the right (blue rectangle) feed into the Categorization chart (red rectangle). The Categorization chart is driven by the categories in FHIR specification's Resource List and is customizable by the suite owner. The current band has a black border circle (red arrow above). The outermost bands of charts on the right are individual test scripts in Touchstone. Those that contain interactions for the current band (FHIR 4.0.1 in this case) will get a dark-grey border. In this case, because we're on the root band in the Categorization chart, all outermost bands on the right have a dark-grey border (blue arrow above).

The **Test Scripts** table below the charts will show all the test scripts for the currently selected band. Because we've selected the root band in the Categorization chart, all test scripts in **FHIR 4.0.1-Standard-Server** suite will be listed in this table.

Note: When a Conformance Suite contains multiple test groups, Categorization is mandatory for the

suite, and the **Categorization chart** and the **Results Summary chart** are the same.

3. If you'd like to focus only on interactions that your capability statement supports, click on the **Supported Only** checkbox:

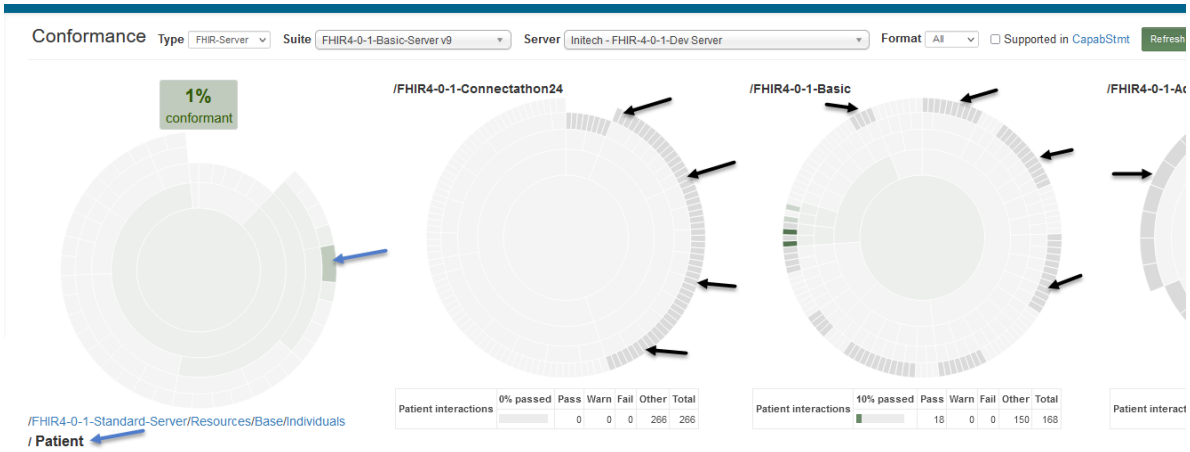


Notice that some bands have changed to silver (red arrows above). Those are the areas that are **not** supported by this test system. Only the interactions that are supported by the Capabilities statement are contributing towards '**% Conformance**'.

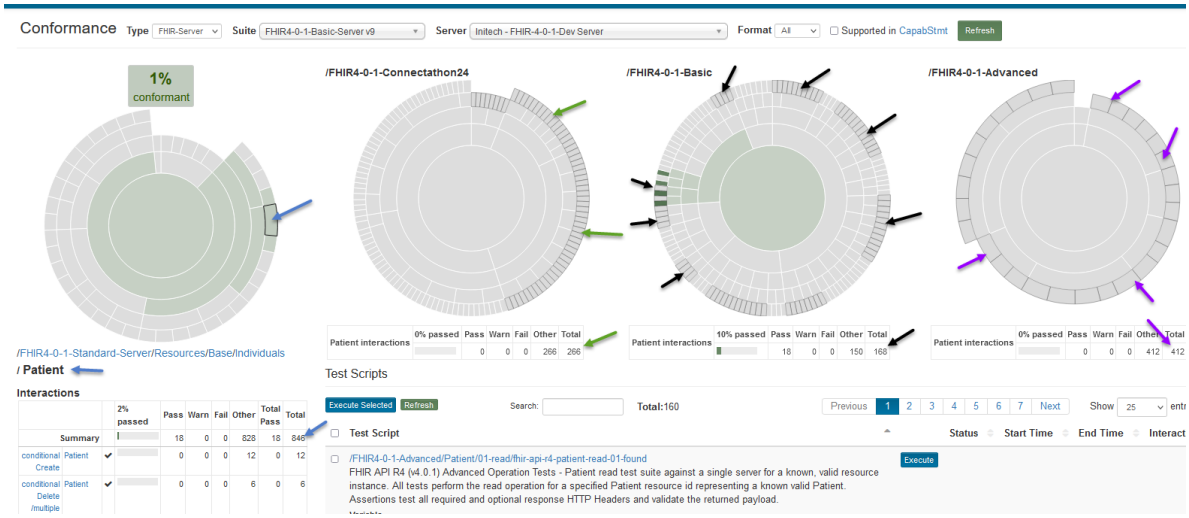
Note that the Supported checkbox may not be available for all Conformance Suites. The Suite owner can choose to make the checkbox **un**available for a suite in which case all interactions will contribute towards '**% Conformance**' regardless of support in Capability Statement.

4. Suppose you're interested in executing tests for the **Patient** resource. You can hover over the Results Summary chart to find the Patient resource (blue arrow below):

As you hover over each band, the test scripts that contain tests for that area are highlighted in dark grey on the right (black arrows below).



5. Click on the /FHIR4-0-1-Standard-Server/Resources/Base/Individuals/**Patient** band (below arrow below) on the Results Summary chart:

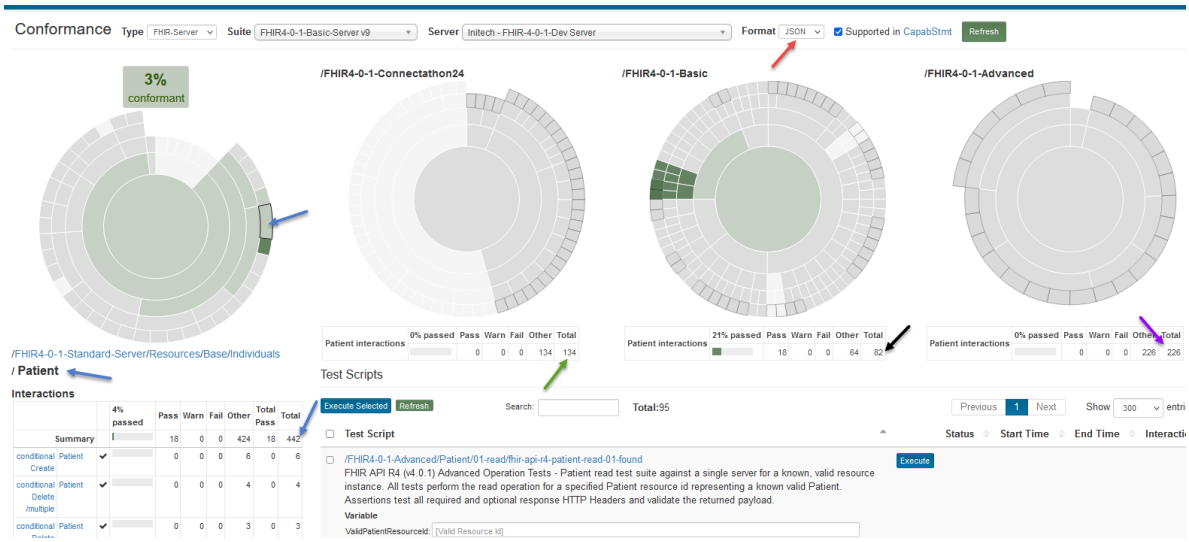


All the test scripts in charts on the right that have interactions for the Patient resource get a dark-grey border (green, black, and purple arrows). The counts of those interactions are displayed for the corresponding group underneath it.

There are a total of 846 Patient interactions available in 160 test scripts in all test groups in this suite. See blue arrows.

- In Connectathon24 test group, there are a total of 266 Patient interactions. See green arrows.
- In Basic test group, there are a total of 168 Patient interactions. See black arrows.
- In Advanced test group, there are a total of 412 Patient interactions. See purple arrows.

6. This particular suite offers filtering by formats (All, XML, JSON). Perhaps you're interested in testing out the JSON format first. You can select **JSON** in the Formats dropdown (red arrow).



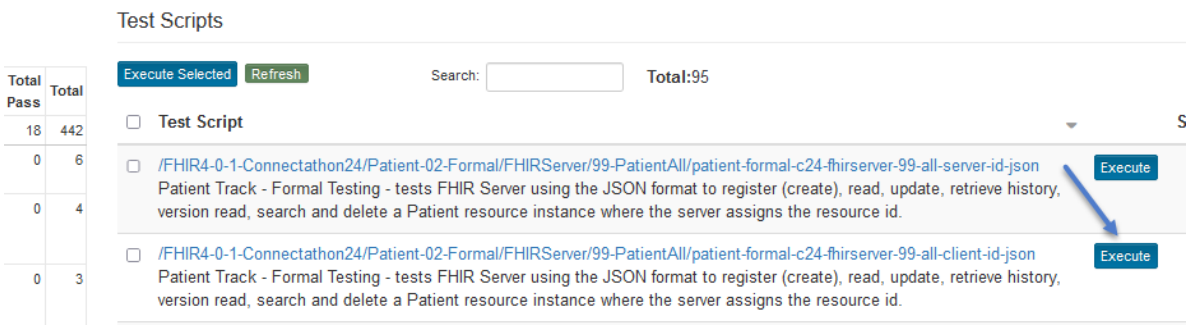
The test scripts and counts change to reflect the JSON interactions available in the suite.

There are a total of 442 **JSON** Patient interactions available in 95 test scripts in all test groups in this suite. See blue arrows.

- In Connectathon14 test group, there are a total of 134 **JSON** Patient interactions. See green arrows.
- In Basic test group, there are a total of 82 **JSON** Patient interactions. See black arrows.
- In Advanced test group, there are a total of 226 **JSON** Patient interactions. See purple arrows.

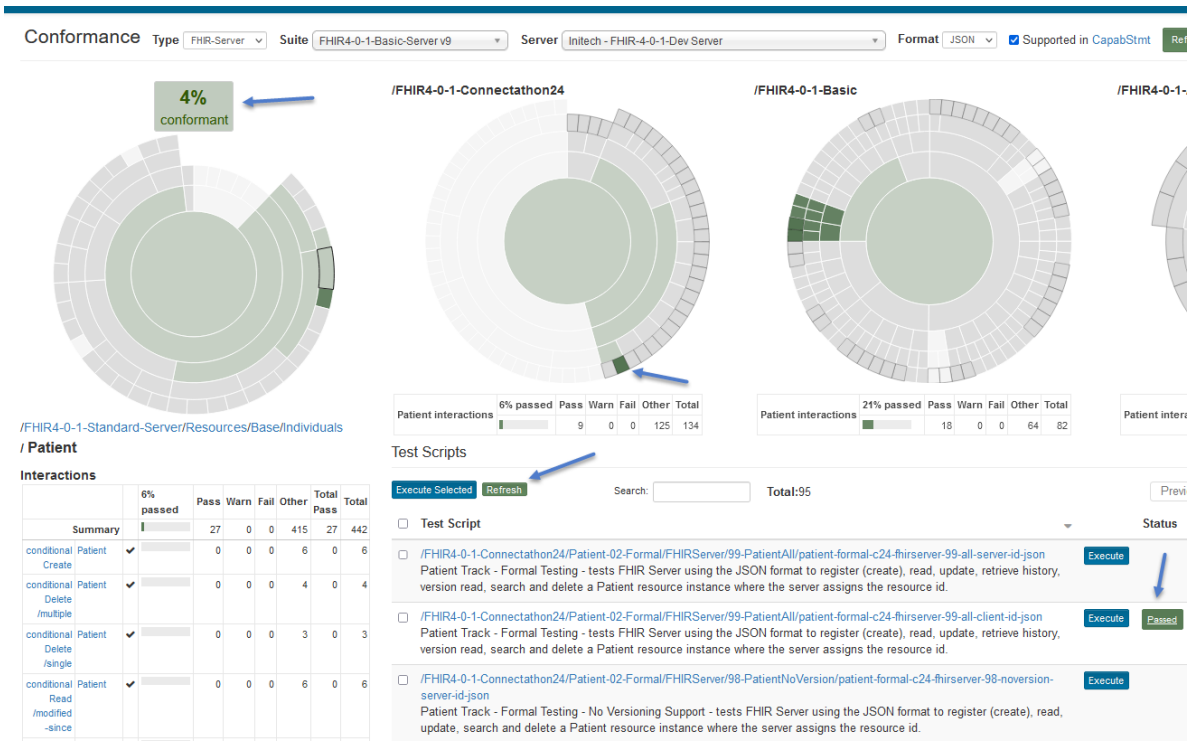
7. Flip the sort direction to Descending:

8. Click on the **Execute** button if you want to launch one test script:

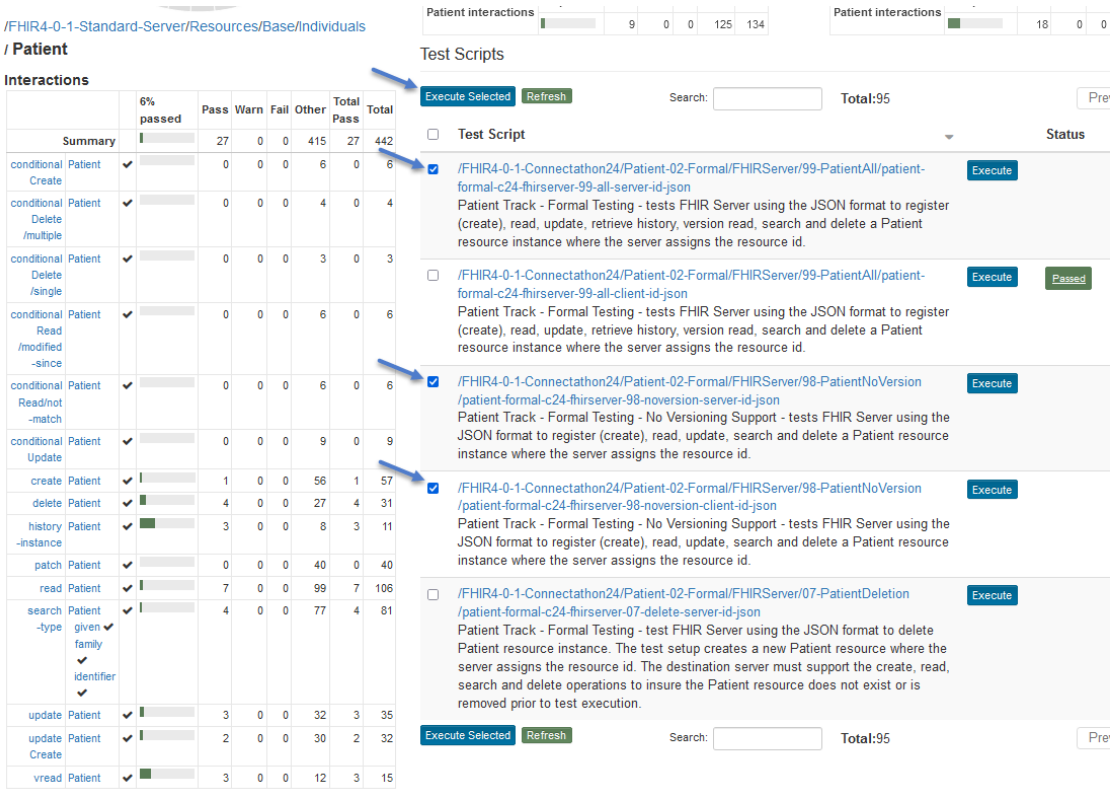


9. The test will get launched in the background. You might need to wait for a few seconds before refreshing the screen to see the status change to **Running** and then to either **Pass** or **Fail**:

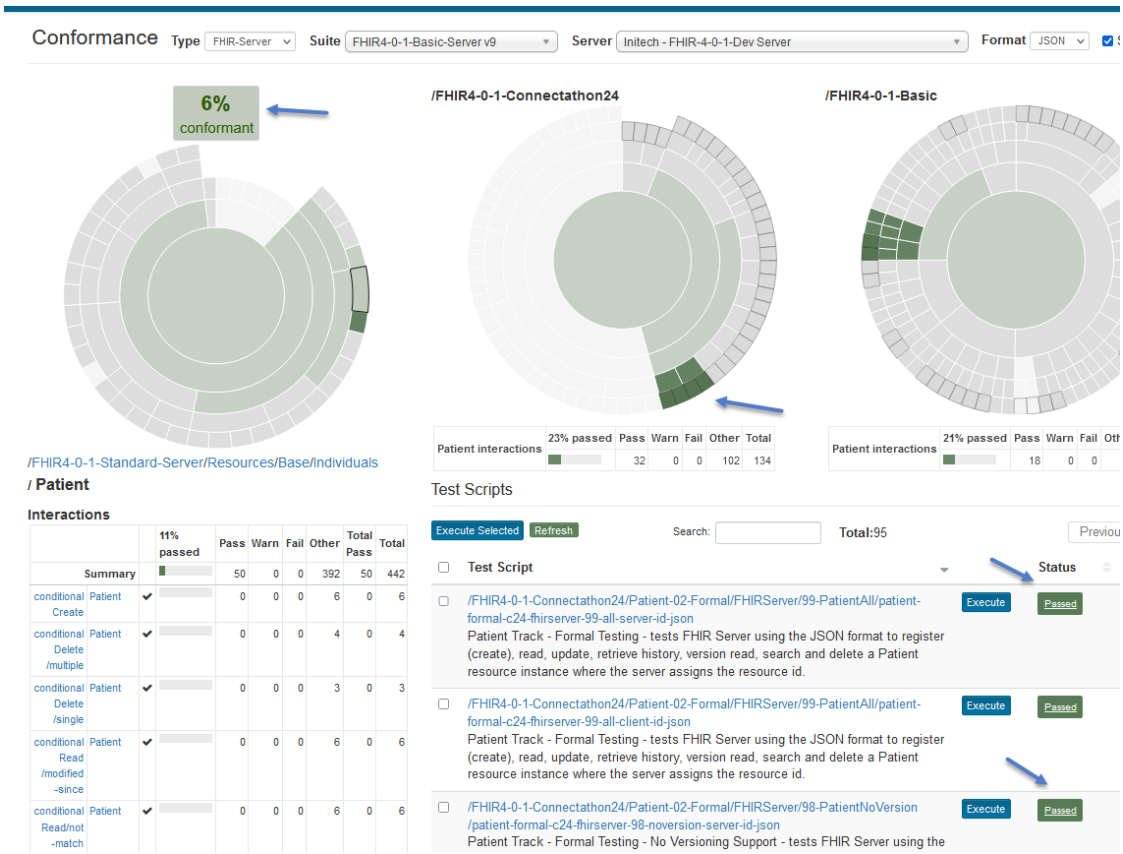
   Notice that the interaction counts in the summary tables have changed and so have the percentages. The overall "**% conformant**" is now at 4%.



10. Select a bunch of test scripts and click on **Execute Selected**:



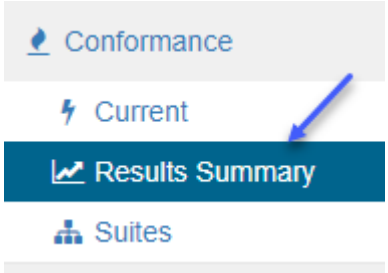
Notice that '**% Conformance**' has gone up to 6%. The "% conformant" does take into account all the interactions that have not been executed yet. It's at 6% because we have only executed a few test scripts.



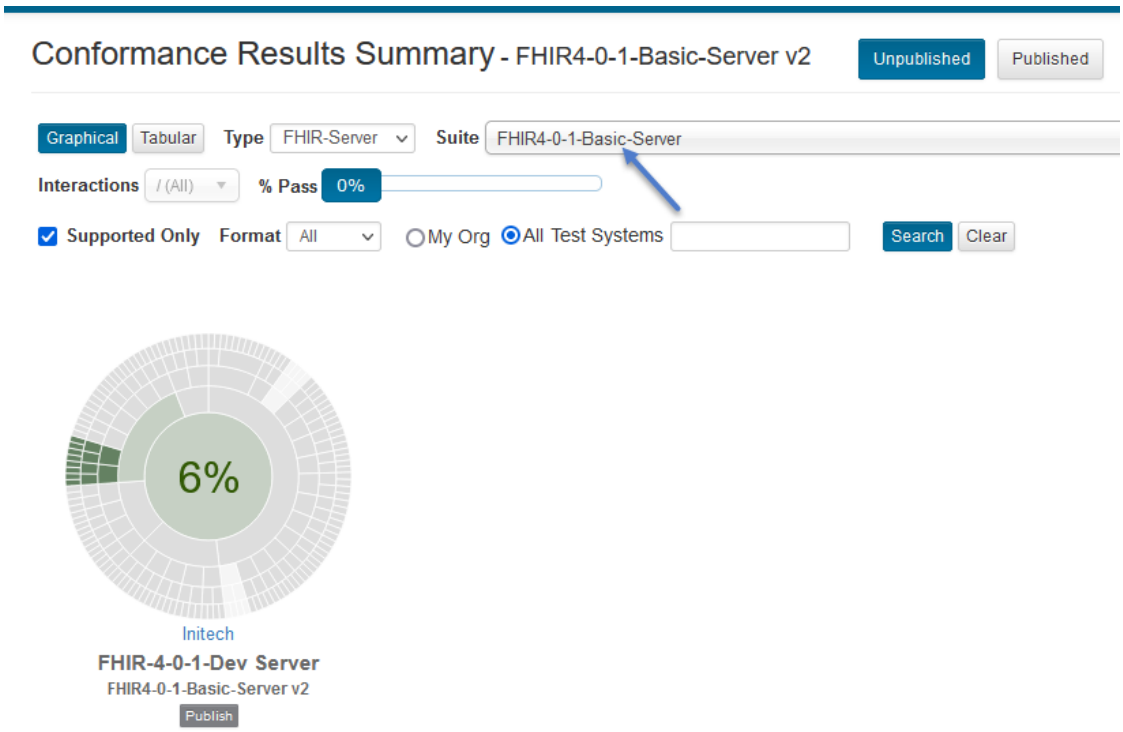
7.3 Results Summary

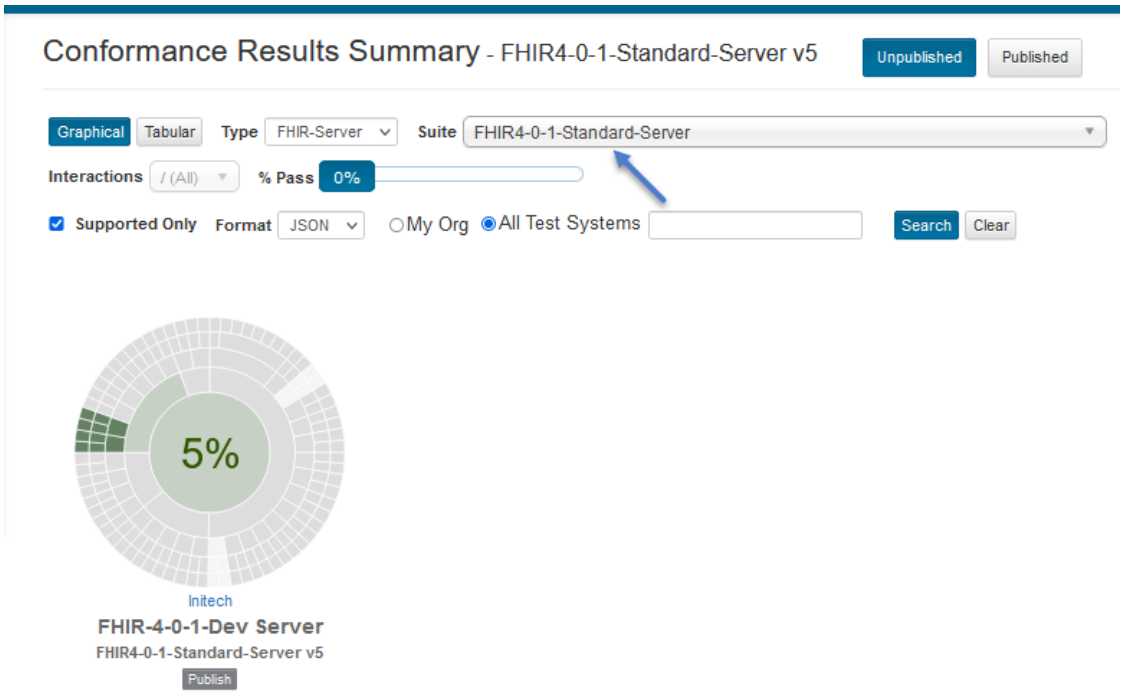
This Results Summary page gives you the Result Summaries (left chart on Current page) of different test systems for the selected Conformance Suite.

1. You can get to the **Results Summary** page by clicking on Conformance / **Results Summary** link on the left menu:

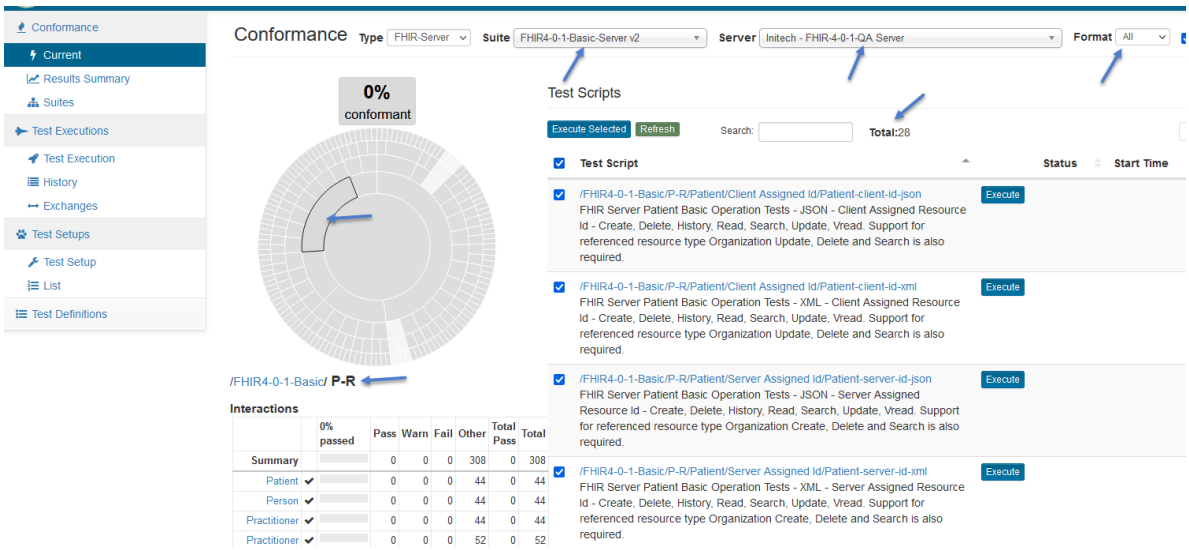


We see one result for **FHIR4-0-1-Basic-Server** Conformance suite and one for **FHIR4-0-1-Standard-Server** suite:



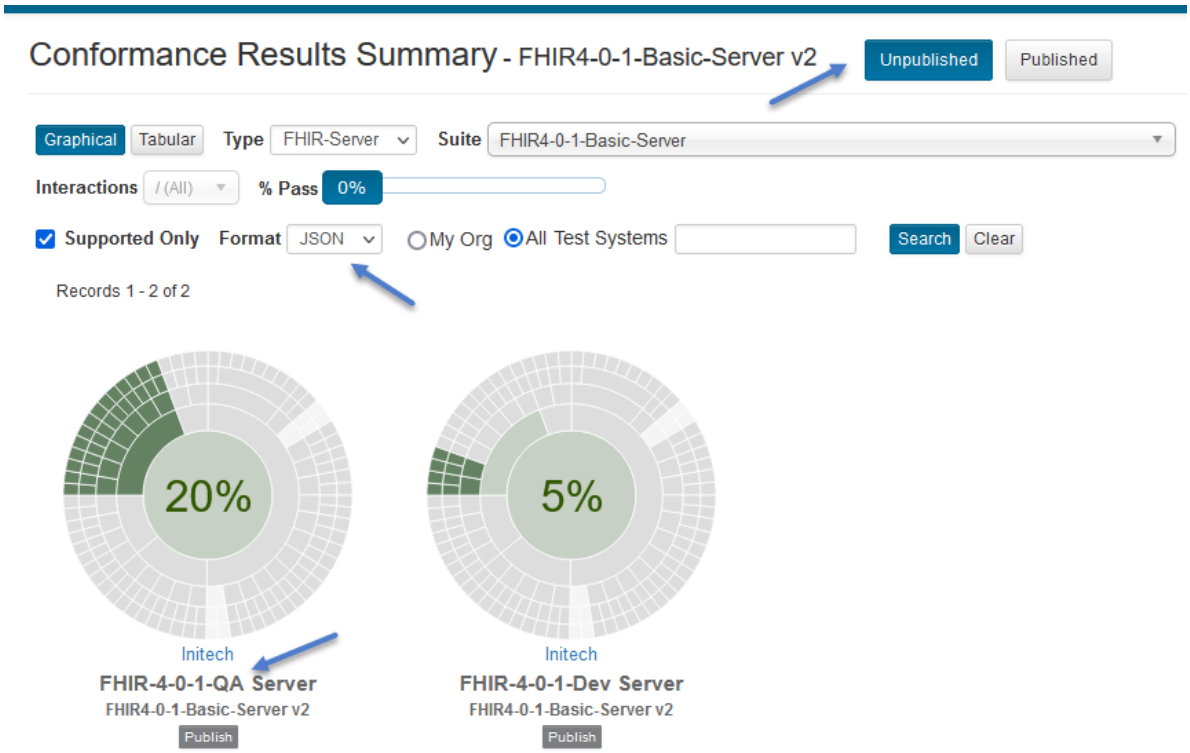


2. Execute some test scripts in **FHIR4-0-1-Basic-Server** Conformance suite on Current page against a different test system:



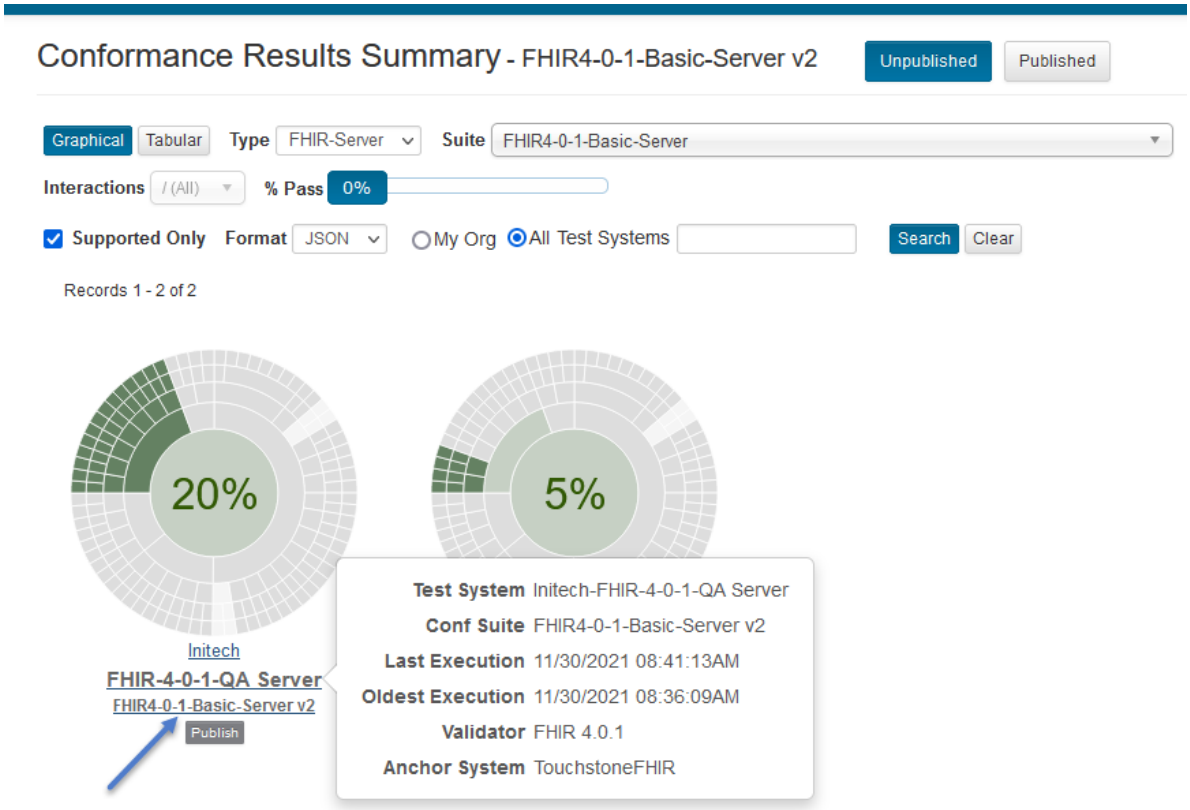
Notice that we're executing both **XML** and **JSON** test scripts against the **FHIR 4-0-1 QA Server** test system.

3. Go back to the Results Summary page and select **FHIR4-0-1-Basic-Server** Conformance suite. Notice the extra test system in the results when **JSON** format is selected:

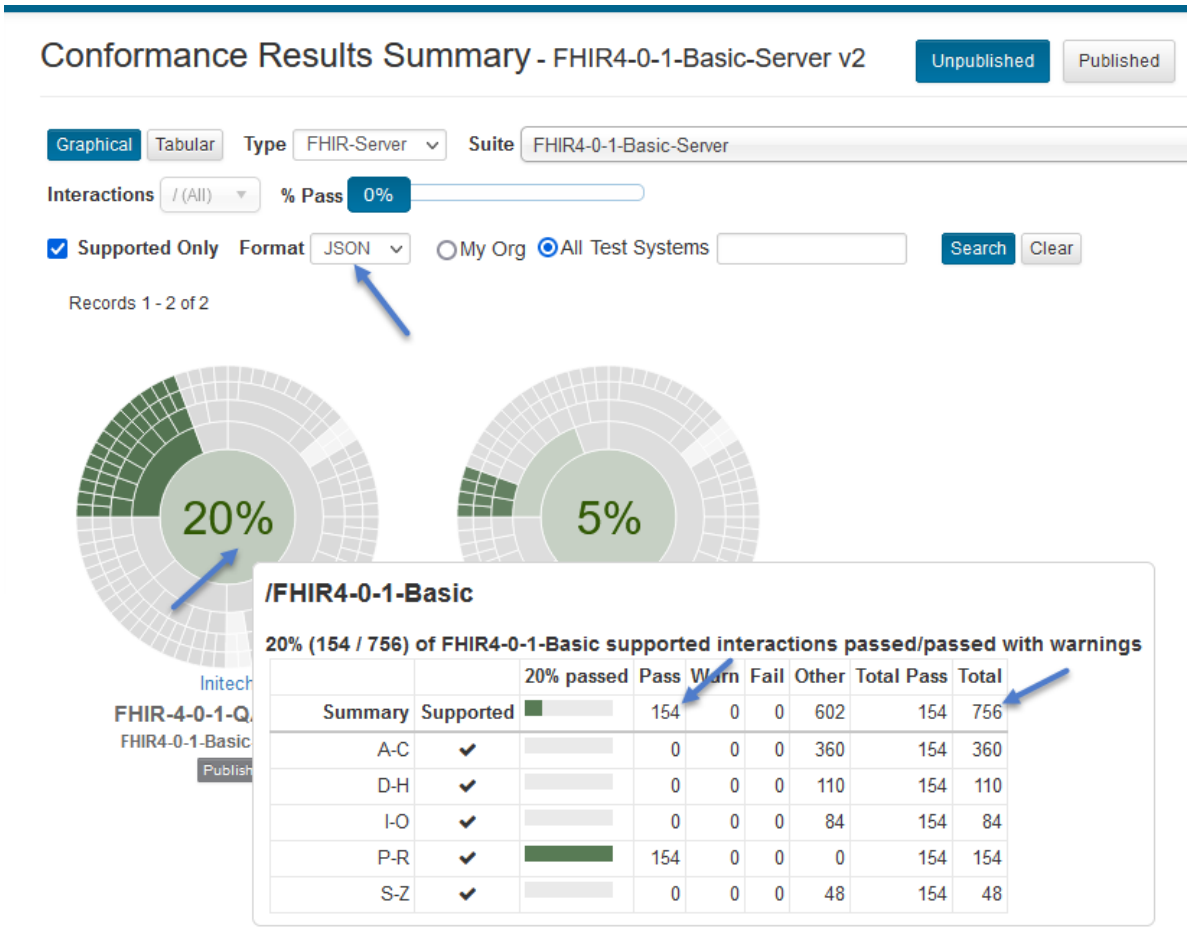


We had executed **JSON** test scripts against both test systems.

Hover over the text to get more information about the results:

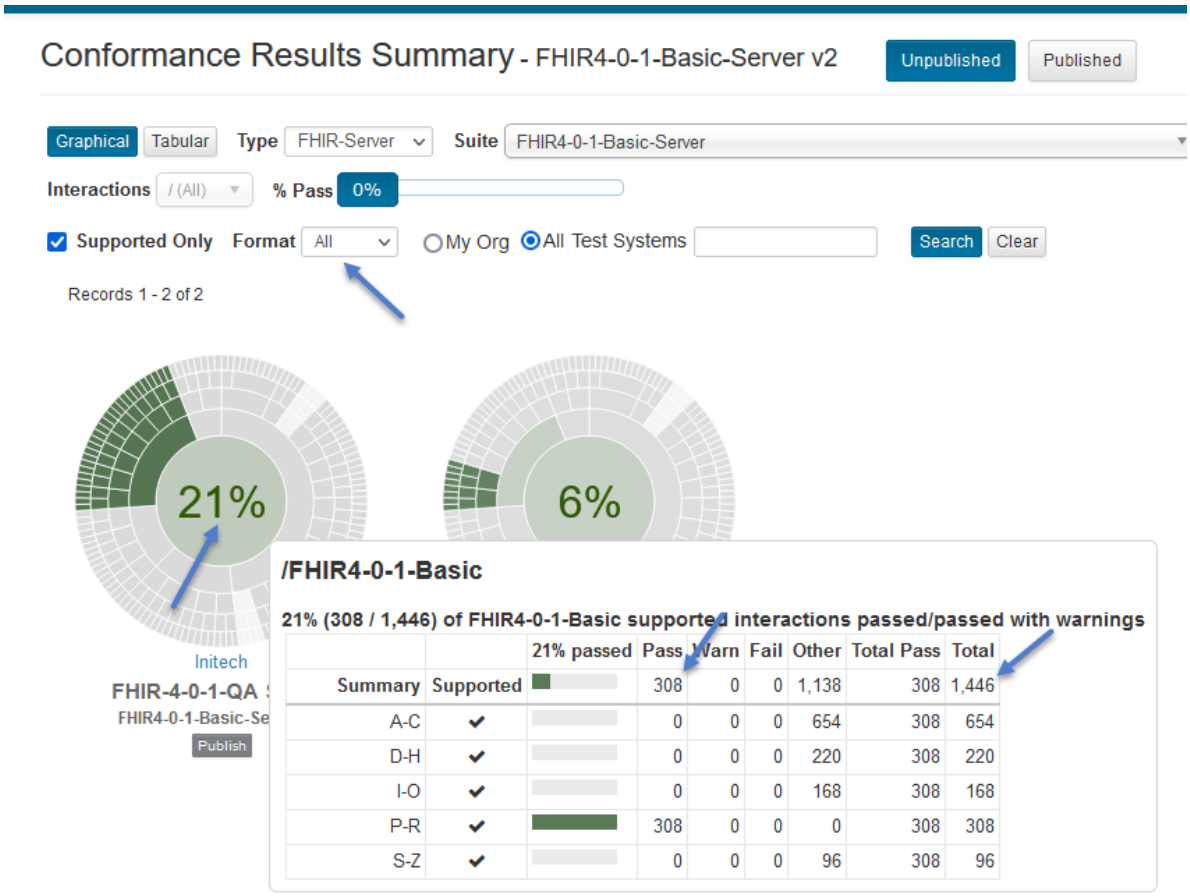


Hover over the central band to get the aggregated counts:

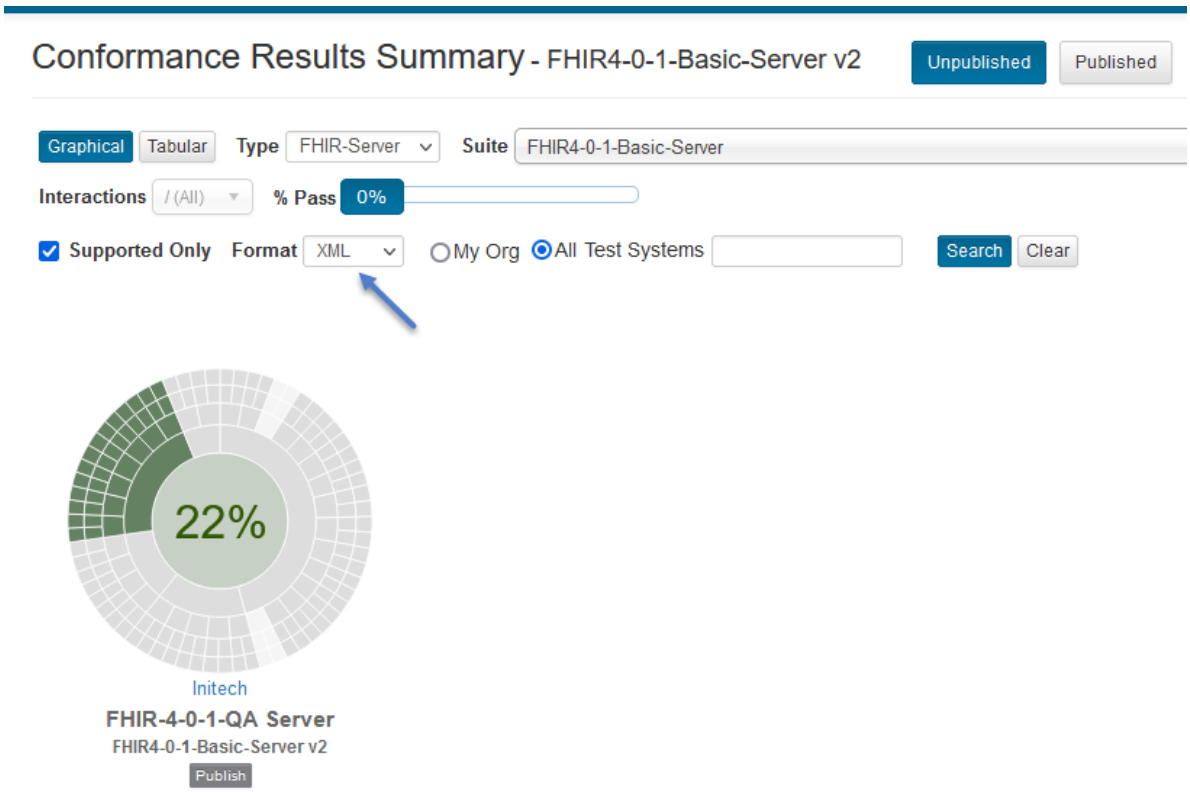


4. Changing the format selection to **All** will continue to show both test systems as **All** includes **JSON** and we executed **JSON** test scripts against both test systems:

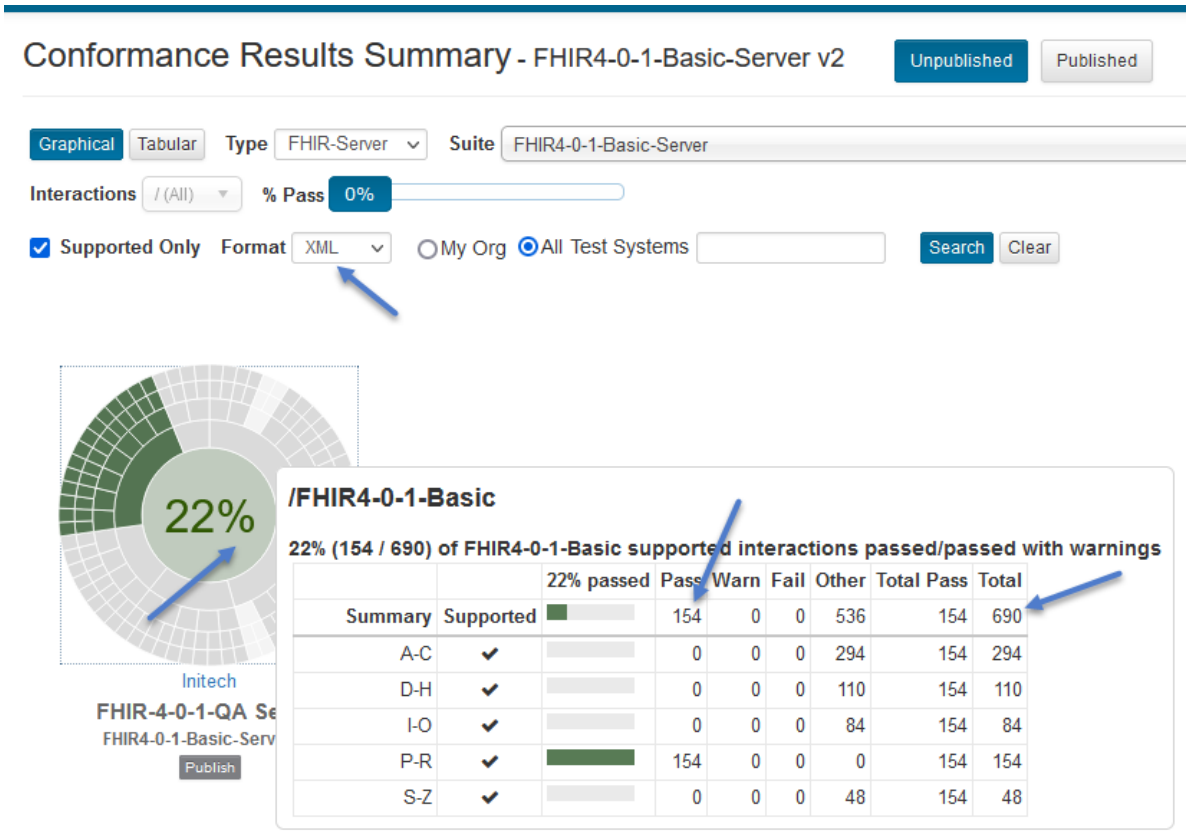
   However, the aggregated counts will be different when the format selection is **All**:



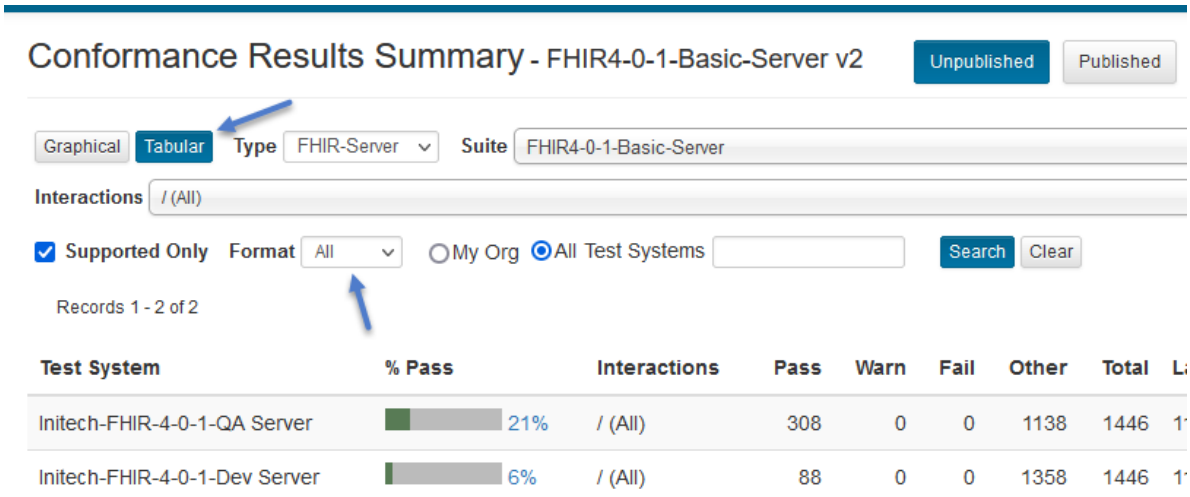
5. Change the format selection to **XML**. Notice that the listing includes only the server that has **XML** results:



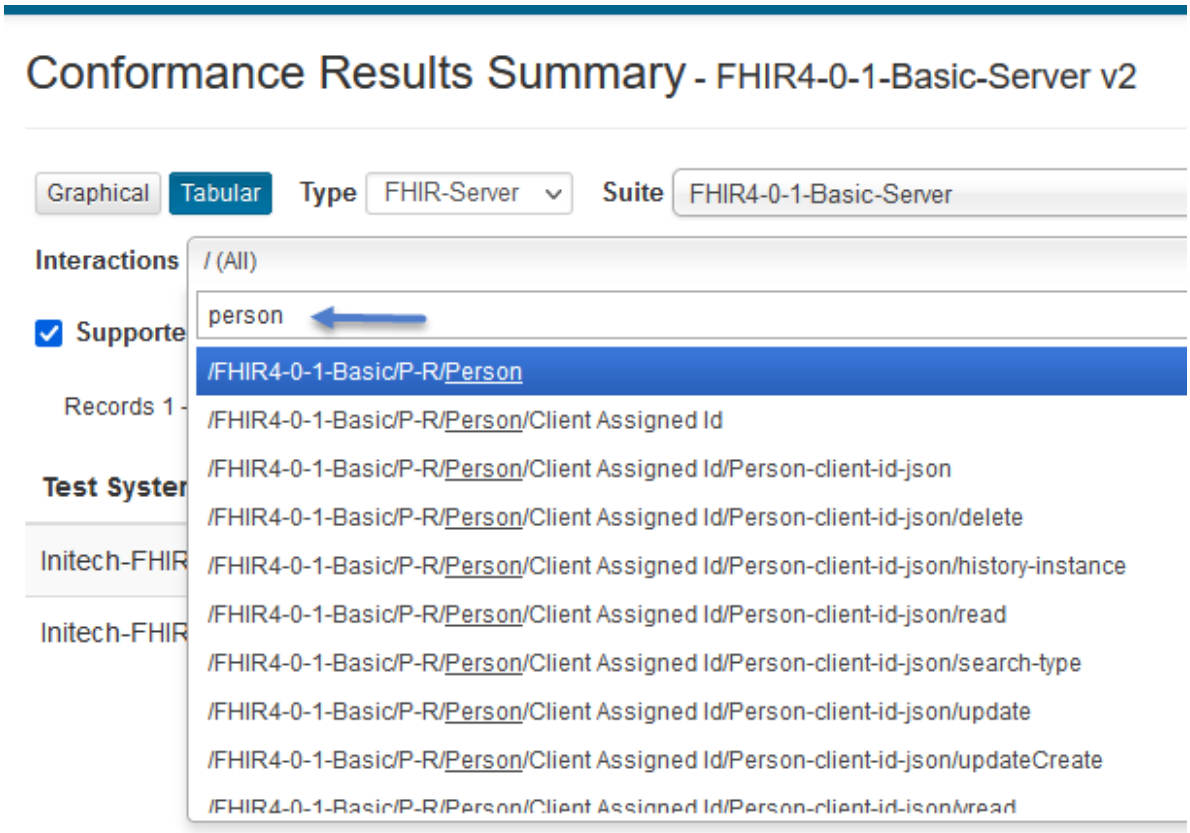
Hover over the central band to get the aggregated counts:



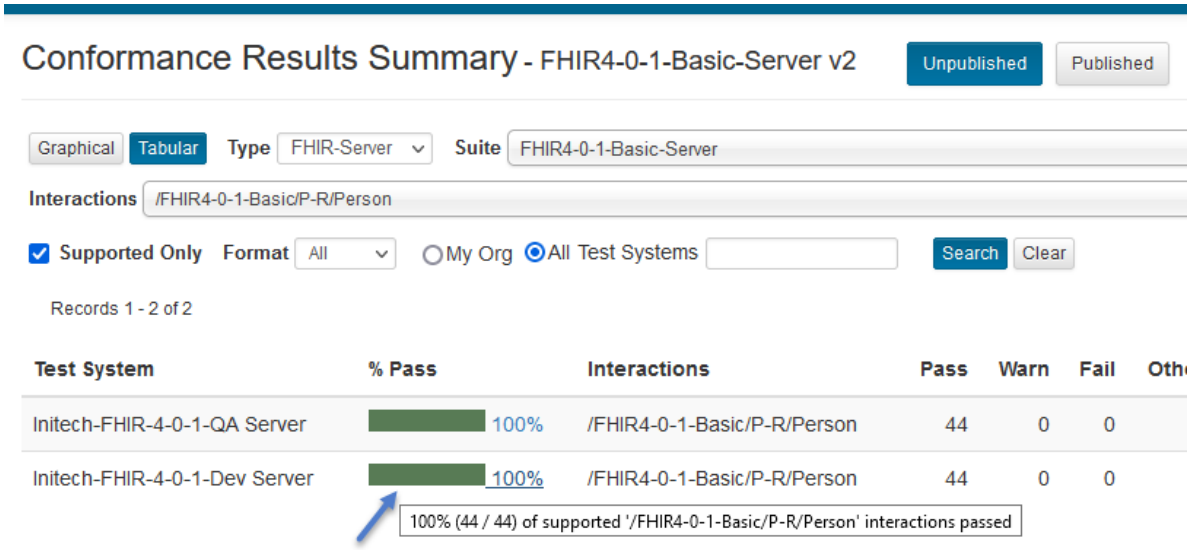
6. Change the view to Tabular:



7. Type '**person**' in the **Interactions** select box to filter for Person resource interactions and select it:



8. Notice that you can compare test systems at more granular levels. Hover over one of the progress bars:



Note: The Interactions filtering is available only on Tabular view and only when a specific Conformance Suite version is selected.

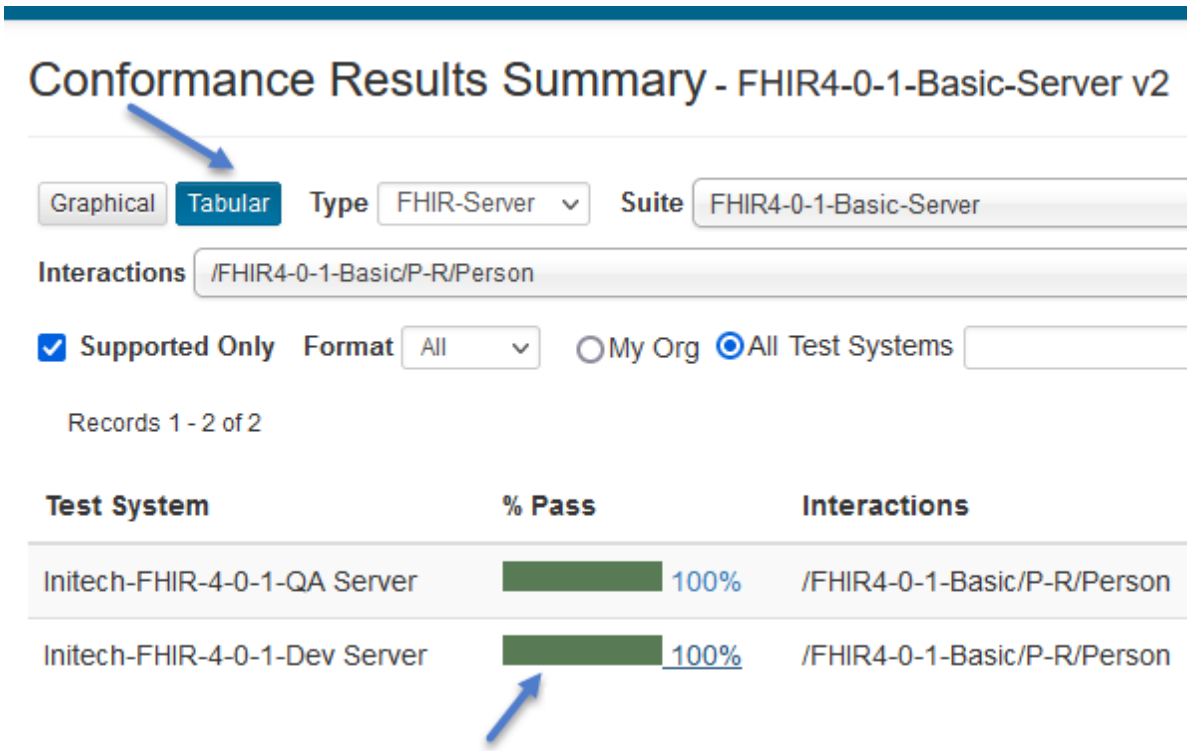
It becomes disabled when:

a. **Graphical** view is selected. This is because the summary charts in Graphical view already break down the interactions within the chart itself.

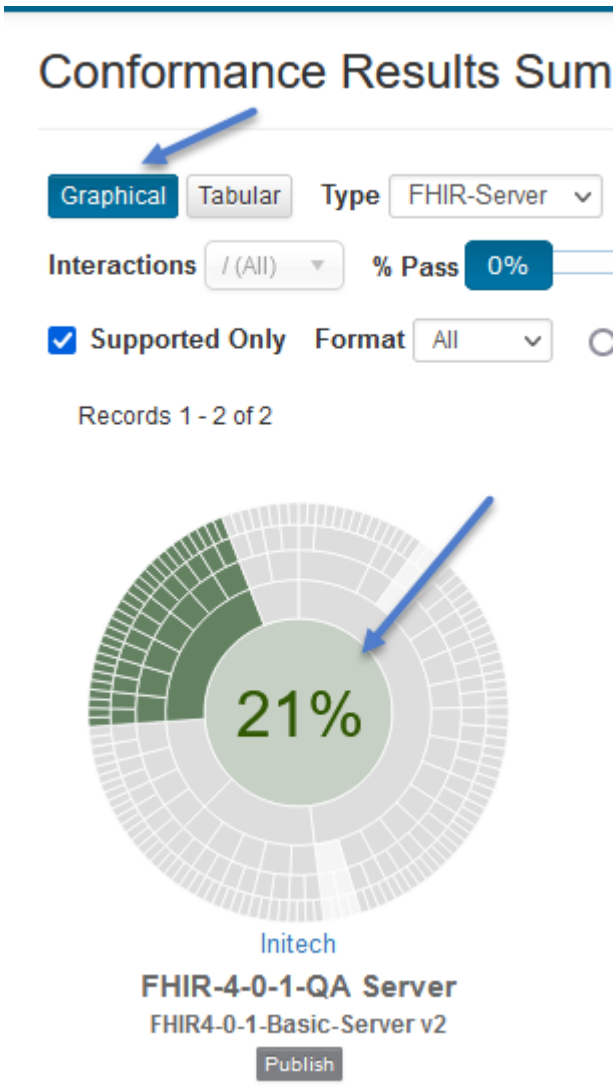
b. **All** is selected for Conformance Suite version. This is because different versions of a given Conformance Suite can have completely different test groups and test scripts.

9. You can go to the Unpublished result **details** by:

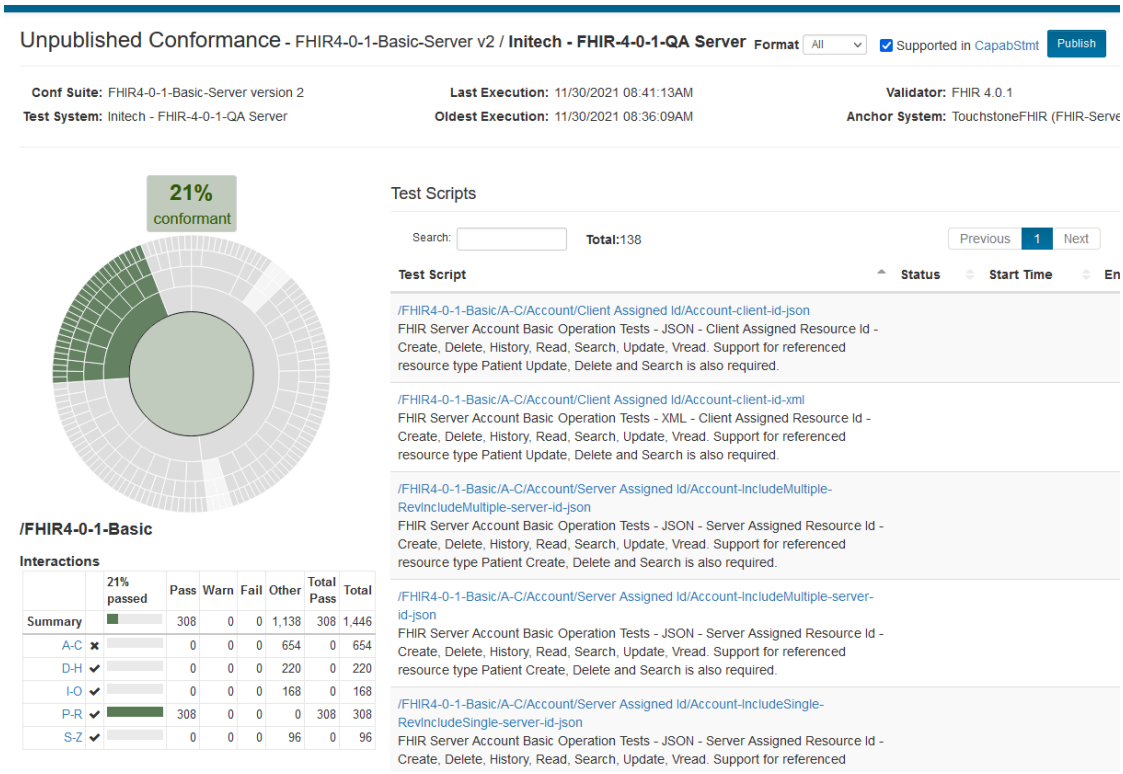
   a. Clicking on a progress bar in Tabular view:



   b. Clicking on a chart in Graphical view:



10. **Unpublished Conformance** details page:



7.4 Publishing Results

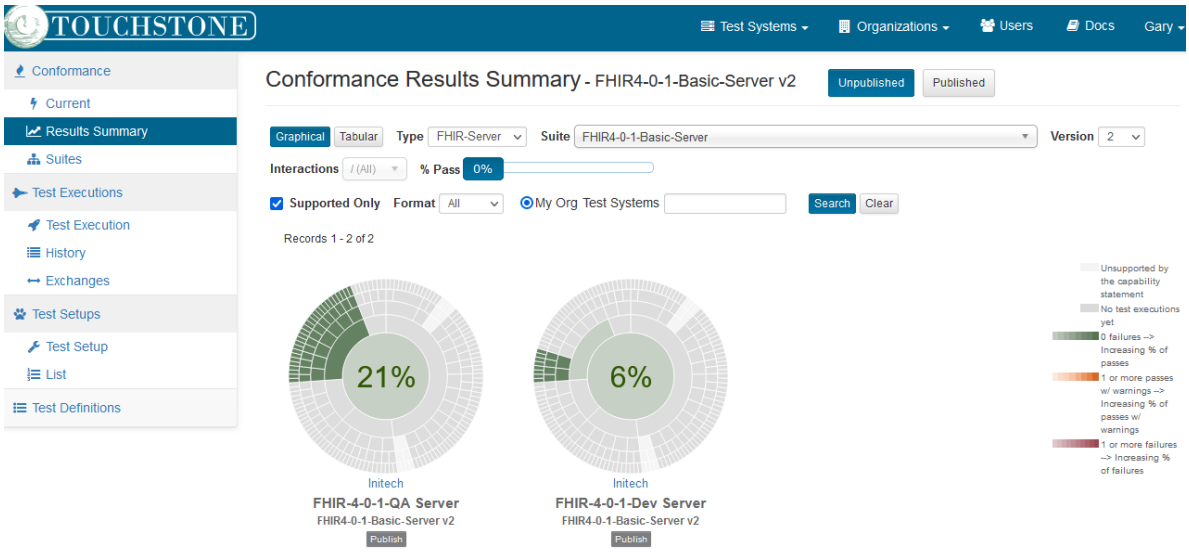
Only Org Rep users can publish results for test systems that their organization owns.

Conformance Results go through 3 stages: Current ==> Unpublished ==> Published

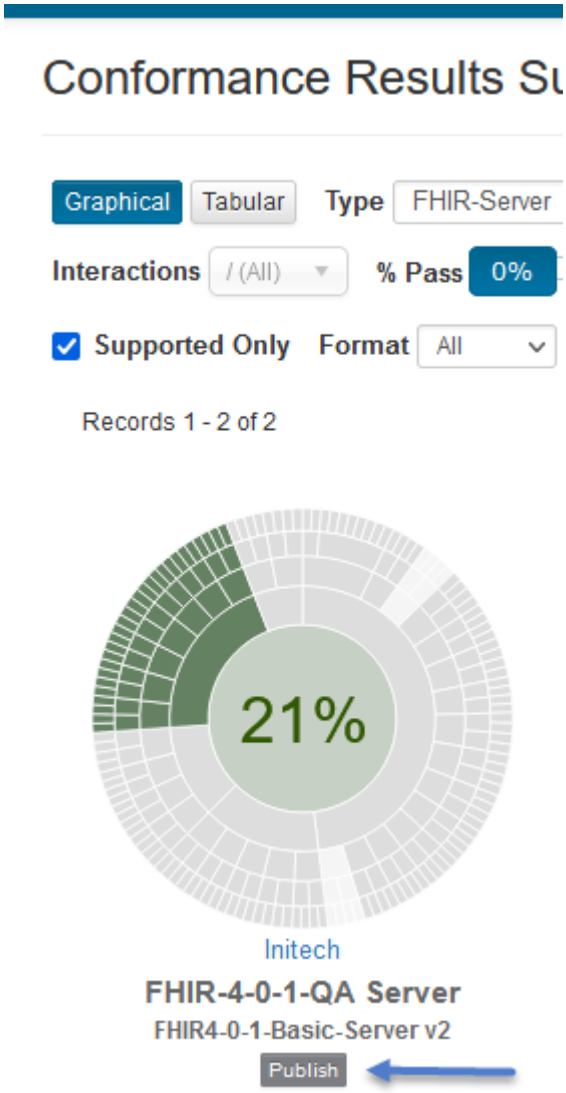
When test execution completes for one or more test scripts that are part of a Conformance Suite, a snapshot of the **Current** Conformance results is taken to create the **Unpublished** results for a given Conformance Suite version against a given test system.

When the Org Rep publishes the results, a snapshot of the **Unpublished** results is taken to create the **Published** results. Users can continue executing test scripts for the same Conformance Suite and test system without impacting the Published results.

1. Access the Results Summary page as an Org Rep and select Unpublished view:

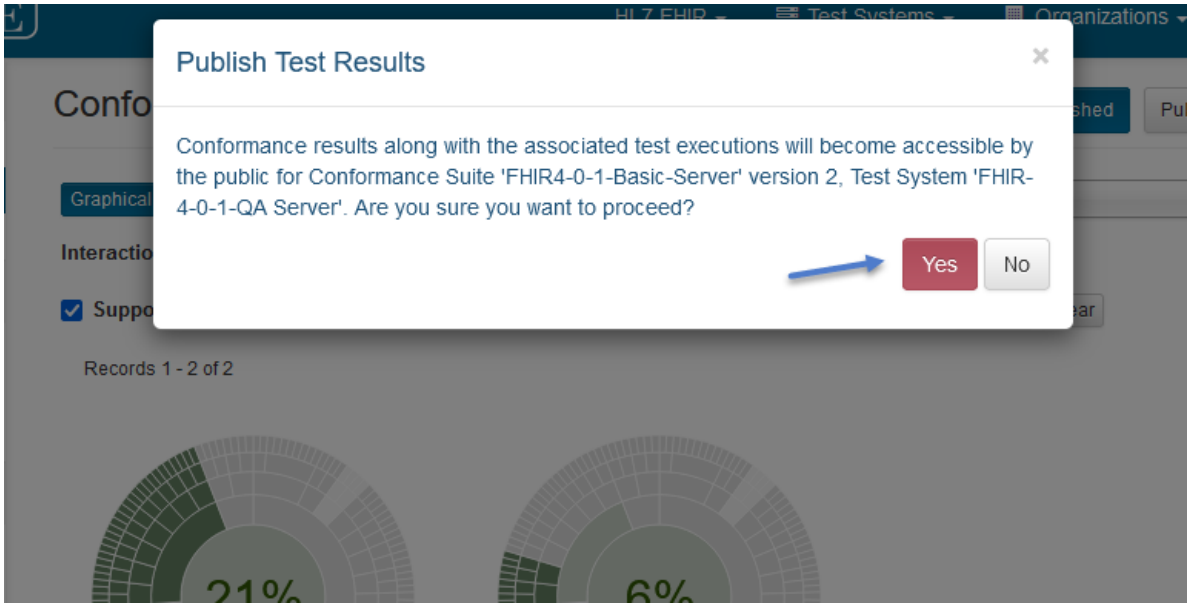


2. Click on the Publish button under one of the charts in Graphical view:

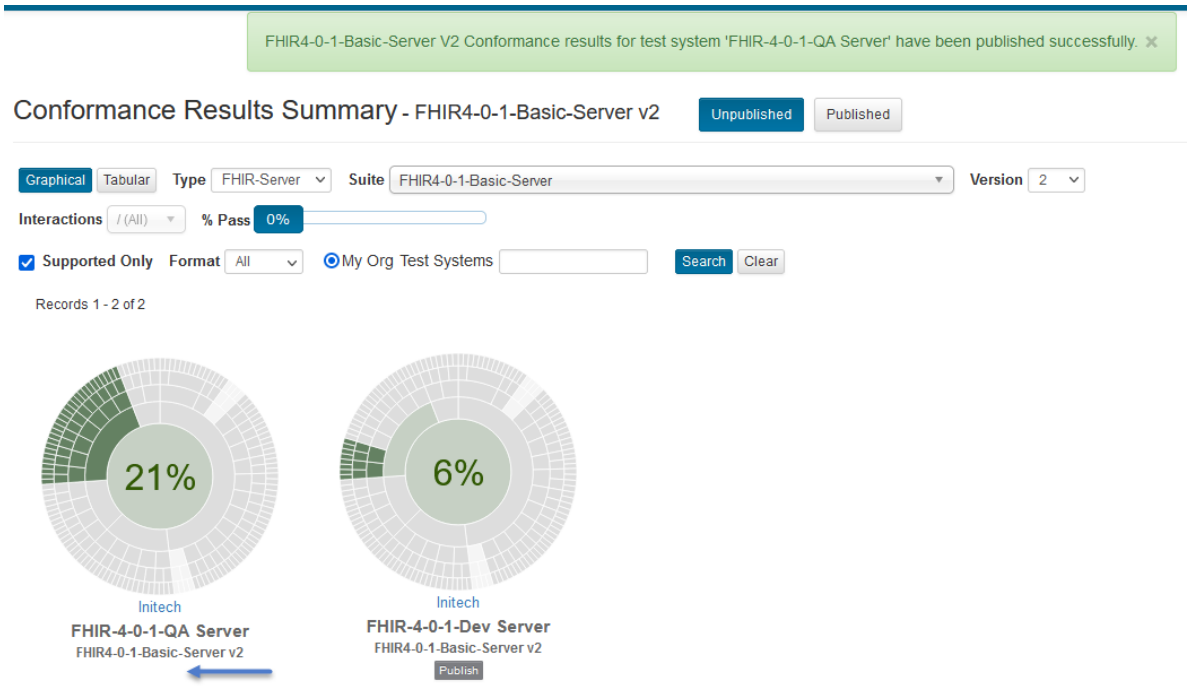


Note: The **Publish** button will be visible only to Org Rep user in Unpublished view and only if the results have not been already published.

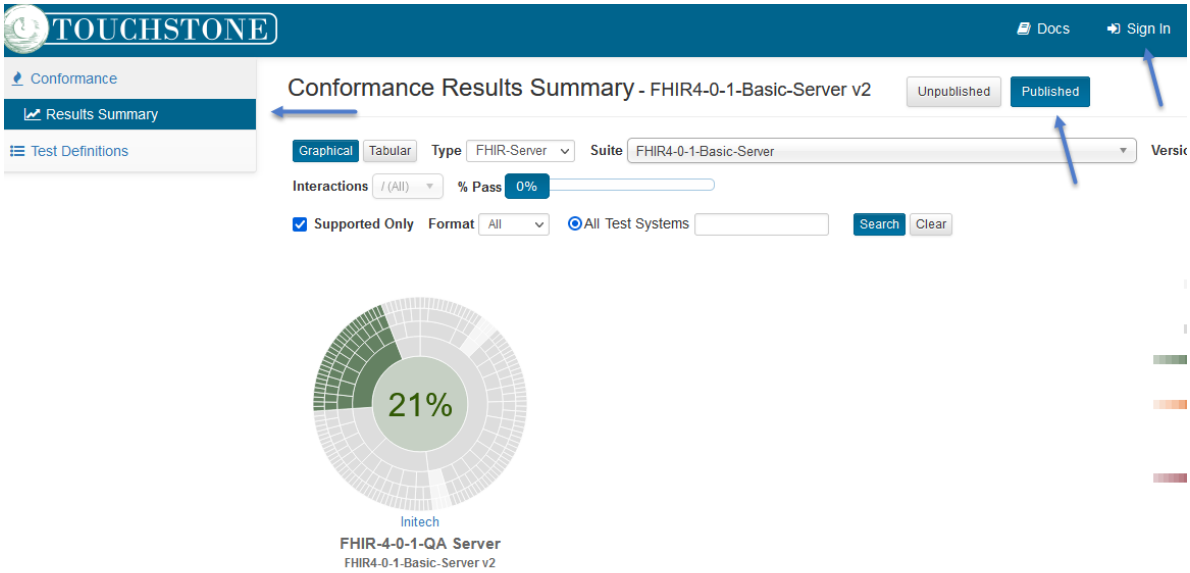
3. Conform that you want to publish:



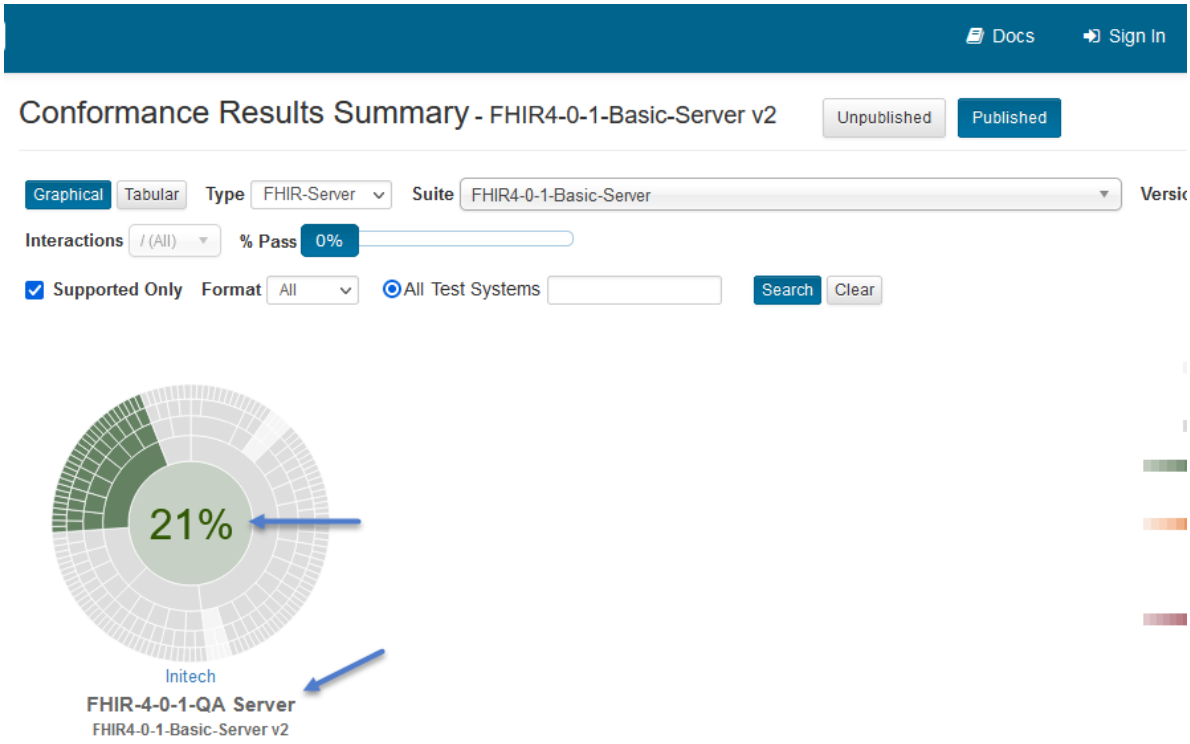
4. Notice that the Publish button is no longer displayed for the results that just got published:



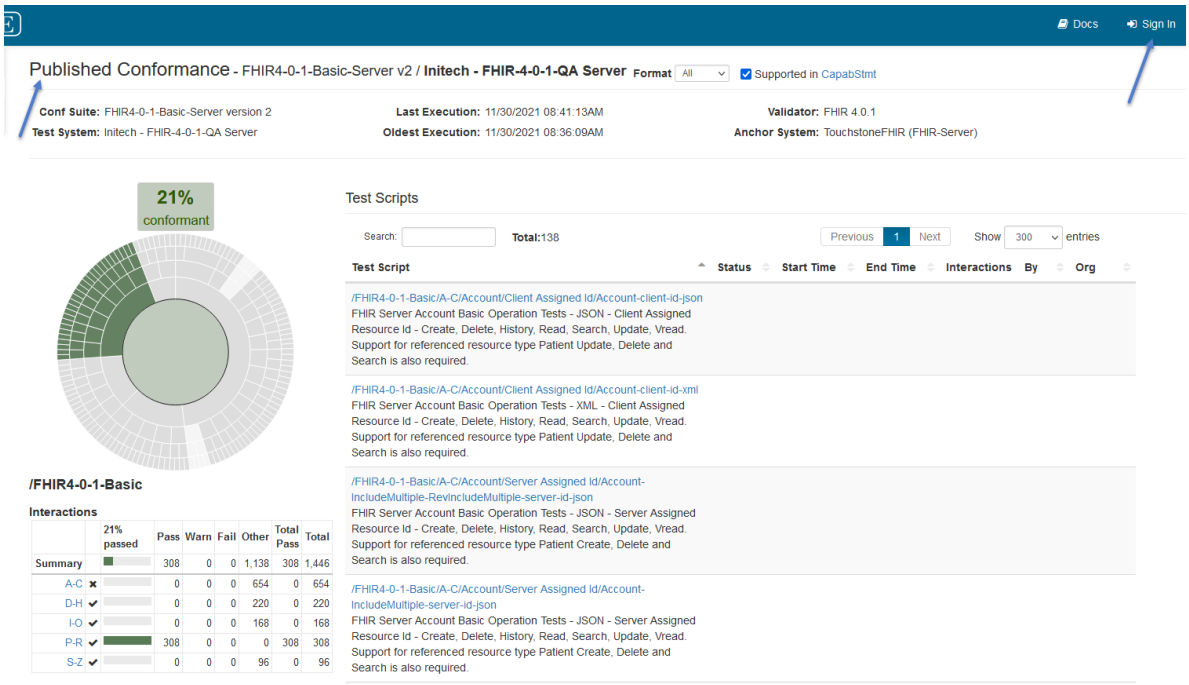
5. Notice that the published results are accessible to guest users on the Published view:



6. Guest user can go to Published details by clicking on the chart:

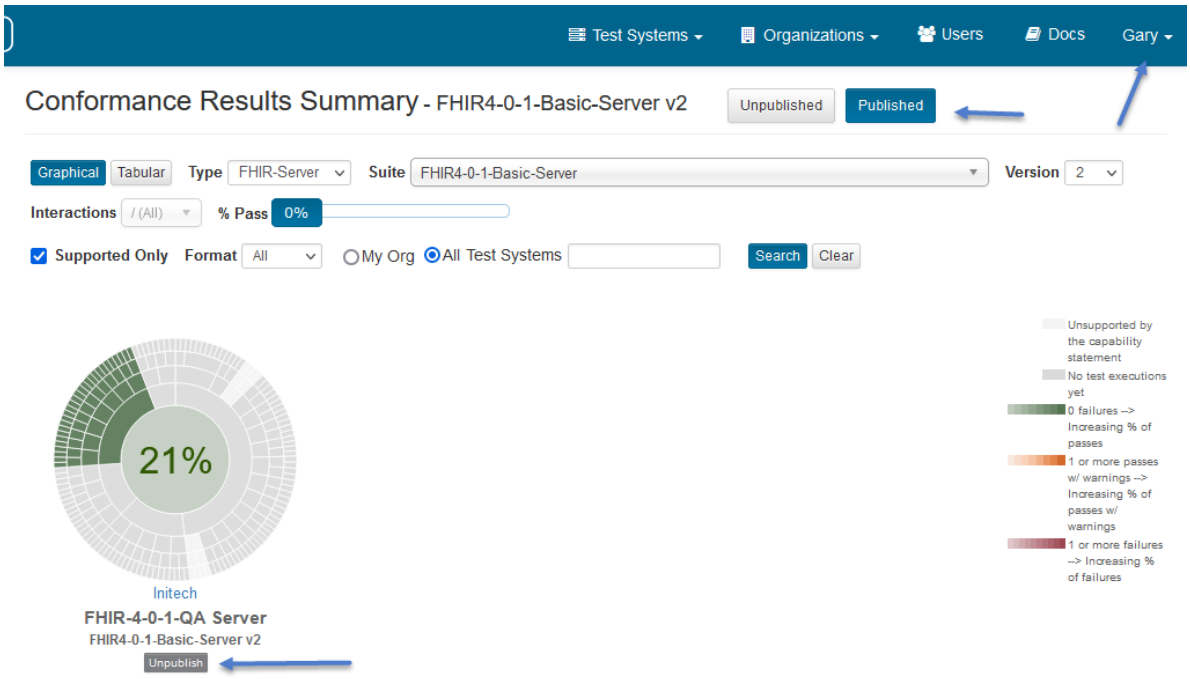


7. Published Conformance details:

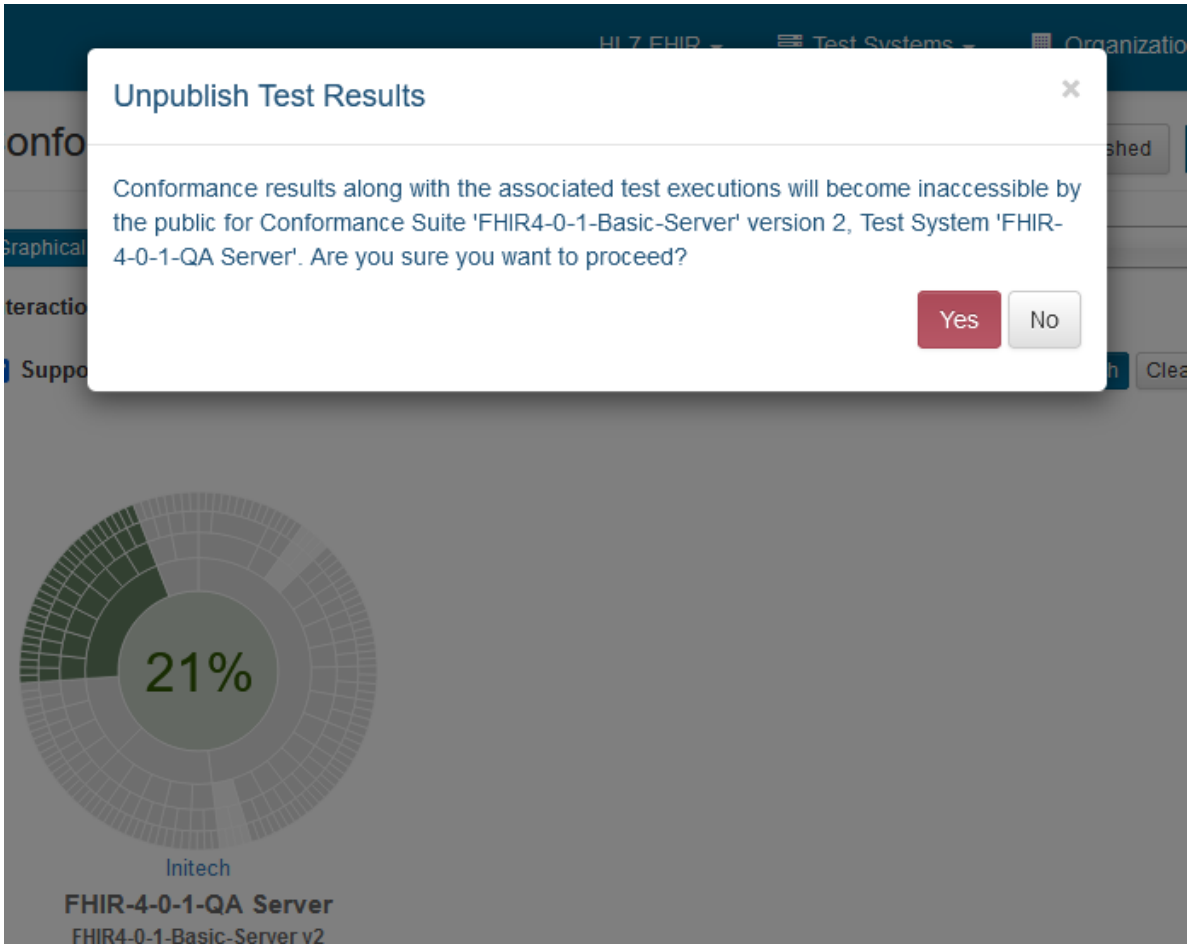


	21% passed	Pass	Warn	Fail	Other	Total Pass	Total
Summary		308	0	0	1,138	308	1,446
A-C	✖	0	0	0	654	0	654
D-H	✔	0	0	0	220	0	220
I-O	✔	0	0	0	168	0	168
P-R	✔	308	0	0	0	308	308
S-Z	✔	0	0	0	96	0	96

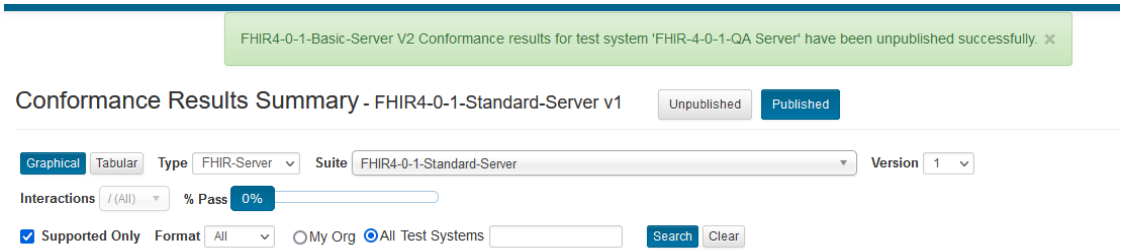
8. To unpublish the results, an Org Rep needs to access the Published view and click on the Unpublish button:



9. Conform that you want to unpublish:



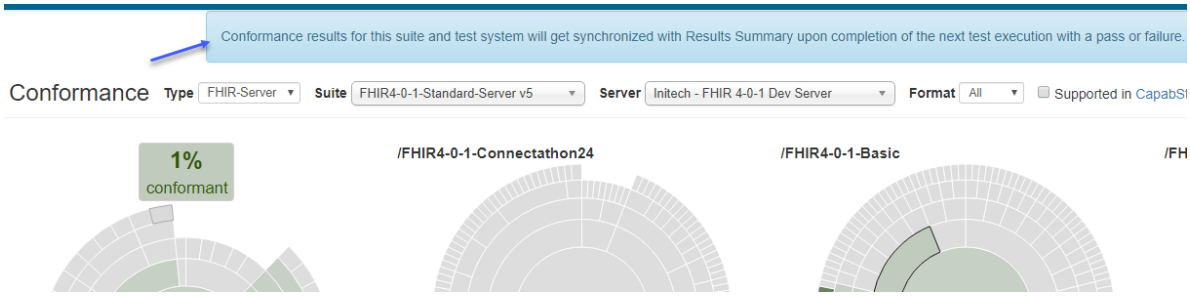
10. Unpublished successfully:



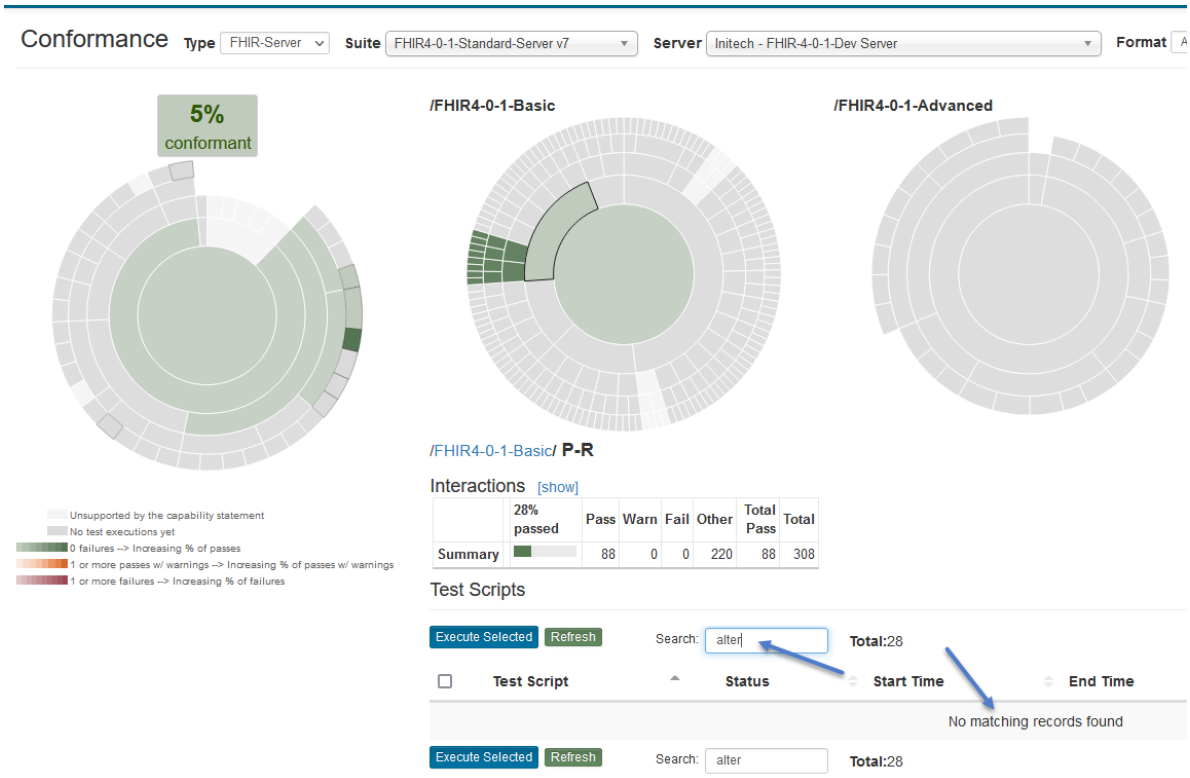
7.5 FAQ

1. The '% Conformance' on the Current page is sometimes different from Result Summary page for Unpublished results of a given suite and test system. Why is that? If a suite is newly created or the version has incremented, there might still be test executions for the test scripts contained within the suite against the selected system. The Current screen takes into account all those test executions whereas Results Summary screen does not. The two screens will get synchronized upon completion of the **next** test execution.

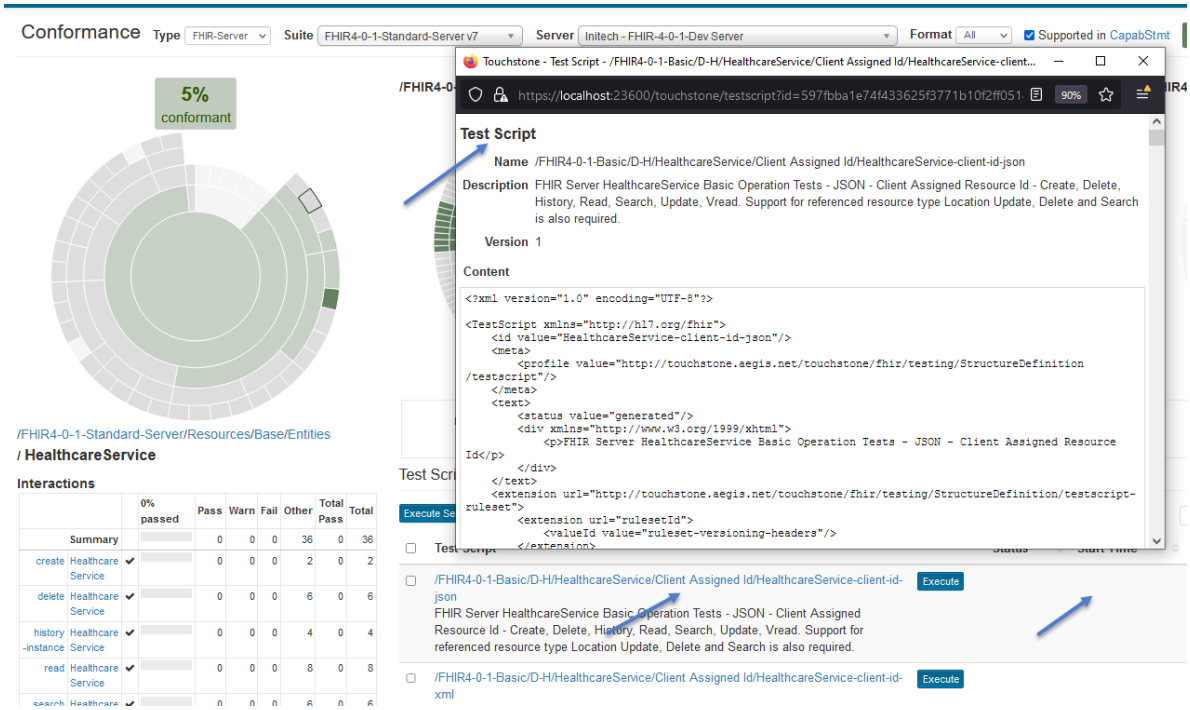
   This is indicated on the Current screen when the two results are out of sync:



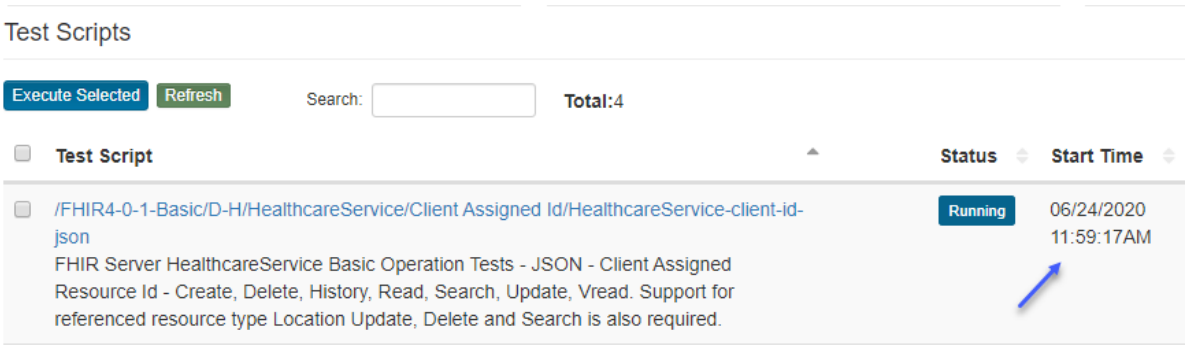
2. I've selected a band and don't see any test scripts. Why is that? Make sure you've cleared the filter box (blue arrow below):



3. When I click on the Test Script link, it sometimes pops the TestScript resource definition and at other times it navigates to the Test Script Execution screen. If the test script has never been executed before (as is the case below), clicking on the Test Script link will pop the Test Script resource definition.

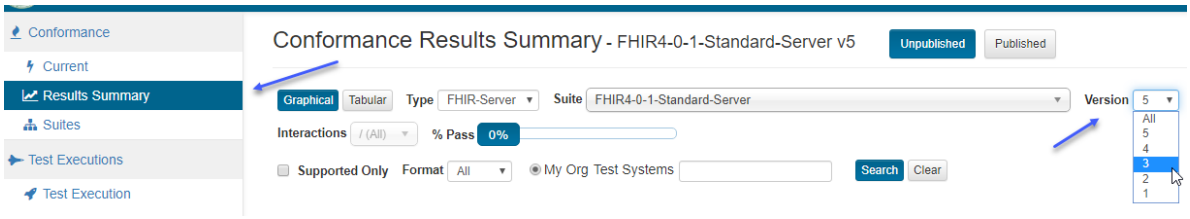


If the test script has been executed already, then clicking on the Test Script link will take you to the Test Script Execution screen. Aftr launching the execution, you may have to wait a few seconds before the Start time appears:

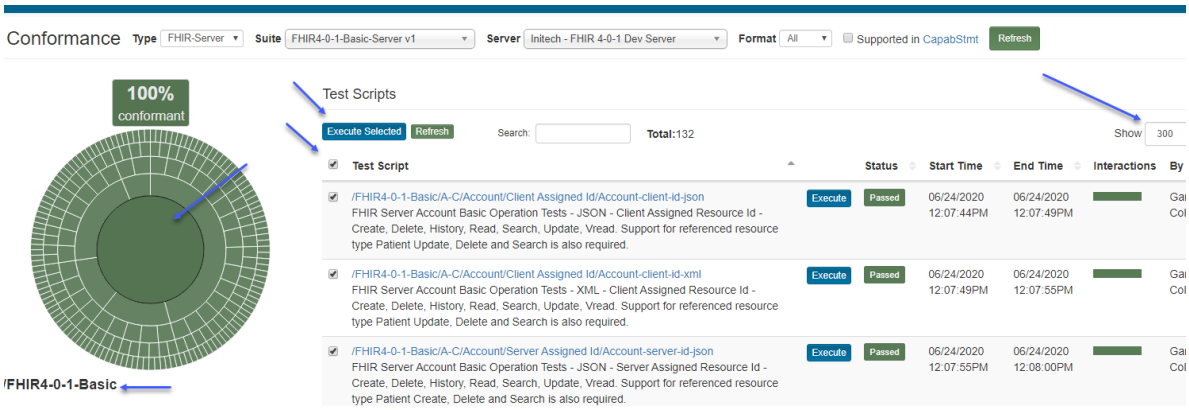


4. I reached a higher '% Conformance' and the system has knocked the % down. I was green on certain bands and now the system shows gray again. Why is that? This can happen if the conformance suite version has incremented. It would be erroneous for Touchstone to continue to show green or red for a conformace suite version that you haven't executed yet.

   The conformance results of previous versions are maintained though and you can see them on Results Summary page by changing the Version dropdown:



5. The Conformance Current page is tedious. I don't want to execute one test script at a time after I've reached my desired '% Conformance' level. You don't need to. Touchstone stores the variable values you entered for test scripts that require variable values. You can click on the root node (**/FHIR4-0-1-Basic** in this case), change the page size to 300, check the "All" checkbox and click on "Execute Selected". This will launch all the test scripts as part of one execution.

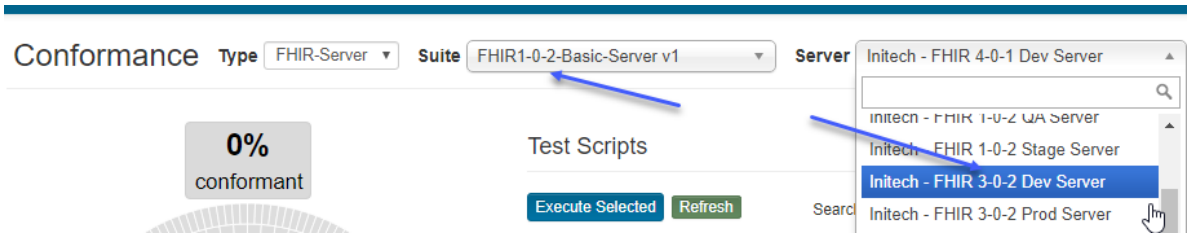


6. Does my '% Conformance' stay the same from one version of the suite to the next? No. It is monitored by version. The '**% Conformance**' resets to zero when a conformance suite version increments. Changes to test groups, categorization, and other components of the suite could drastically change the interaction totals and meaning.

   Please refer to Versioning for the list of events that can cause the suite version to increment.

7. Are the Conformance results going to be different from the results that I would get by executing tests via Test Definitions screen? They are the same from execution standpoint. Test executions launched from Test Definitions screen can be viewed on Conformance screens and vice versa. Every launch of test execution on Conformance screens creates a Test Setup in the background and launches it. You can view those test setups on Test Setups screen and launch from there if you like. The difference though is that Conformance screens relay the test system's conformance level while the test executions screens don't.

8. Why does my test system show up in the Test System dropdown if it doesn't even support the validator version that the suite is on? You may choose to execute test scripts of one specification version against a test system that hasn't been declared to support that specification version yet and see how it responds. After all, this is a testing environment. For that reason, we don't filter test system dropdown based on suite's validator version. The dropdown though displays only test systems that's accessible by you and not all test systems.



CHAPTER

EIGHT

CLIENT/PEER-TO-PEER TESTING

In Peer-to-Peer testing, test systems within your organization can exchange messages with other test systems within and outside your organization. The interactions would take place via Touchstone proxy endpoints. This allows Touchstone to capture the messages and run the assertions in test scripts.

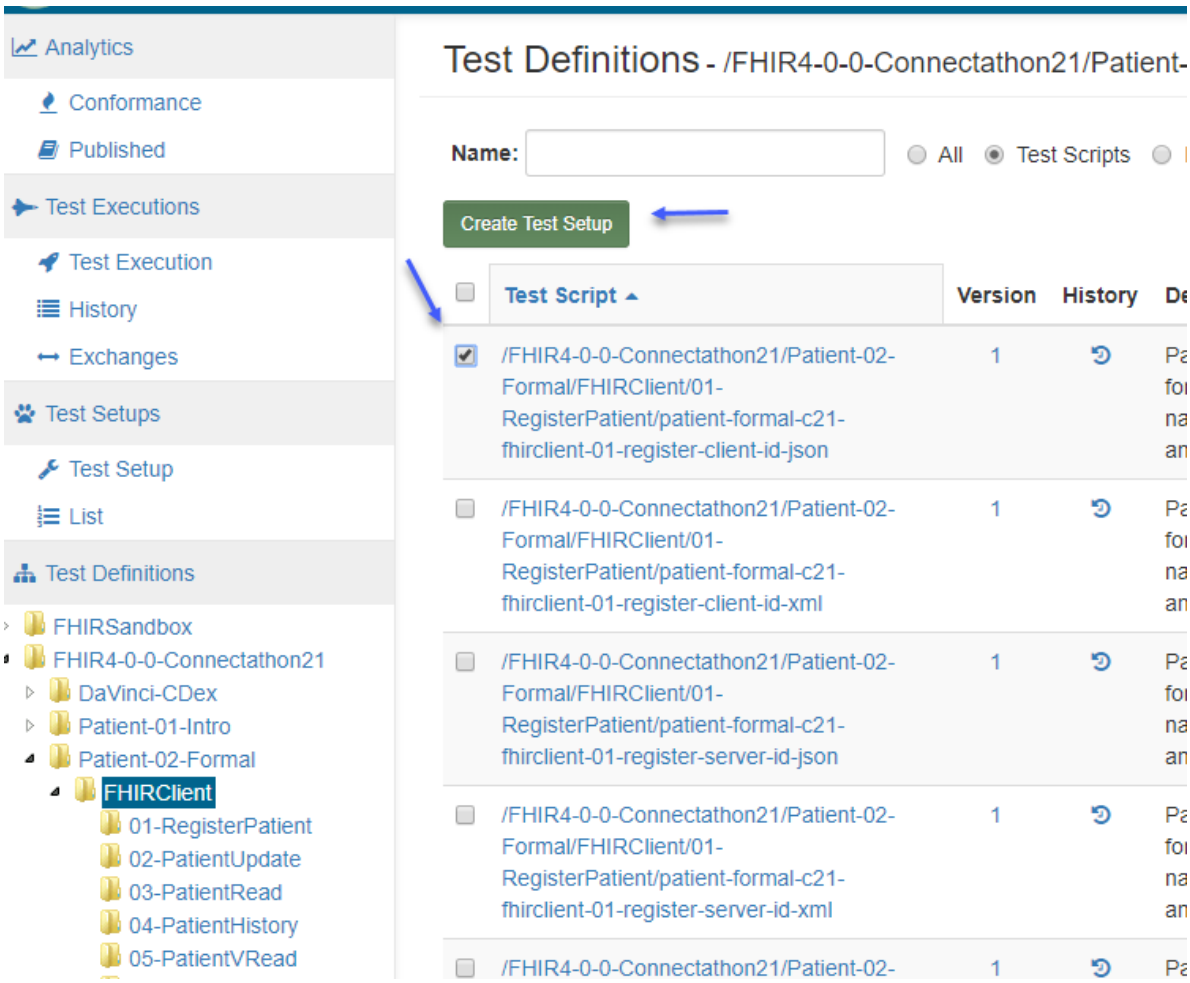
Your test systems can both initiate and respond to requests from other test systems.

Here is a diagram that depicts such communication:

8.1 Launching Executions

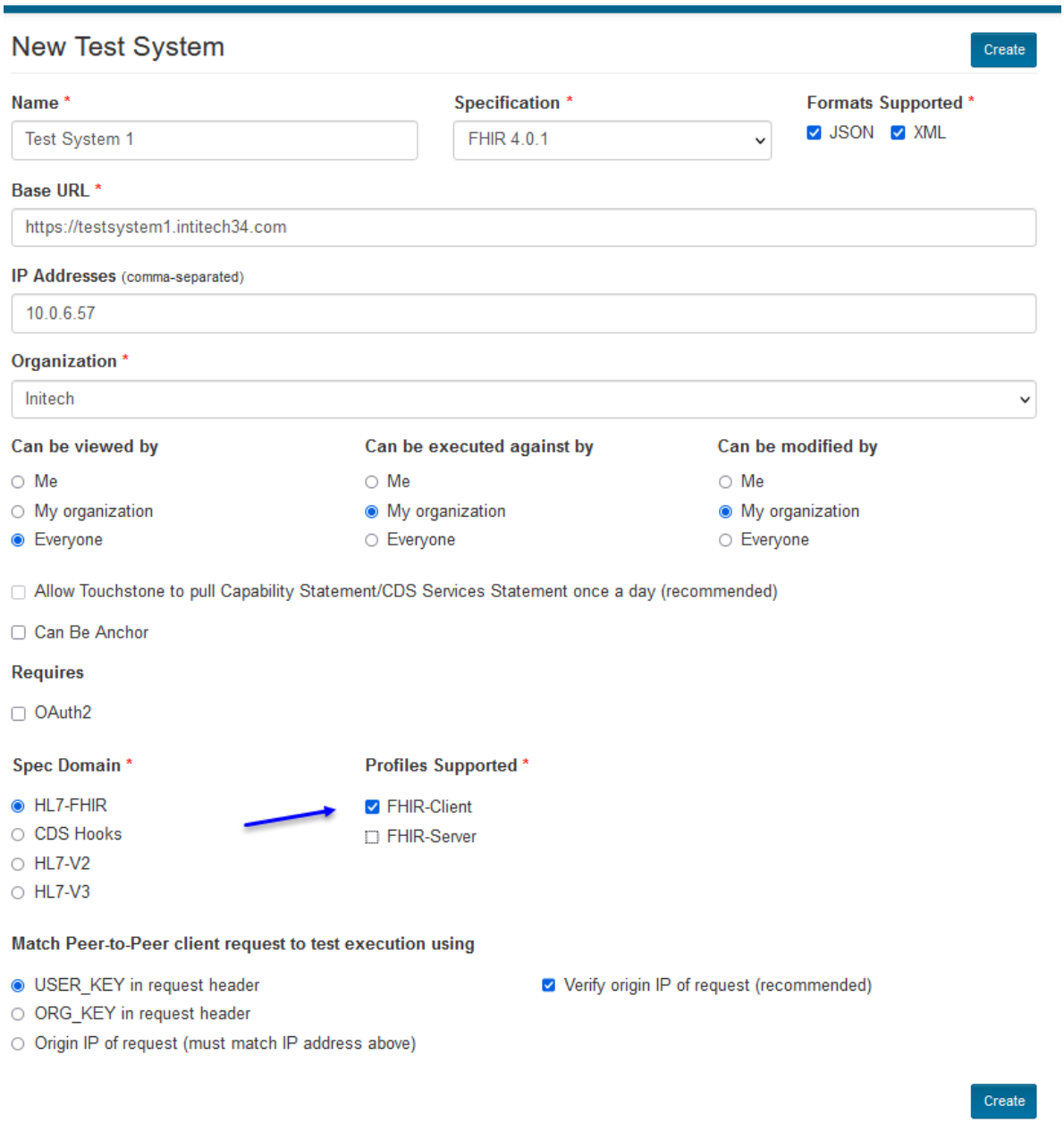
To execute a Client-side test script, you can take the following steps:

1. Under Test Definitions, find the test script that supports client-side testing, select it, and click the `Create Test Setup` button:

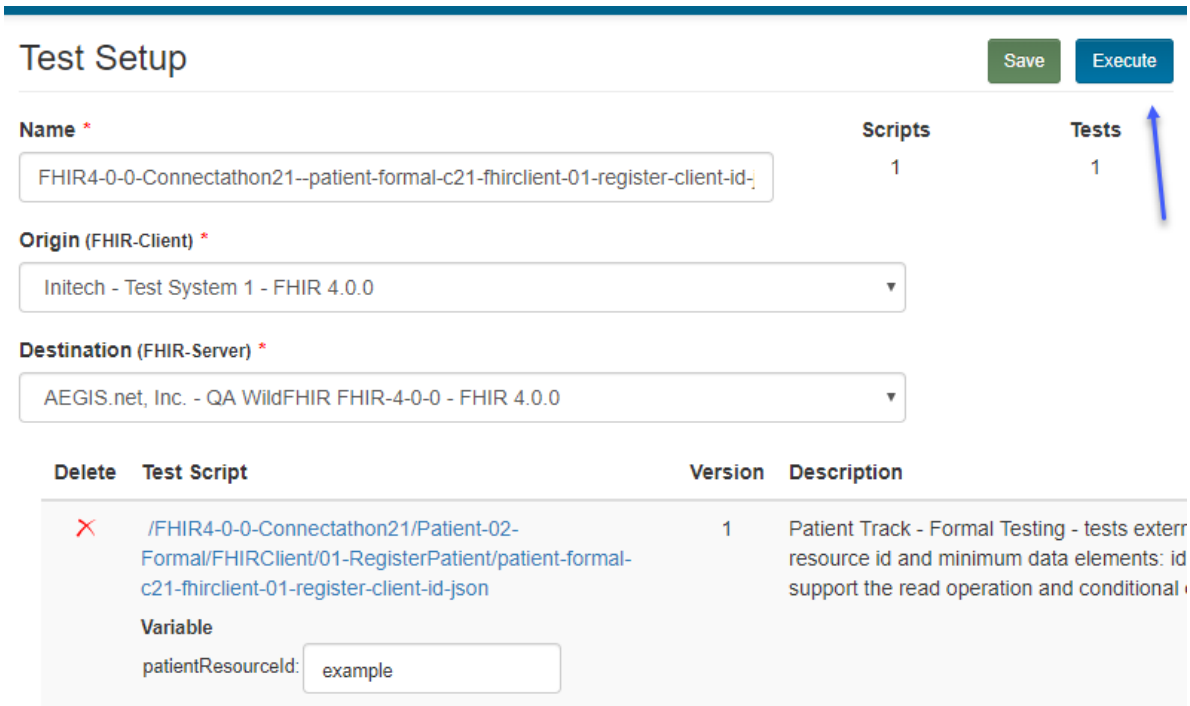


2. Select your test system (client/initiator system) as the Origin. It's marked in blue below. The target of the exchange will be the Destination server and is marked in red below:

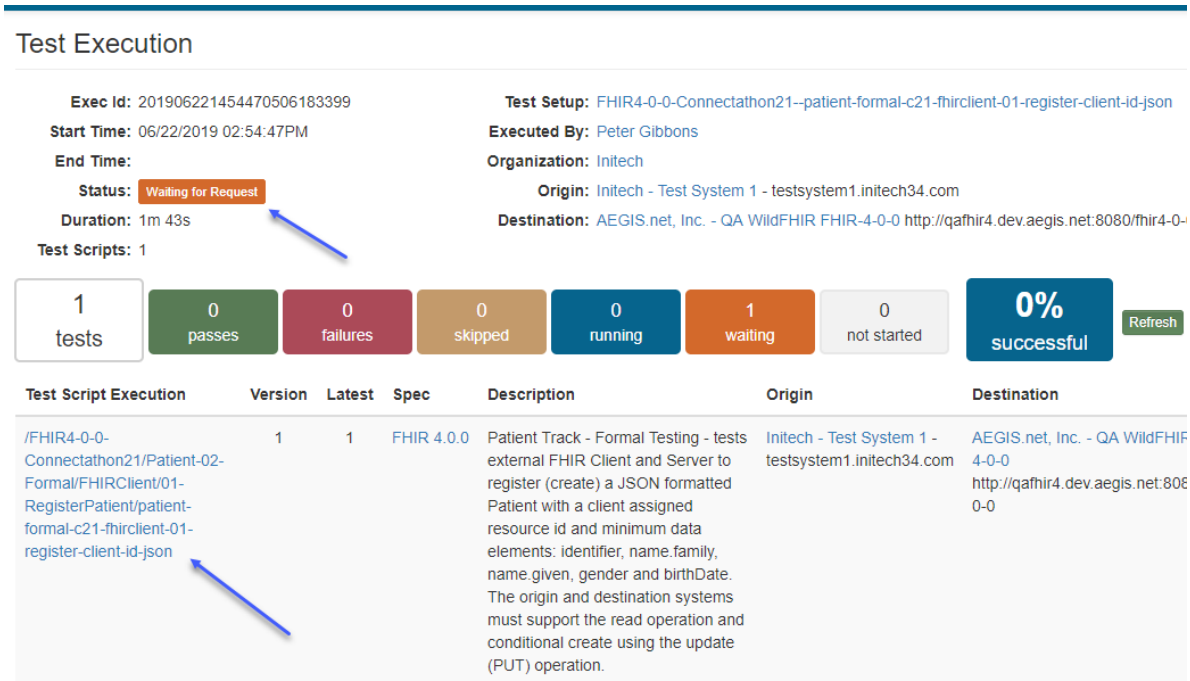
To get your server to appear in the Origin drop-down, you need to select the `Client` option as a supported profile in your test system's configuration:



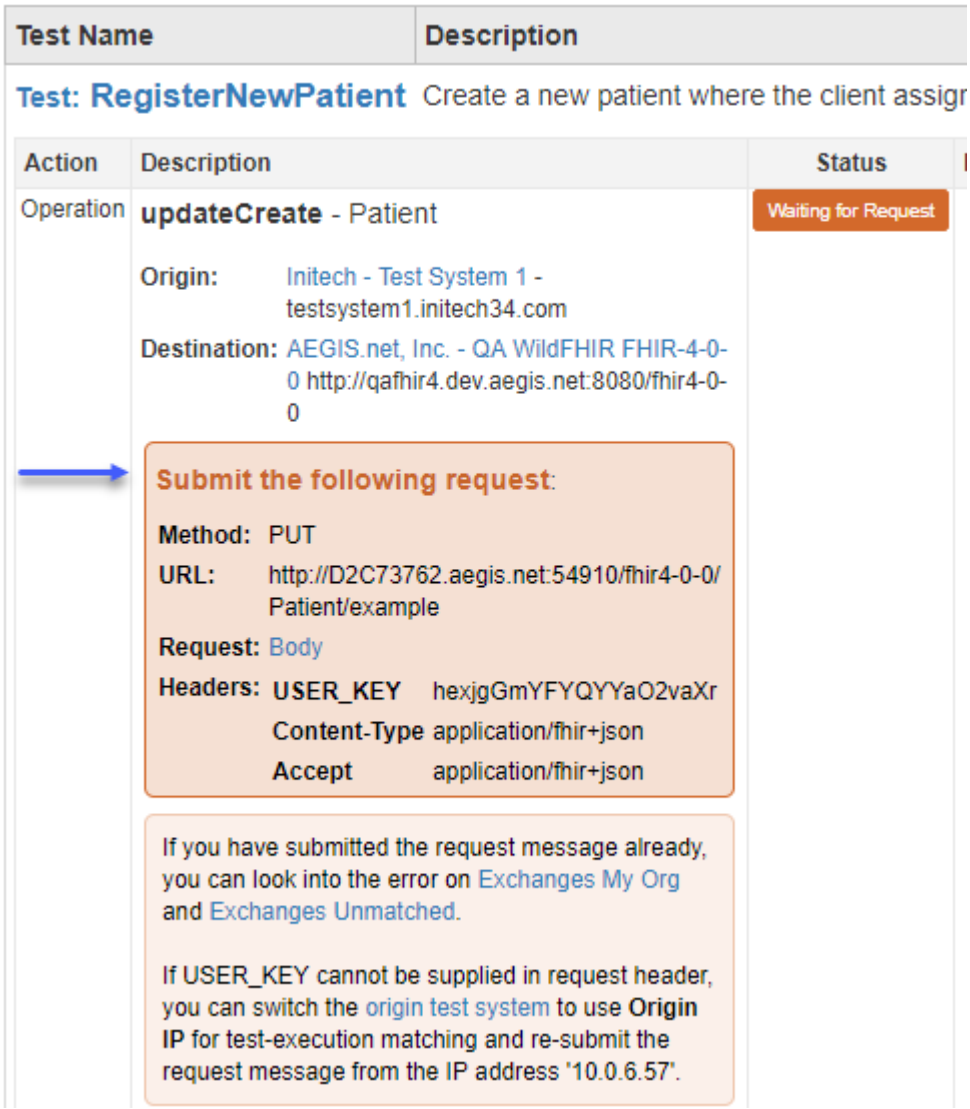
3. Click on `Execute` button to launch the test execution:



4. You will land on the `Test Execution` screen. The Test Script will have sections that don't need your intervention (e.g. Setup and operations where <origin> element is missing). The test script execution proceeds until it hits an operation where <origin> element is present and for which input is required by the client test system i.e. you. That operation execution along with its test script execution and test execution will assume the status of `Waiting for Request` at that point:



5. Click on the test script link above to get to the Test Script Execution screen:

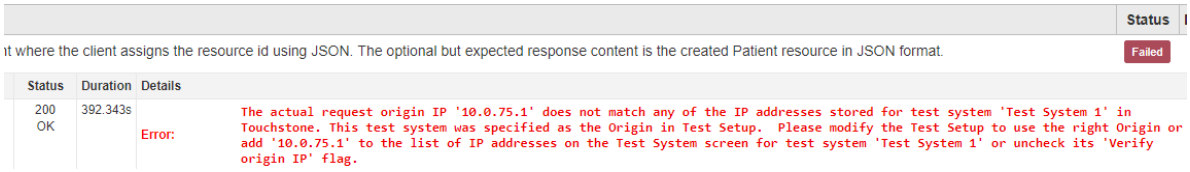


In the panel highlighted above, you will find all the data you need to submit as a FHIR Client. It has the following information:

- The HTTP **method** you need to use (GET, POST, PUT, etc.) and the URL you need to send your request to. This URL will be the Touchstone Proxy URL and will be unique to each test system. *It will be different from the Base URL of the test system*. The Proxy URL allows Touchstone to intercept the message and execute the assertions for that operation execution.
- The Request payload (**body**) if the method is PUT or POST.
- The **USER_KEY** that needs to be in the request header. This USER_KEY value will be unique to each user. It can be provided in the request header or request body and will be used by Touchstone to tie the intercepted message to your test execution.
- Any other headers that are required (e.g. Accept, Content-Type, etc.).

> **Warning:** It is important to match the request data being sent from your FHIR Client to the one indicated in the panel above.

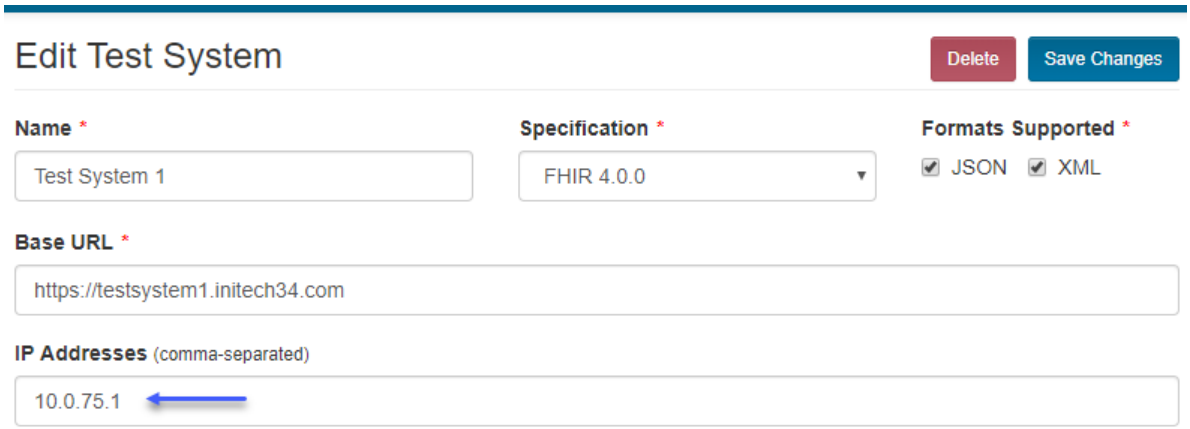
6. You might get the following error if you are in the initial stages of using your FHIR Client system in Touchstone:



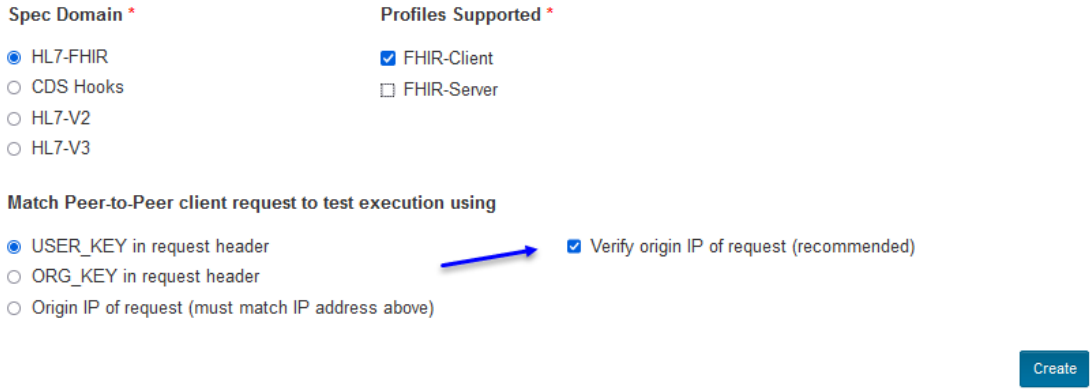
For added security, Touchstone verifies that the the actual IP address of the request matches the IP address of your test system in Touchstone. You get the error above if they don't.

You can do **one** of the following to get around this error:

a. Add the IP address indicated in the message above (`10.0.75.1` for the case above) to the list of IP addresses of your test system:

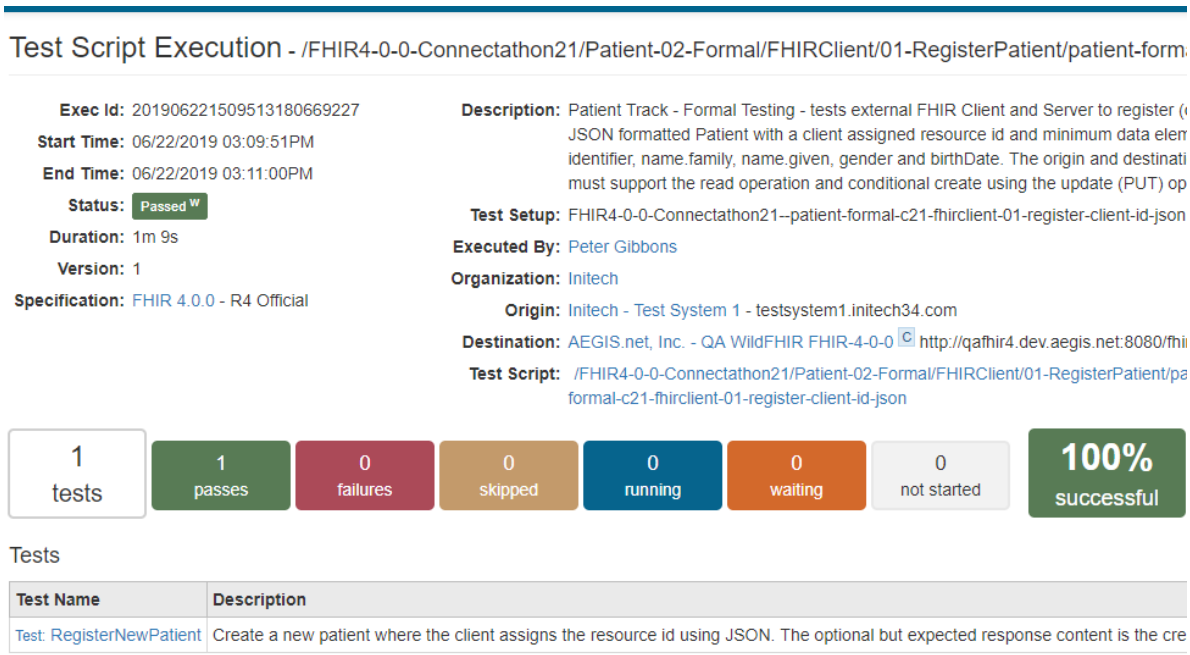


b. Uncheck this box on your test system.



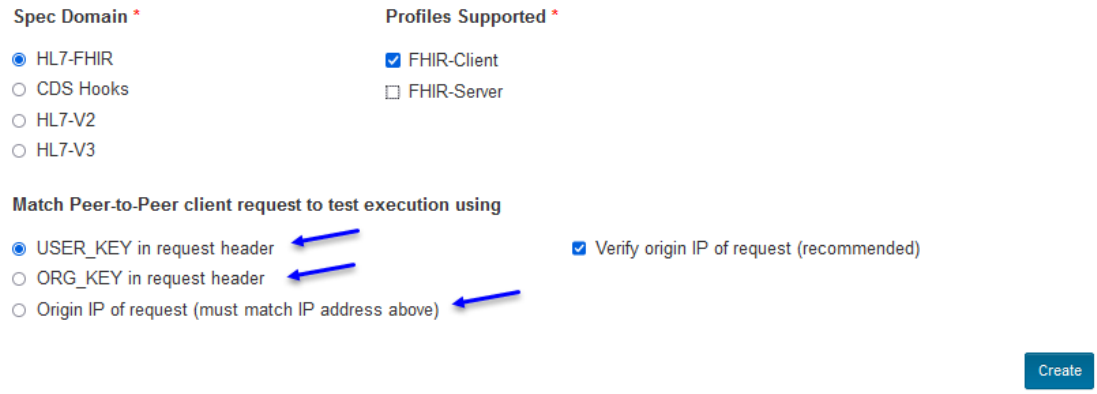
It is recommended to do option (a). If you do option (b), then other users can configure their test setups with your test system playing Origin and submit requests from their FHIR Clients. This will give the impression to end-users that the request came from your test system when in fact it came from another FHIR Client. If you don't have such a concern (because you have marked your client test system as accessible/executable only by 'My Org'), then option (b) is more convenient as you don't have to change the test system's IP Address in Touchstone every time the IP address of your FHIR Client changes.

7. You continue to initiate from your client test system as required by any remaining `Waiting for Request` operations until the test script execution completes successfully:

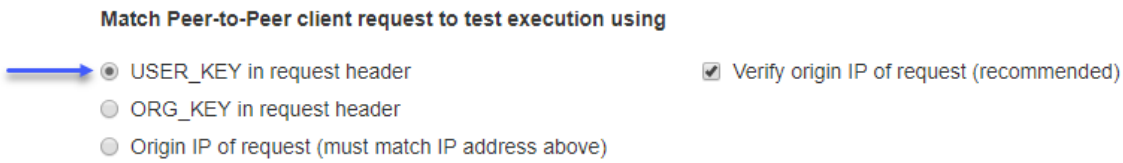


8.2 Execution Matching

When the Touchstone Proxy receives a request from the client system, it has to associate the request to the user's active test execution. There are three mechanisms in Touchstone to do that and they can be specified when editing the client/origin test system:

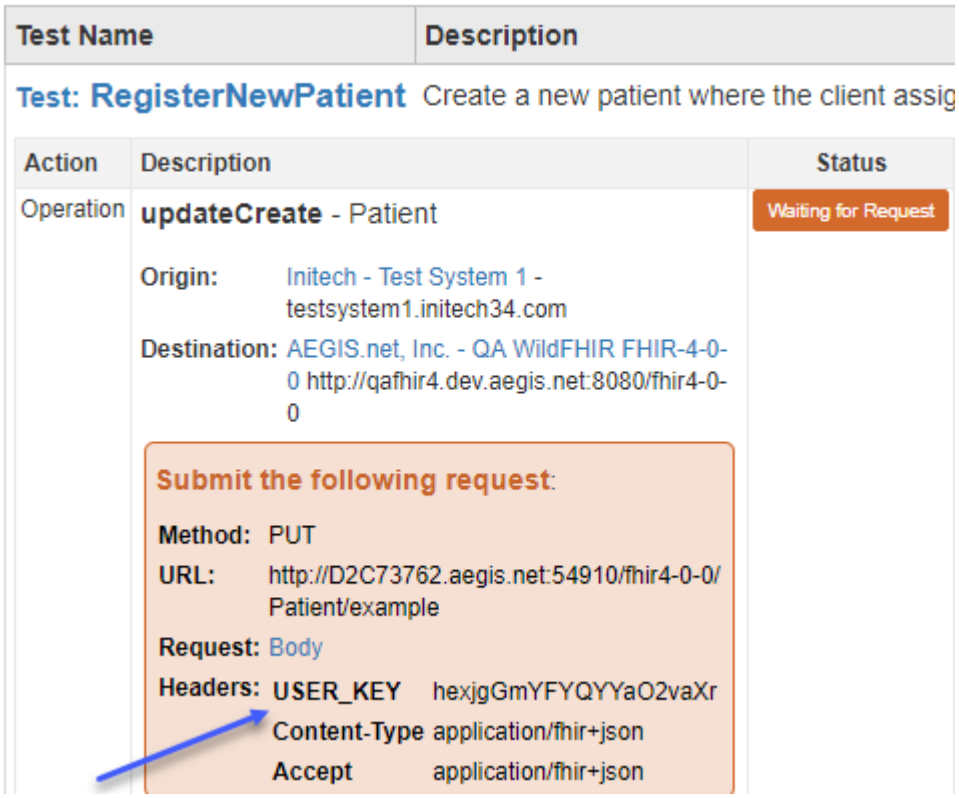


1. Touchstone first looks for `USER_KEY` in the request headers. If it finds it, then it stops looking any further. For client systems that can specify custom request headers, it is **highly recommended** to keep the default option of `USER_KEY` as the matching mechanism:

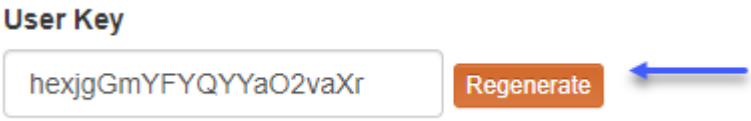


Because each USER_KEY is unique to a user, Touchstone can find the matching active test execution even when there are multiple active test executions from users within your organization. This makes the `USER_KEY` matching the least error-prone of the three options.

The instructions on the Test Script Execution's waiting step are tailored for the matching mechanism chosen on the client test system. When USER_KEY is chosen as the matching mechanims, the instructions are as follows:

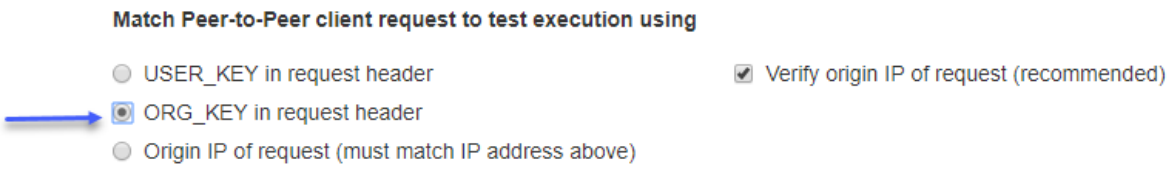


You can regenerate your USER_KEY under My Settings if it has been compromised. You'll know if it has been compromised if your Waiting test executions progress to completion without any action on your part.



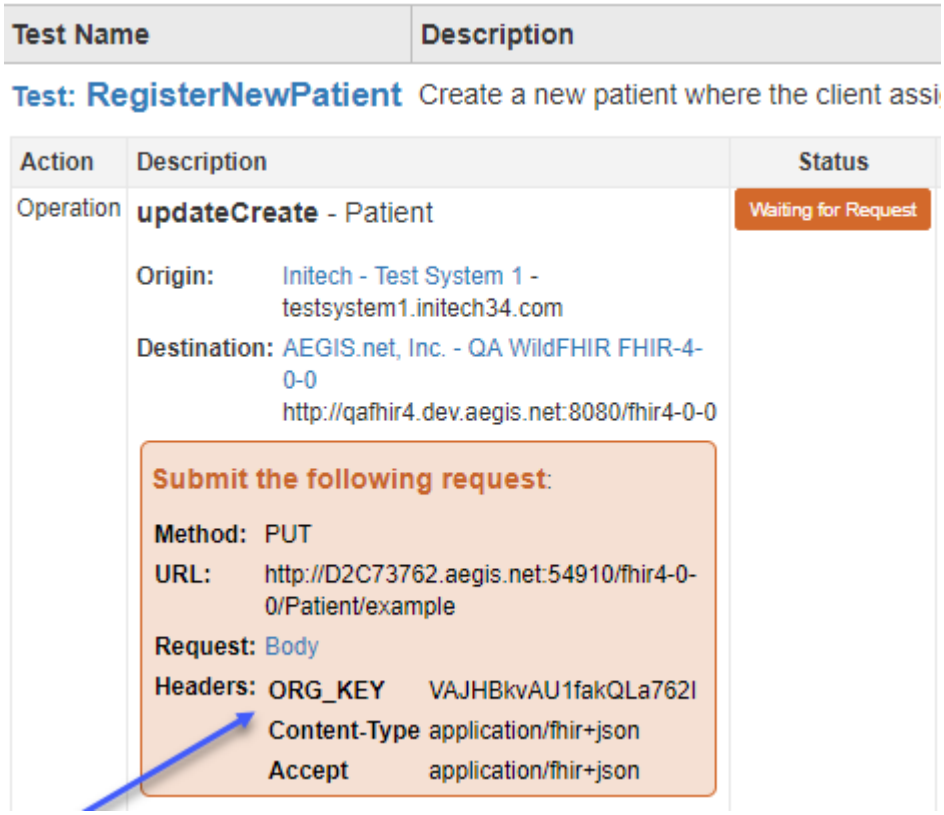
2. Touchstone then looks for `ORG_KEY` in the request header (if `USER_KEY` is missing). If it finds it, then it stops looking any further.

   If it's difficult for your client test system to specify distinct USER_KEY values for each user within your organization, then using ORG_KEY will probably be easier than using USER_KEY as the ORG_KEY will be the same for all users within the organization. You can select `ORG_KEY` matching mechanism on the client test system if that's the case:

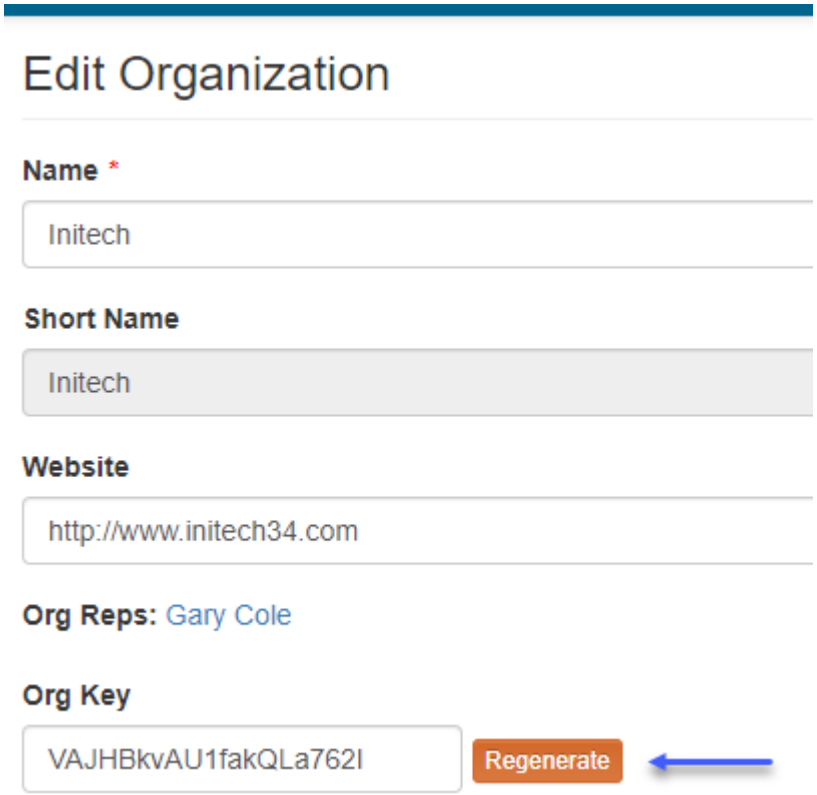


The disadvantage of using ORG_KEY though is that only one test execution can be in `WAITING_FOR_REQUEST` for the entire organization. Use of USER_KEY does not have this limitation. When using `USER_KEY`, **multiple** test executions can be in `WAITING_FOR_REQUEST` status for the organization but each user can still have only one test execution in `WAITING_FOR_REQUEST` status.

The instructions on the Test Script Execution's waiting step are tailored for the matching mechanism chosen on the client test system. When ORG_KEY is chosen as the matching mechanims, the instructions are as follows:



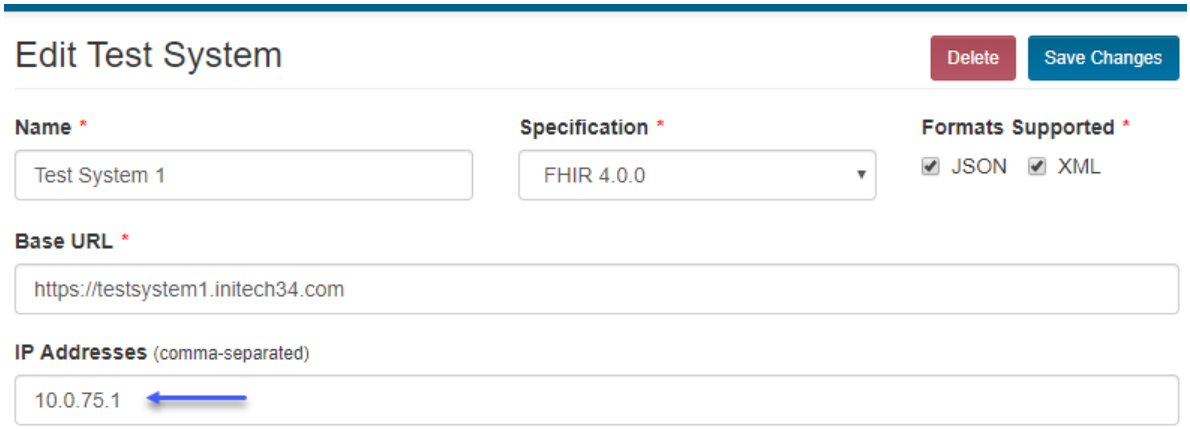
Each organization is assigned a unqiue ORG_KEY by Touchstone. It can be regenerated by the Org Rep on Edit Organization screen if it has been compromised. You'll know if it has been compromised if your Waiting test executions progress to completion without any action on your part.



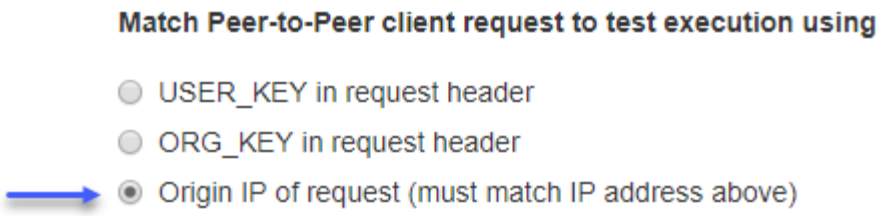
3. If both `USER_KEY` and `ORG_KEY` are absent in the request headers, then Touchstone tries to tie the request to the user's test execution by matching the actual IP address of the request to the IP address entered for the test system in Touchstone.

   This mechanism has the same limitation as ORG_KEY in that only one user within the organization can execute client-tests at a time.

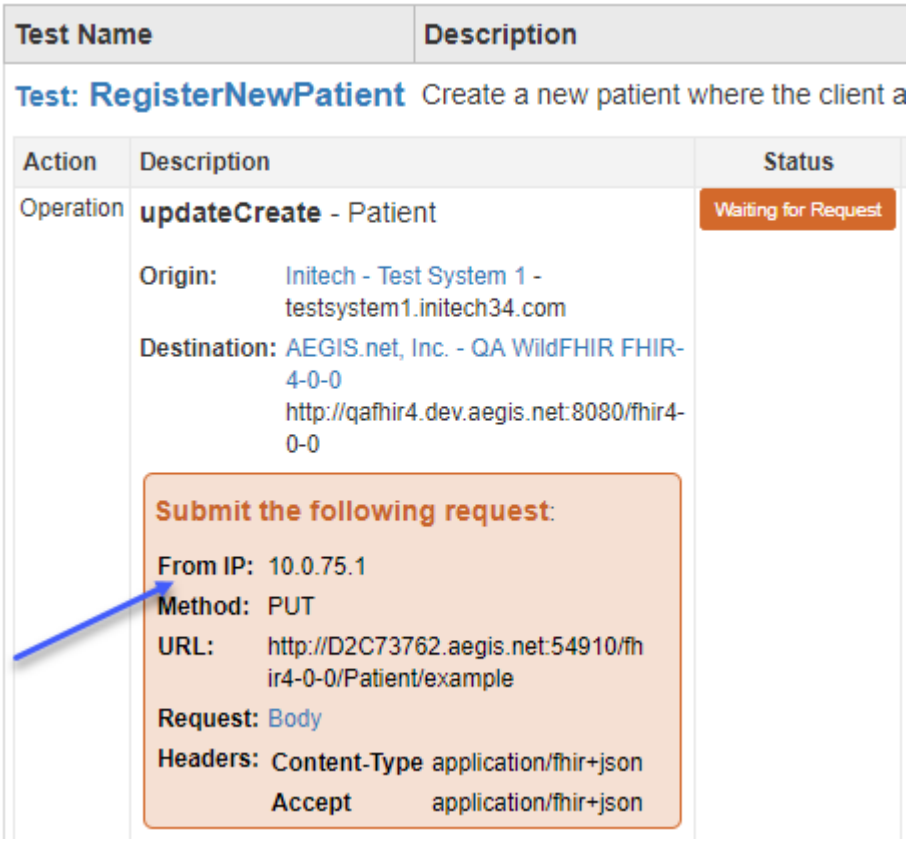
   This mechanism has a further disadvantage. If the IP address detected is unknown to Touchstone, then Touchstone will not be able to match the request message to the test execution and your operation execution will continue to stay in `Waiting for Request` status. To work around this, you will always need to keep your IP address up-to-date on the Test System screen:



   You can explicitly specify this mechanism as the matching mechanism when editing the client test system:



Doing so will allow Touchstone to tailor the instructions on the Test Script Execution's waiting step as follows:

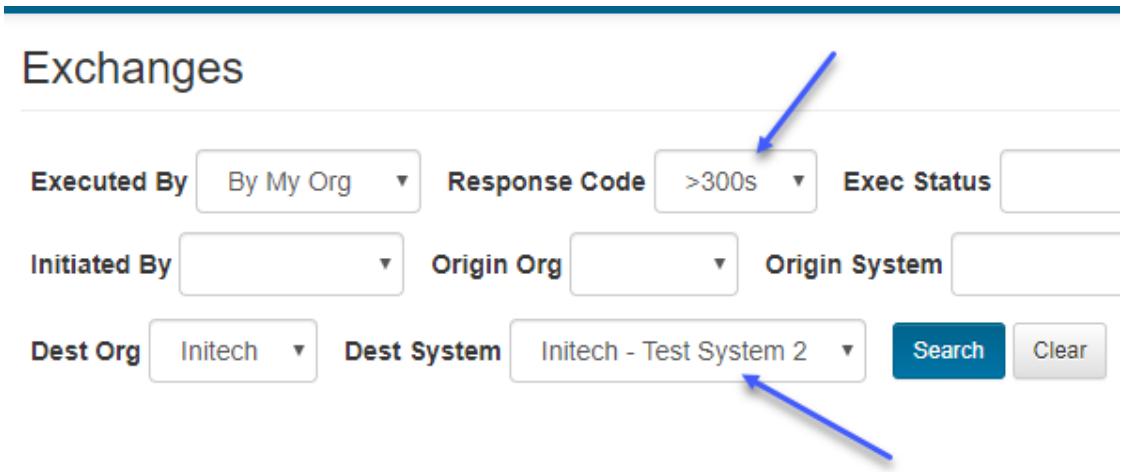


Specifying `Origin IP` as the matching mechanism on Edit Test System screen will also allow Touchstone to enforce constraints on concurrent launches of peer-to-peer test executions by users using the same client test system as the Origin system for the same test script and against the same destination.

8.3 Exchanges Screen

This screen is applicable to both Touchstone-initiated and Client-initiated tests. It shows all the message exchanges (interactions) that have taken place between test systems.

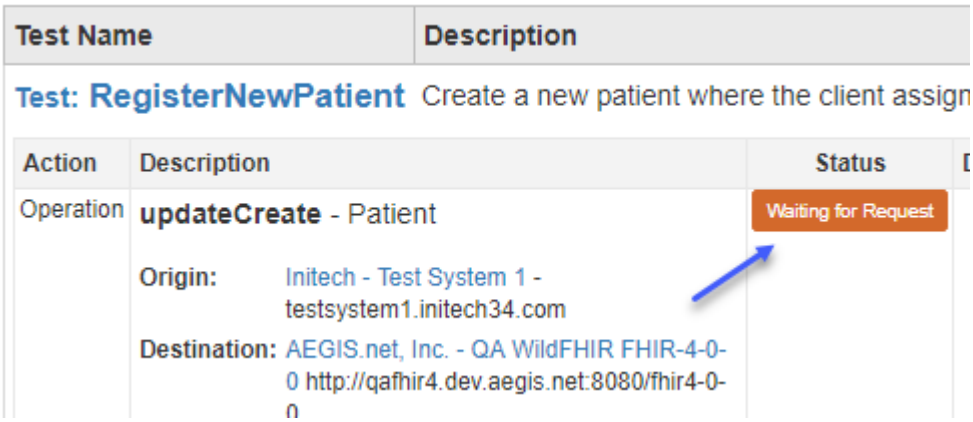
For example, you can specify your destination test system and a filter value of `>300`, `>400`, or `>500` for Response Code to see which exchanges caused your system to return these status codes:



8.3.1 Common Errors

Normally, when a client test system sends a request message to Touchstone Proxy, the Test Script Execution screen should progress from `Waiting for Request` status to `Running`, and then to `Waiting for Response`, and finally to `Passed` or `Failed`.

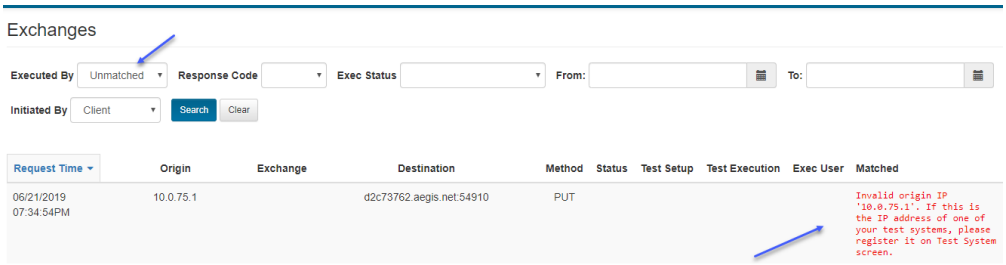
Sometimes, even though the request message has been sent to Touchstone Proxy, the Test Script Execution screen continues to stay in `Waiting for Request` status:



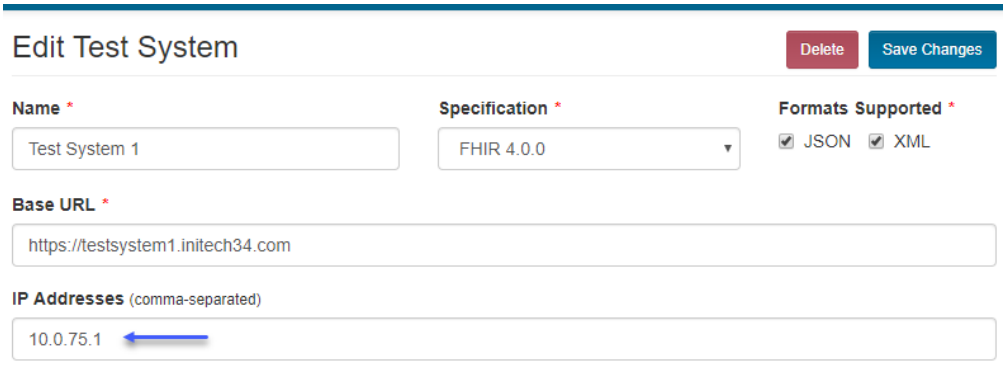
You can visit the Exchanges My Org and Exchanges Unmatched screens to look for an error message (in red) that would describe what happened.

Below are common errors that can be encountered:

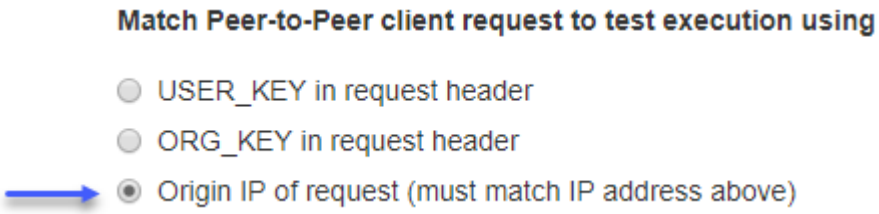
Invalid origin IP 'xxxxx'. If this is the IP address of one of your test systems, please register it on Test System screen.



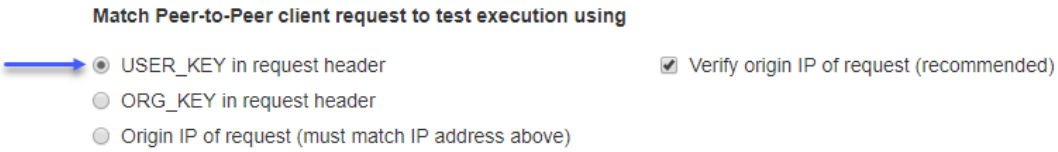
This error can take place if USER_KEY and ORG_KEY request headers were not specified in the request and the IP address that the request came from is unknown to Touchstone. You can correct the IP address on the Edit Test System screen and resubmit the request:



Additionally, you can specify `Origin IP` as the matching mechanism if the client test system cannot specify custom headers (USER_KEY or ORG_KEY) in the request.

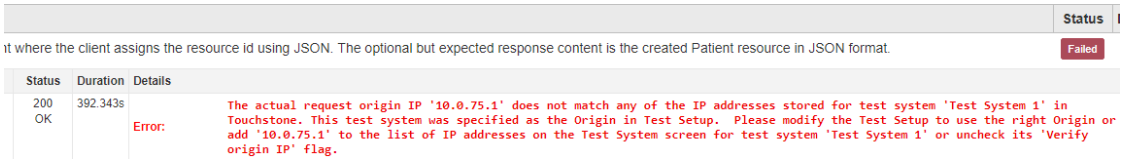


On the other hand, if the client test system can specify custom headers, then it is **highly recommended** to use the `USER_KEY` matching option and specify USER_KEY in the request header during submission.



The actual request origin IP 'xxxxxxx' does not match any of the IP addresses stored for test system 'Test System 1' in Touchstone. This test system was specified as the Origin in Test Setup. Please modify the Test Setup to use the right Origin or add 'xxxxxxx' to the list of IP addresses on the Test System screen for test system 'Test System 1' or uncheck its 'Verify origin IP' flag

This error can be encountered on the Test Script Execution screen after the request message has been successfully matched to the active test execution for the user:

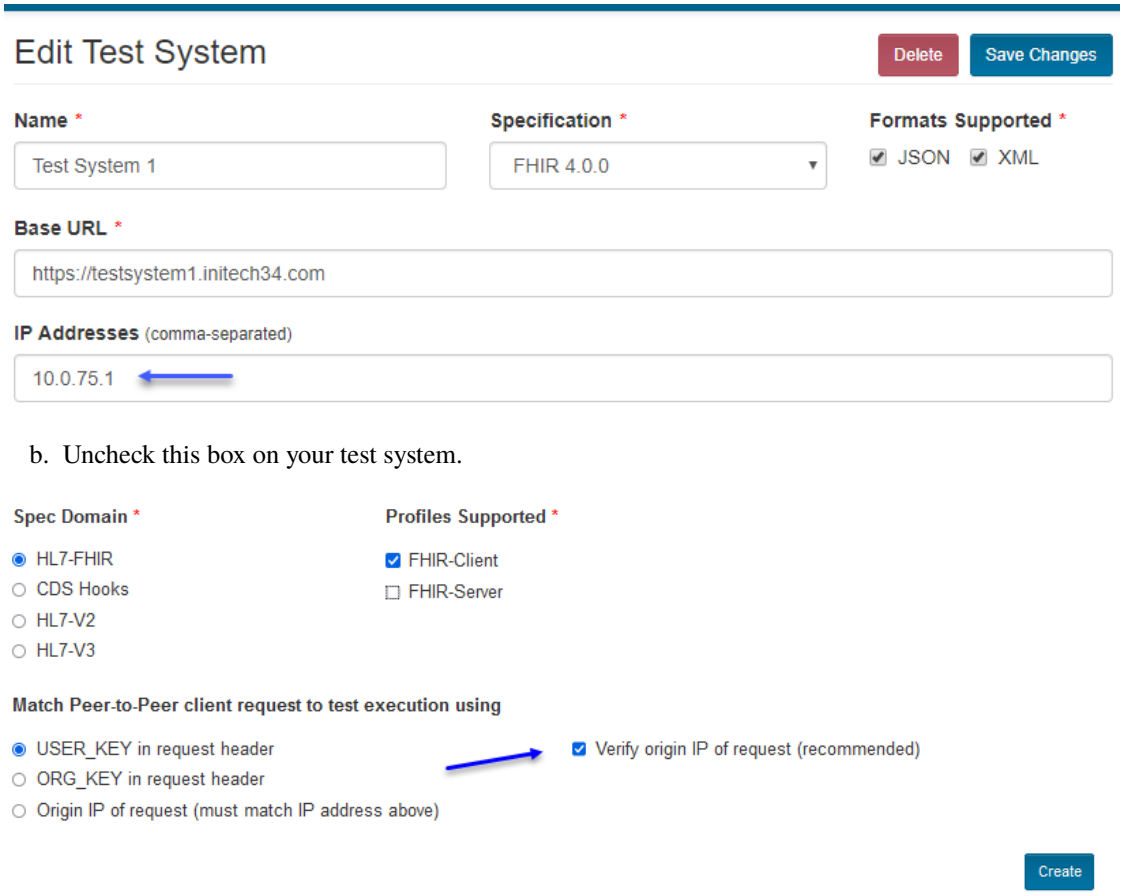


It can take place if either USER_KEY or ORG_KEY header was specified in the request and the active test execution was successfully matched but the additional origin IP verification failed.

For added security, Touchstone verifies that the the actual IP address of the request matches the IP address of your test system in Touchstone. You get the error above if they don't.

You can do **one** of the following to get around this error:

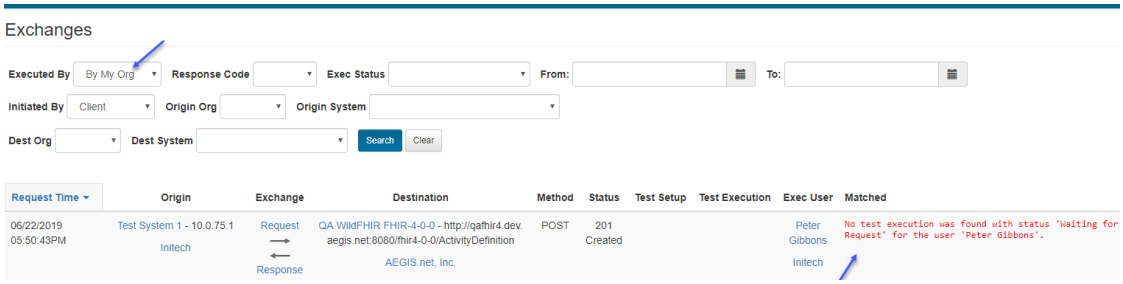
a. Add the IP address indicated in the message above (`10.0.75.1` for the case above) to the list of IP addresses of your test system:



b. Uncheck this box on your test system.

It is recommended to do option (a). If you do option (b), then other users can configure their test setups with your test system playing Origin and submit requests from their FHIR Clients. This will give the impression to end-users that the request came from your test system when in fact it came from another FHIR Client. If you don't have such a concern (because you have marked your client test system as accessible/executable only by 'My Org'), then option (b) is more convenient as you don't have to change the test system's IP Address in Touchstone every time the IP address of your FHIR Client changes.

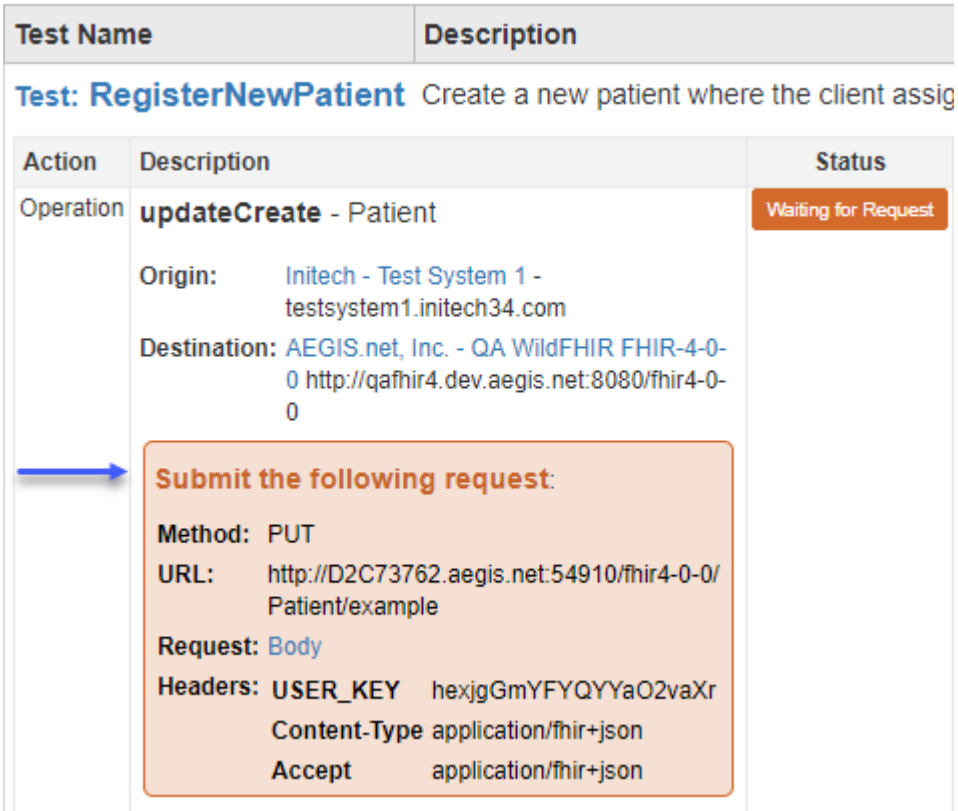
No test execution was found with status 'Waiting for Request' for the user



This error can take place if the user submits a request but has not launched a peer-to-peer testscript execution.

You can launch the testscript execution and wait for it to reach `Waiting for Request` status before submitting the request again. It is highly recommeded to follow the instructions on the waiting step of the

Test Script Execution screen as the URL port and other submission details could be different from what you are submitting:



8.4 FAQ

1. When executing a client-side test script, I have submitted what the system asked me to but my operation execution is still stuck on `Waiting for Request` You can visit the Exchanges My Org and Exchanges Unmatched screens to look for an error message (in red) that would describe what happened. See Common Errors for some of the errors.
2. When doing client-side testing, do I need to start all over if I submit the wrong USER_KEY accidently? It's best to click the `Execute Again` button on your Test Execution screen and start from the beginning. The target system is no longer in the state that your operation execution expects it to be in. Multiple submissions of the request message are still forwarded to the target system and processed. Resource ids and versions would change. So it might seem like nothing happened (because your operation execution still says `Waiting for Request`), behind the scenes message exchanges are taking place with the target system.

3. My peer to peer requests are not getting matched to my Waiting test but I cannot see the unmatched request in the Exchanges screen, why is this happening? Touchstone may not allow the request and not record the message exchange when 2 systems are communicating using HTTPS/TLS. When communicating using HTTPS/TLS, each system must "trust" the other. This is done via the SSL certificates presented by each system. This "trust" is established based on a system's presented SSL certificate being from a known and trusted root and/or intermediate CA (Certificate Authority).

   Touchstone utilizes the Java JDK trust store which contains the industry standard and known public CA root and intermediate SSL certificates. The Touchstone Proxy also utilizes the Java JDK trust store and follows the same "trust" logic when used as the intermediary between systems.

   If an organization's test system uses SSL certificate(s) issued from a different or newer CA, please contact Touchstone_Support@aegis.net.

4. Why does Touchstone not support restart from a certain point in the test script? It's painful (especially with client-side testing) to start from the beginning of the script. This is not a limitation in Touchstone. It's the way the test script specification was designed (see http://hl7.org/fhir/testing.html#execution). The Setup section is executed only once per test script. The Setup section is what enables repeatable and reliable test results. It cleans up the target system from previous resource updates/creates/etc. before launching the tests within the script. If you were to restart from a certain point in the script and not the beginning, the target system will not be in the state that the point you want to restart from expects it to be in. So you'll most likely get failed assertions and failed tests.

CHAPTER

NINE

TESTSCRIPT AUTHORING

9.1 FHIR TestScript

Touchstone test executions are controlled by the specification FHIR TestScript resource. Here are some of the high-level elements:

- *Setup* - Contains operations that seed the target system with the test data needed for the tests.
- *Tests* - Contains operations and assertions
  - *Operations* - HTTP-based requests against the target system.
  - *Assertions* - Verifications to ensure that the target system behaved as expected by the specification.

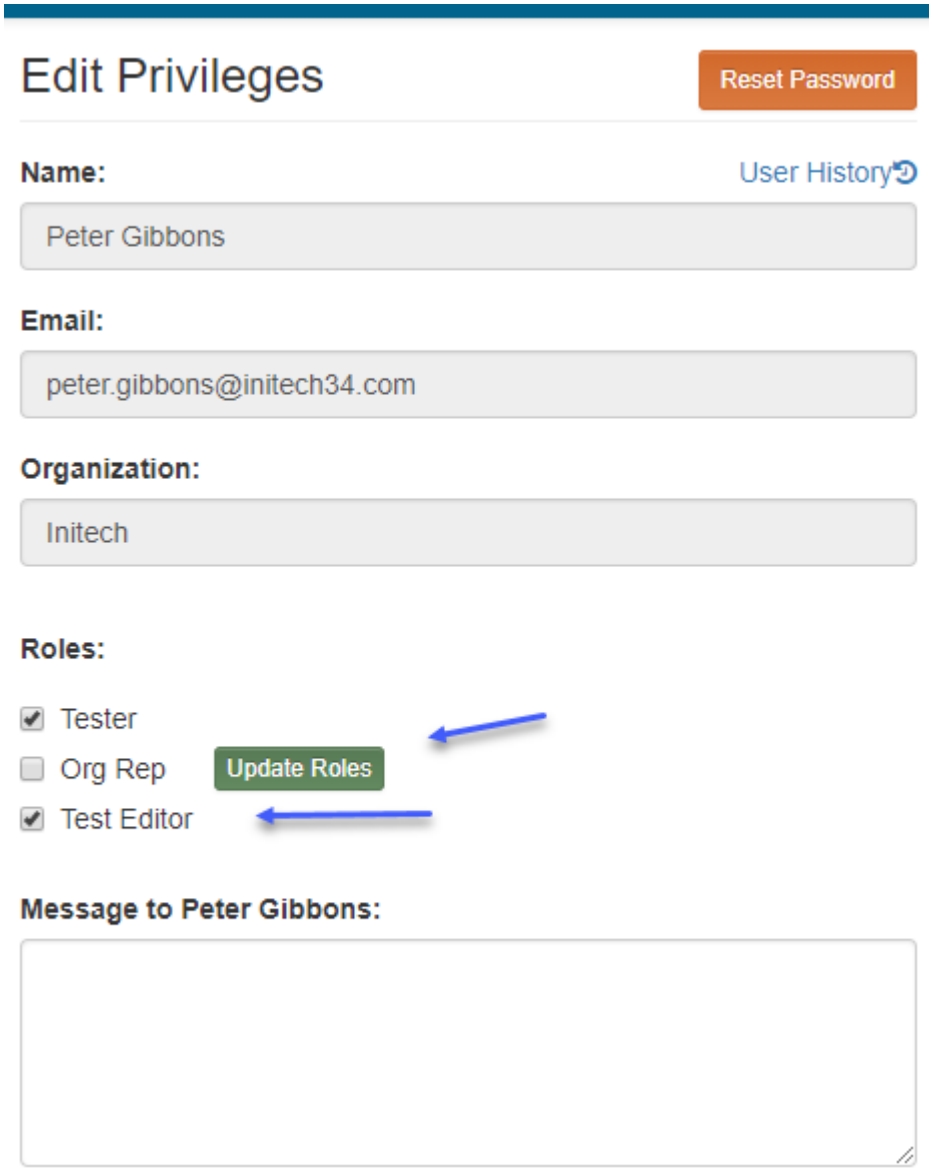
While TestScripts in Touchstone follow the FHIR TestScript, there are extra constraints enforced by Touchstone and extra extensions allowed through the Touchstone IG. Examples of such are:

- Operation.Resource is required for any Instance or Type Level REST Interactions where the Resource Type cannot be inferred by the source Fixture.
- Rules, Rulesets, and the asserts that use them are all extensions for R4 testScipts defined in the Touchstone IG.

There are many more elements in the FHIR TestScript. Please refer to Testing FHIR and TestScript for details on how the TestScript works.

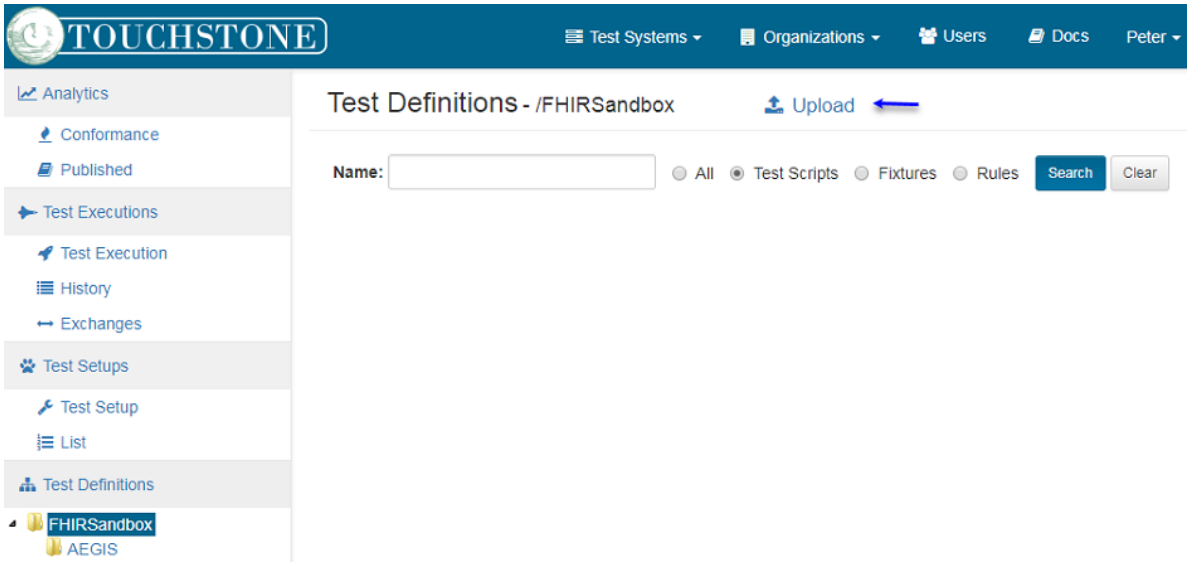
9.2 Upload on UI

Organizations that are subscribed at the Starter level and above can upload testscripts to Touchstone using the web interface. Org Reps within these organizations can decide which users within their organization can upload test scripts:

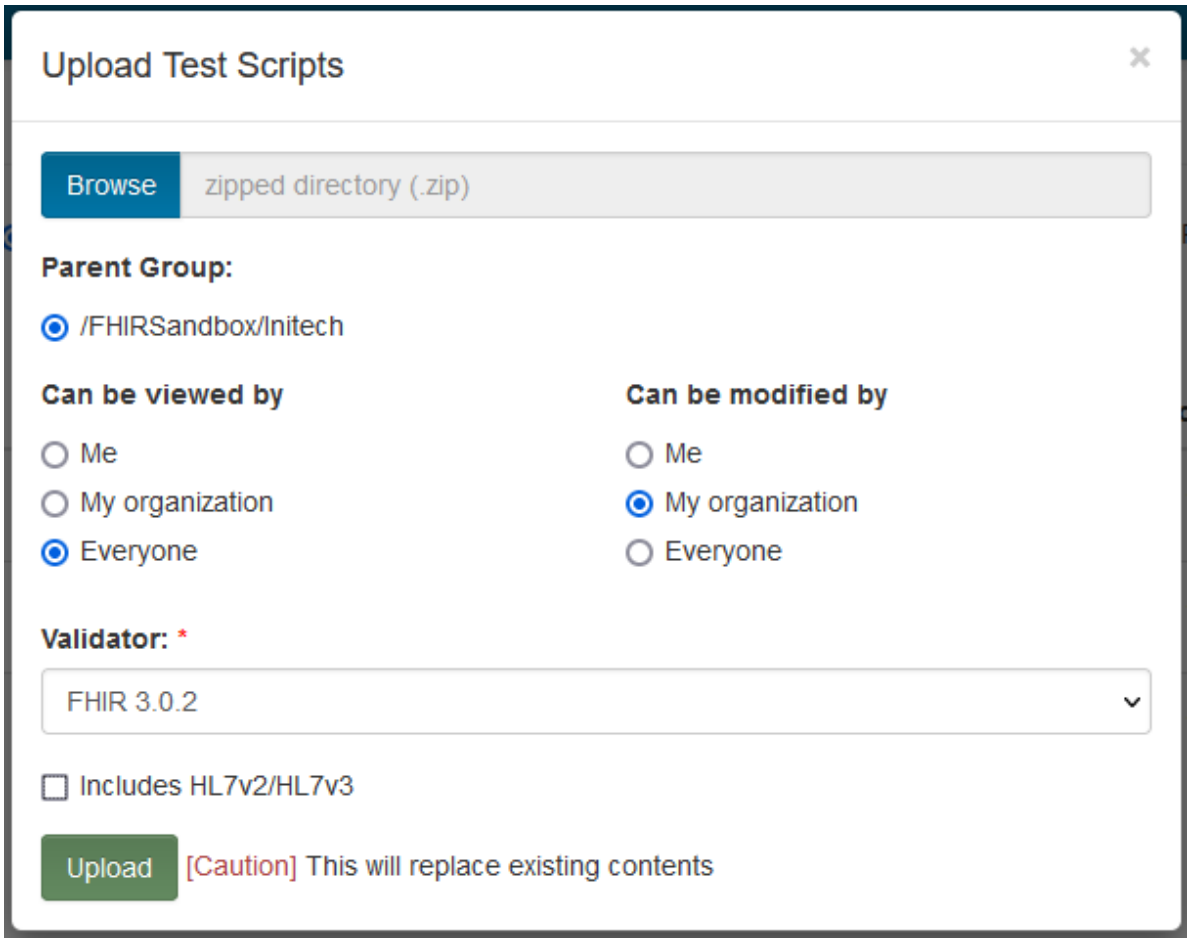


Folders (containing test scripts, fixtures, rules, etc.) can be uploaded as a zip file on the Test Definitions screen. The folder will land under your organization name in the FHIRSandbox folder.

1. Click on the `Upload` link on the Test Definitions screen:



2. Browse to the zip file containing the test scripts and fixtures that you want to upload:



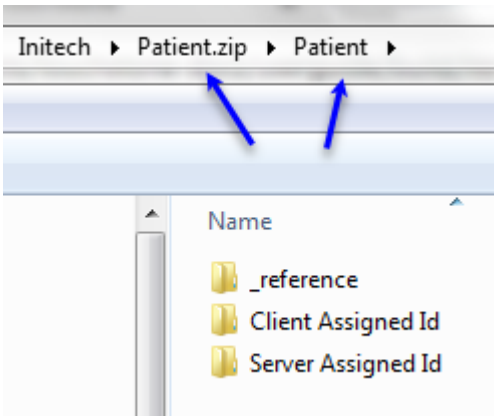
- Browse – Point this to the zip file that you want to upload. It should contain the test scripts and the referenced fixtures, rules, etc.
- Parent Group – The destination of the zipped folder. There will be one option if this is the first time you're uploading or if you're situated at the root of the hierarchy.

- Can be viewed by – If `Me` or `My Organization` is selected, then users outside your organization will not be able to see the test group or execute the test scripts within it. If `Me` is selected, then even other users within your organization will be unable to see the test group.
- Can be modified by – If `Me` or `My Organization` is selected, then users outside your organization cannot overwrite the uploaded test group. If `Me` is selected, then even other users within your organization will be unable to overwrite the test group.
- Validator – This is the default Validator that the test scripts will be testing against. The Validators are listed by what base FHIR Specification they depend on. All resources including the test scripts and fixtures within this uploaded test group has to abide by the constraints imposed by the specification version. Additionally, the workflow logic being tested by a test script will vary from one version to the next. For additional details, refer to Validators.
- Includes HL7v2/HL7v3 – Check this box if the test scripts you are uploading are `HL7v2` or `HL7v3` test scripts. You will be presented with another drop down to choose the V2 or V3 validator to be used during the execution of the test scripts. The chosen validator for the main **Validator** option will be used for validation when the test scripts are uploaded. More info can be found at HL7v2 Validation and HL7v3 Validation.

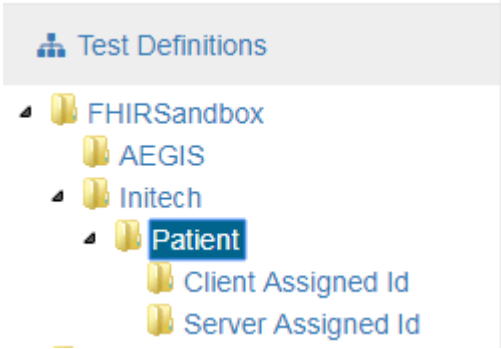
For users whose organizations are part of Org Groups, there is an additional option of `My Org Groups` offered in the `Can be viewed By` and `Can be modified By` option list. If you chose e.g. `Org Group A` in `Can be viewed by`, then only users of organizations that are part of Org Group A will be able to see the test group and execute test scripts in that test group. If you chose e.g. `Org Group A` in `Can be modified by`, then those users will also be able to modify the test group. For additional details, refer to Test Definition access.

Zip with one folder

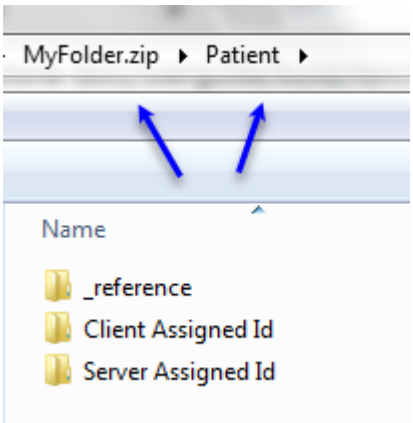
- If the zip file contains one folder inside of it and that folder name matches the name of the zip file, then Touchstone will **not** create a duplicate sub-folder. The zipped folder will land under the org name:



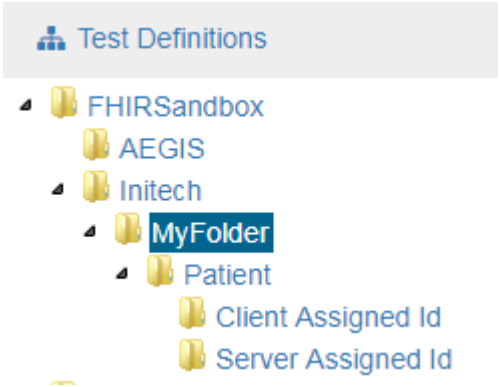
Notice that after the upload, there is only one `Patient` folder under Initech i.e. We don't have `Patient/Patient` under Initech:



- If the zip file contains one folder inside of it and that folder name does not match the name of the zip file, then Touchstone will use the zip file name as the destination folder and the contained folder becomes a sub-folder under that destination folder:



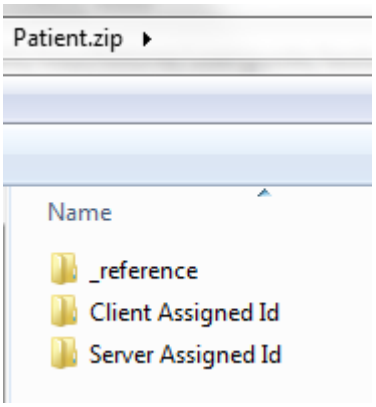
Notice that after the upload, the destination folder is the same as the zip file name:

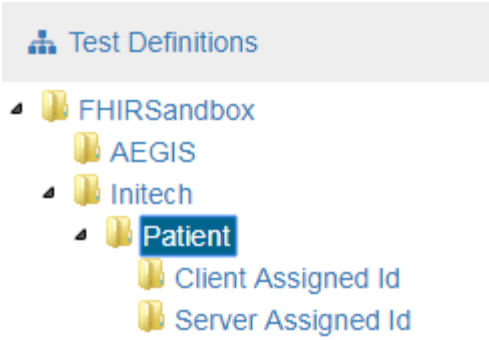


Zip with multiple folders

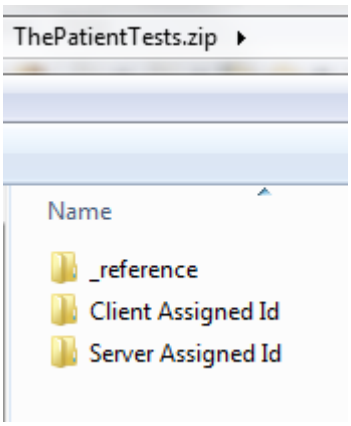
If the zip file contains multiple folders, then Touchstone will use the zip file name as the destination folder and the contained folders become sub-folders under that destination folder:

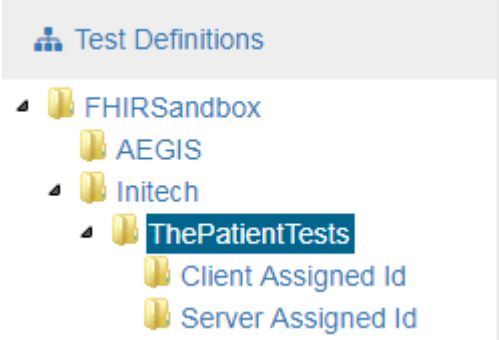
- If the zip file is named `Patient.zip`, then destination folder will be `Patient`:





- If the zip file is named `ThePatientTests.zip`, then destination folder will be `ThePatientTests`:





Note: When uploading testscripts, two testscripts cannot have the same name but different file extensions (.xml or .json). They must have unique names for upload.

9.3 TestScript Editor

The TestScript Editor is an Eclipse-based desktop development environment. It provides a comprehensive suite of development tools for creating, managing and publishing FHIR TestScript resources. It is designed to simplify test script development and accommodate a large number of users, ranging from beginners to experts.

The TestScript Editor can be used to:

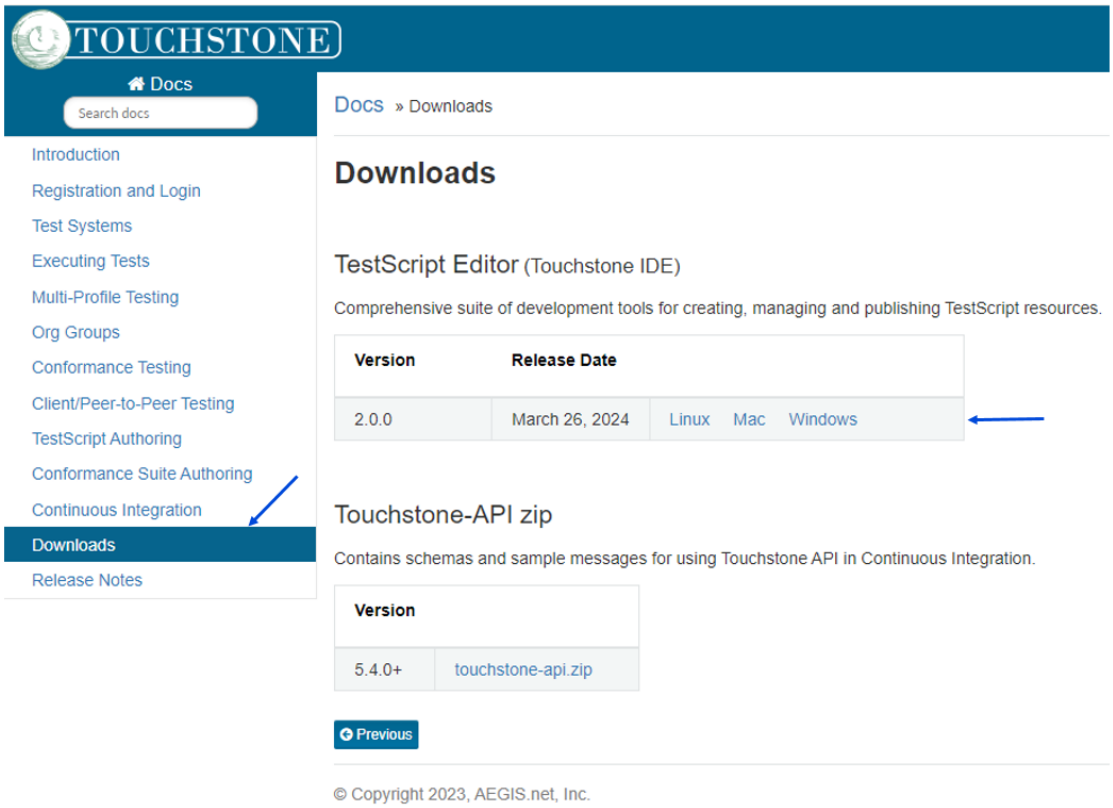
- Upload Test Groups and TestScript resources to Touchstone.
- Upload Test Groups to and download them from Simplifier.
- Manage TestScript resources by integrating with Version Control systems such as SVN, GIT etc.

The TestScript Editor leverages the following built-in Eclipse editors:

- XML Editor (*.xml)
- JSON Editor (*.json)
- Groovy Editor (*.groovy)
- Text Editor (*.txt)
- Java Editor (*.java)

9.3.1 Download

You can download the TestScript Editor by going to Docs > Downloads and selecting the appropriate platform:



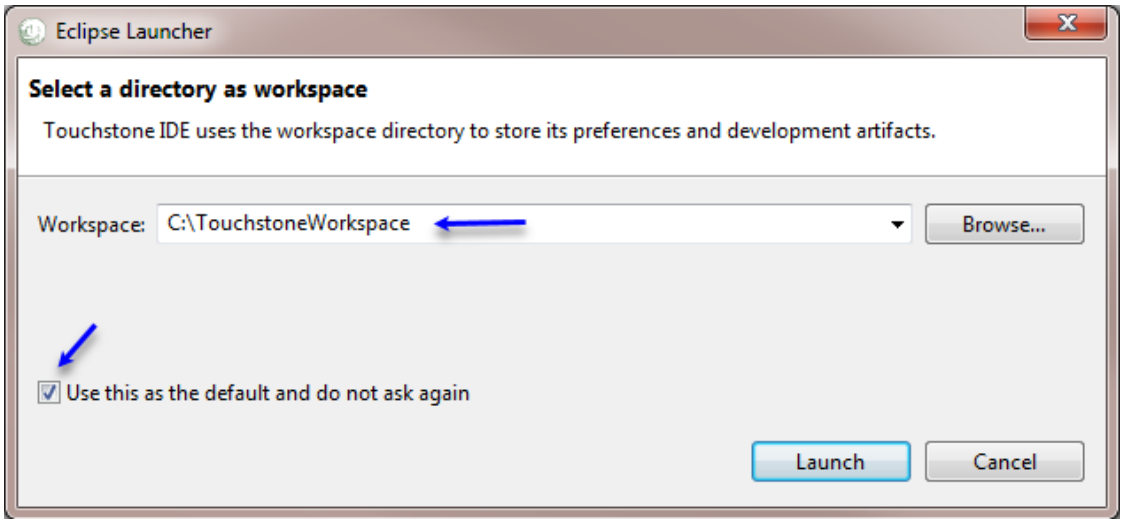
Once the download is complete, unzip the zip file to a directory of your choice.

9.3.2 Start TestScript Editor

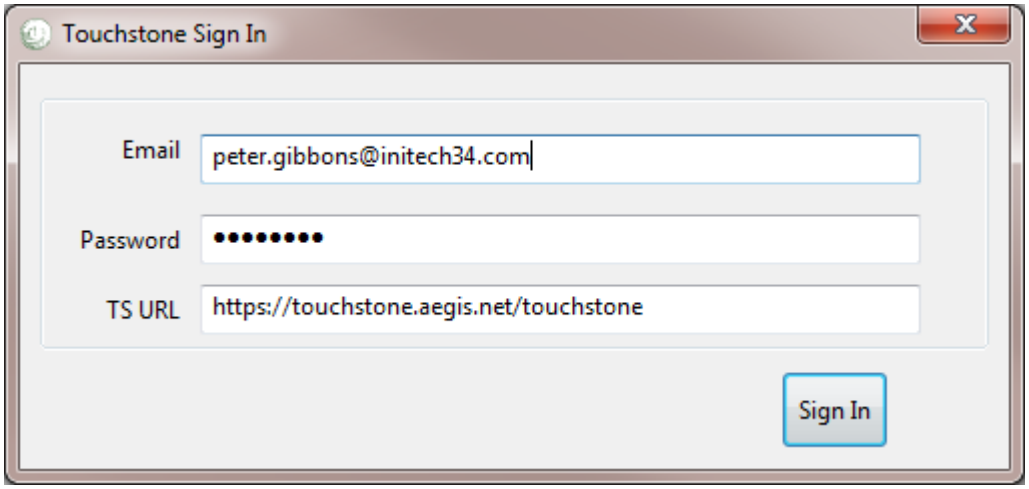
To bring up the TestScript Editor, run/open the `Touchstone IDE` file in the unzipped directory. It will be `Touchstone IDE.exe` on Windows, `Touchstone IDE.app` on Mac, and `Touchstone IDE` on Linux.

Feel free to create a shortcut to this file.

The first time, the editor is brought up, it will prompt you for the location of your workspace. You can select any location. The workspace location can be different from the location of your TestScript resources.



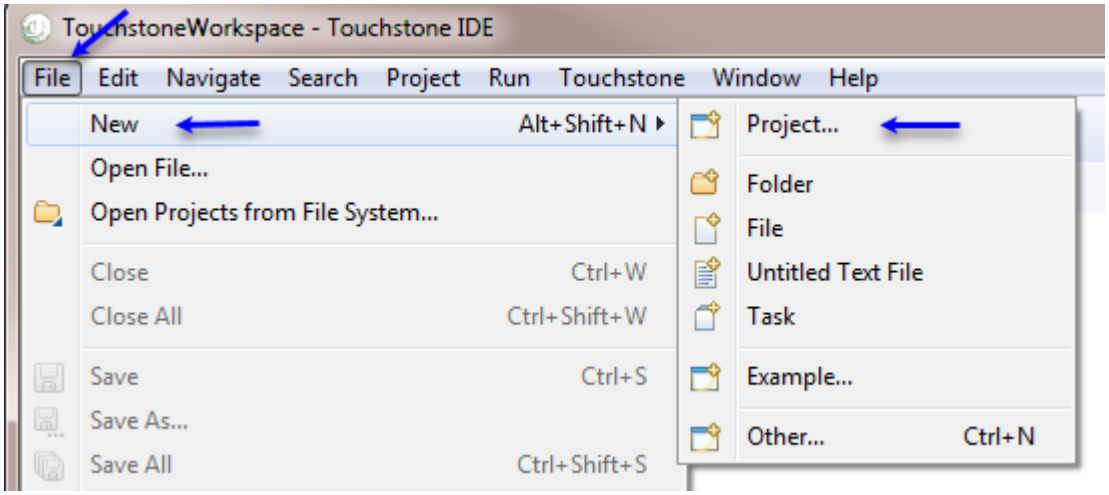
Use of TestScript Editor requires Enterprise-Level subscription. You must use the same credentials you use to sign into Touchstone web site:



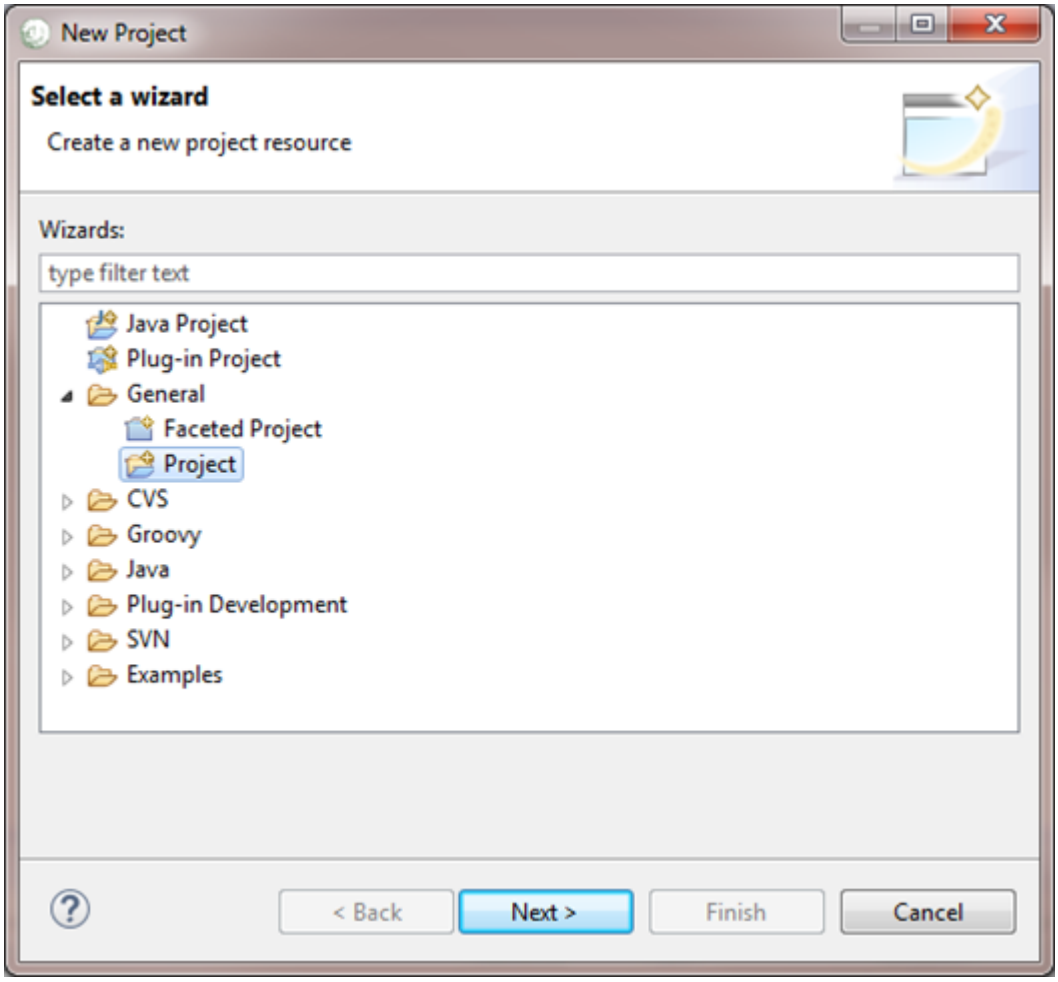
9.3.3 Create TestScript Project

You can start by creating a simple project as follows:

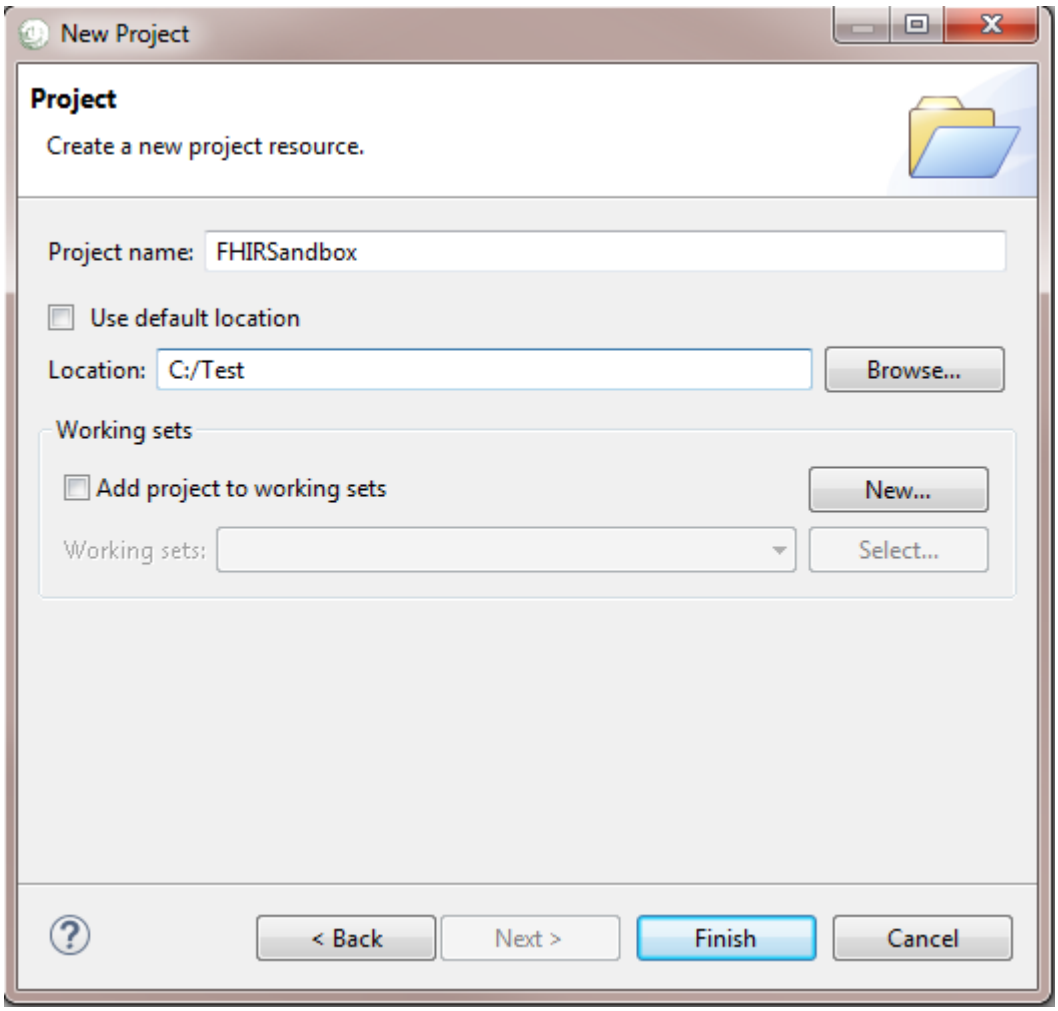
1. From the menu bar, select `File > New > Project...`



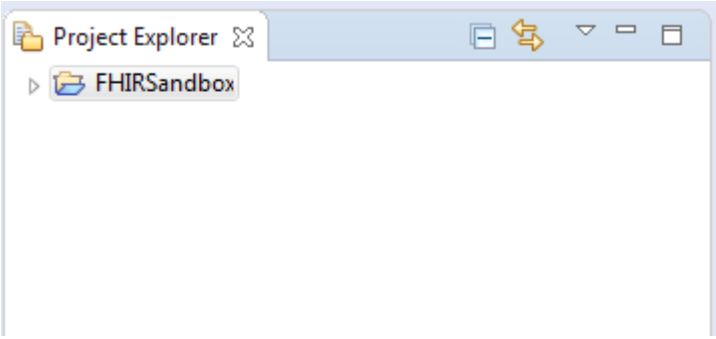
2. In the New Project wizard, select `General > Project` and click Next:



3. In the `Project` name field, type the name of your new project e.g. **FHIRSandbox**.
4. Leave the box checked to use the default location or uncheck the default location and select a new location for your new project. Click `Finish` when you are done.



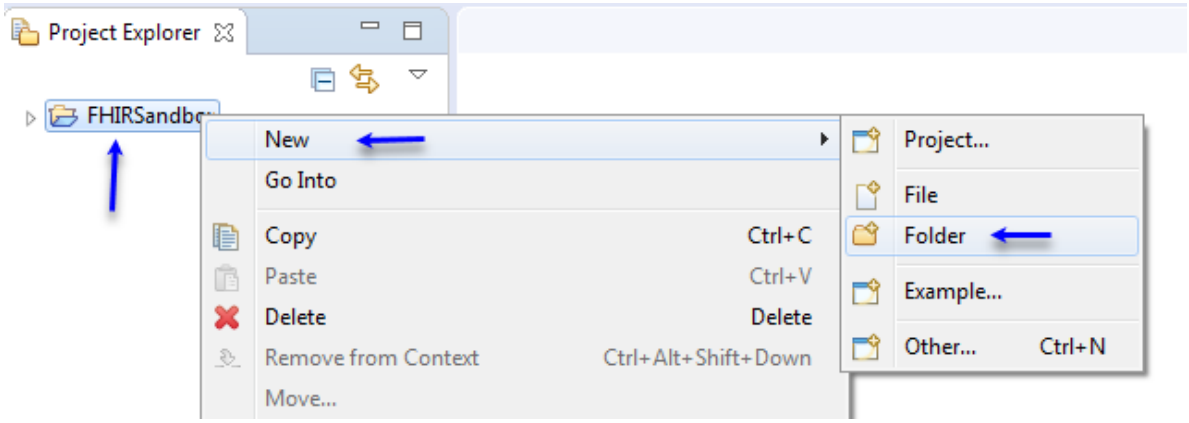
The navigation view will now contain the FHIRSandbox project you just created.



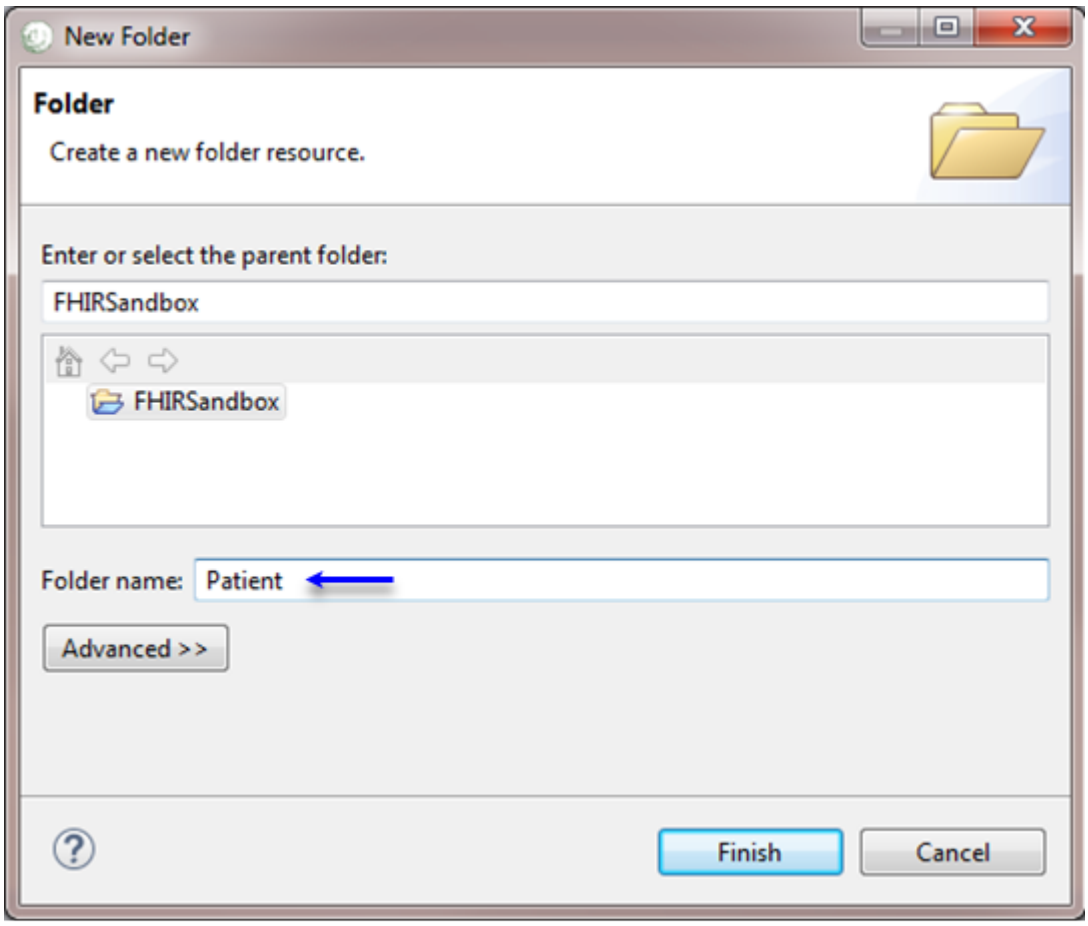
9.3.4 Create a Test Group

Test Groups in Touchstone are represented by folders in TestScript Editor. You can create a new folder using the Project Explorer view's popup menu:

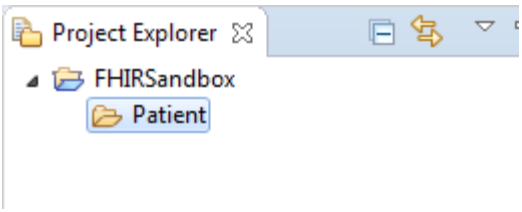
1. Activate the Project Explorer view and select the project **FHIRSandbox** (the first project we created in the Project Explorer view). Right click on **FHIRSandbox** folder and select `New > Folder`.



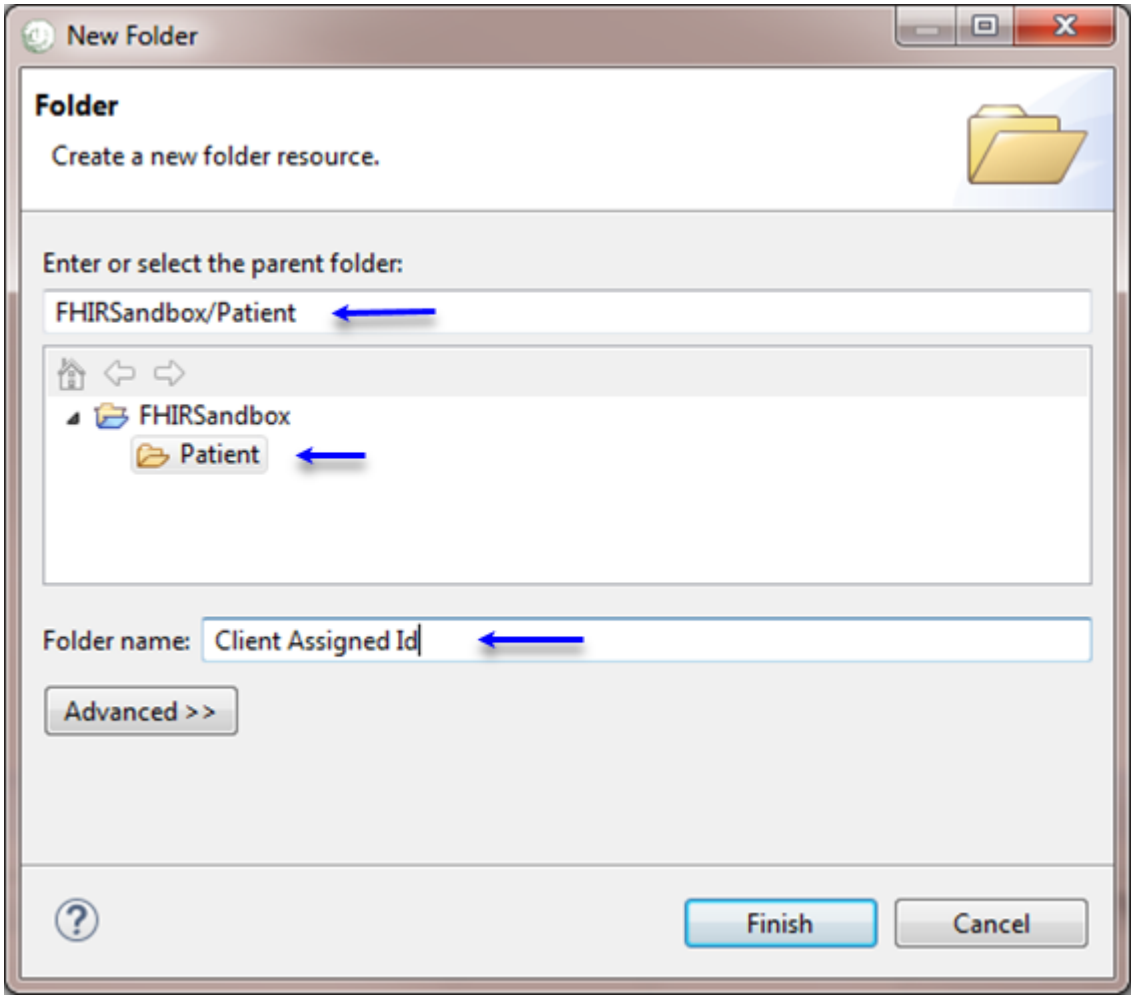
2. In the `Folder name` field, type a unique name for your new folder, e.g. **Patient**:



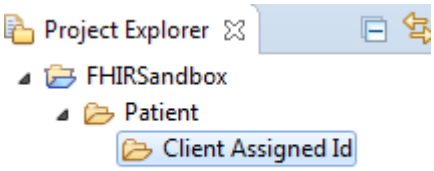
3. Click `Finish` when you are done. The Project Explorer view will update to show your newly created folder:



4. Repeat the folder creation steps. This time, create **Client Assigned Id** under the **Patient** folder:



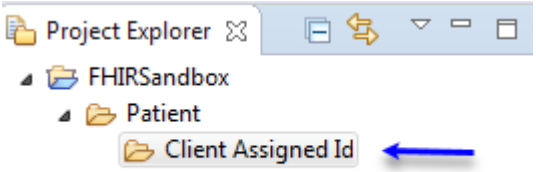
5. The Project Explorer view will update to show your newly created folder:



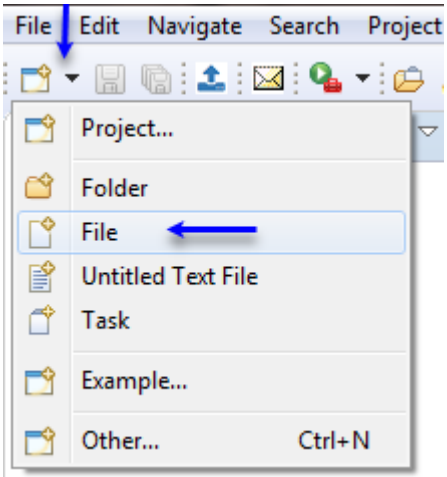
9.3.5 Create a TestScript

You can create a TestScript by following the steps below:

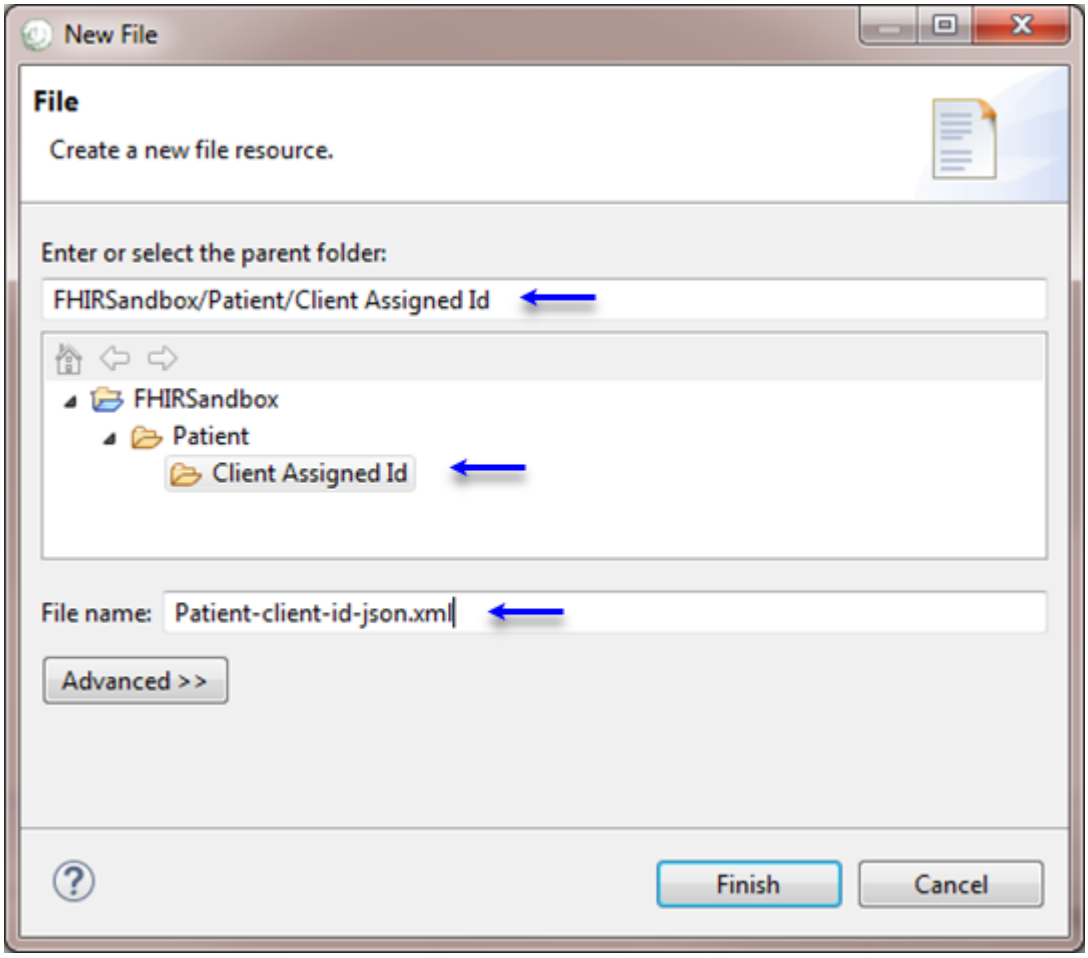
1. Select the folder **Client Assigned Id** in one of the navigation views.



2. In the toolbar, activate the drop-down menu on the New Wizard button and select `File`:



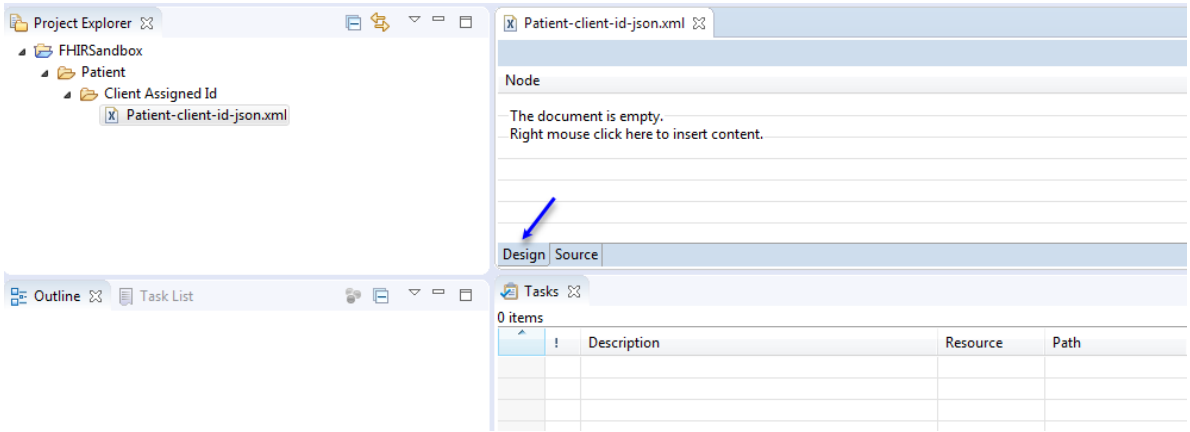
3. Enter the file name including the extension e.g. **Patient-client-id-json.xml**:



The file extension is **".xml"** indicating that the source will be written in XML format.

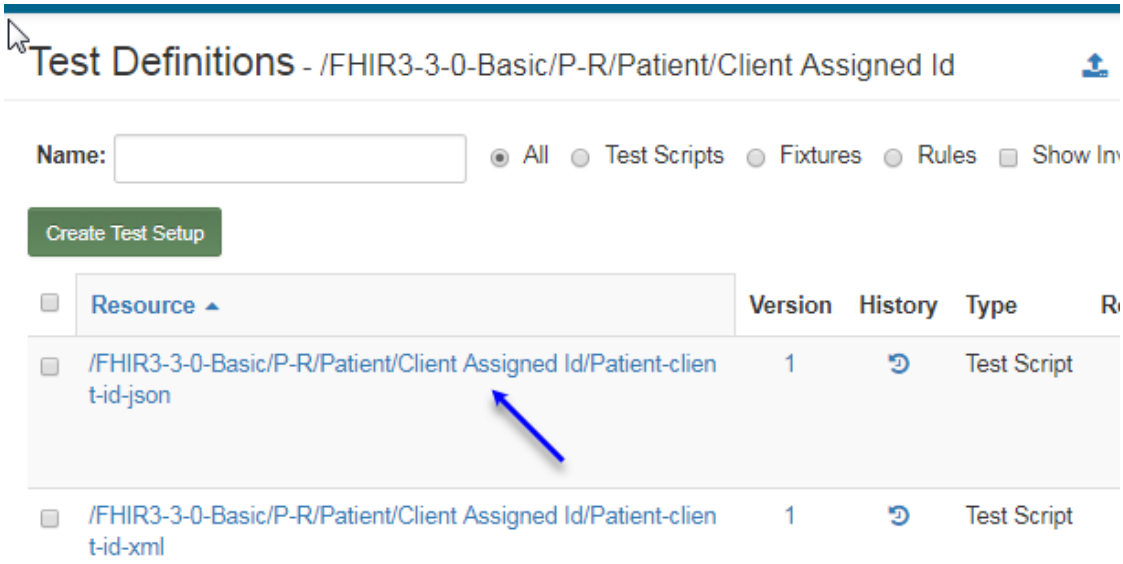
The file name has **"-json"** indicating that the TestScript will be used to call REST operations in JSON format. You don't have to follow this convention.

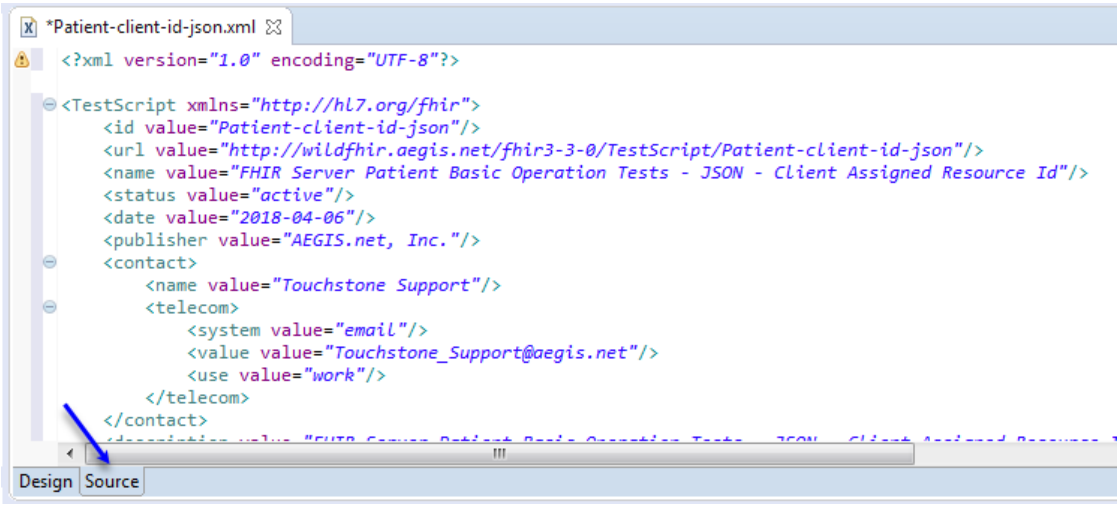
4. Click **Finish** when you are done.
5. The Workbench has an editor capable of editing XML files. The editor is automatically opened on the newly created file:



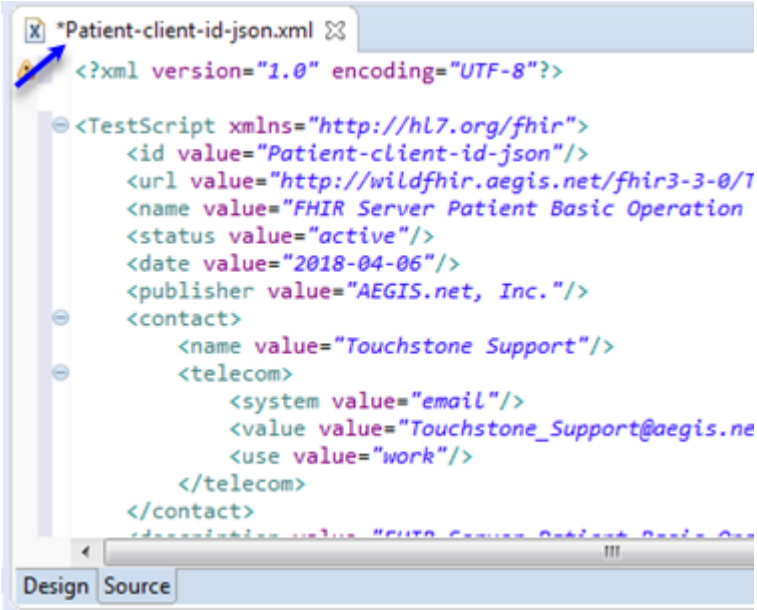
6. Select the `Source` tab and copy into it the contents of the Patient test scripts. Current up-to-date scripts may be

obtained from Test Definitions in Touchstone. You can download these example Patient test scripts and fixtures: Example Patient Test



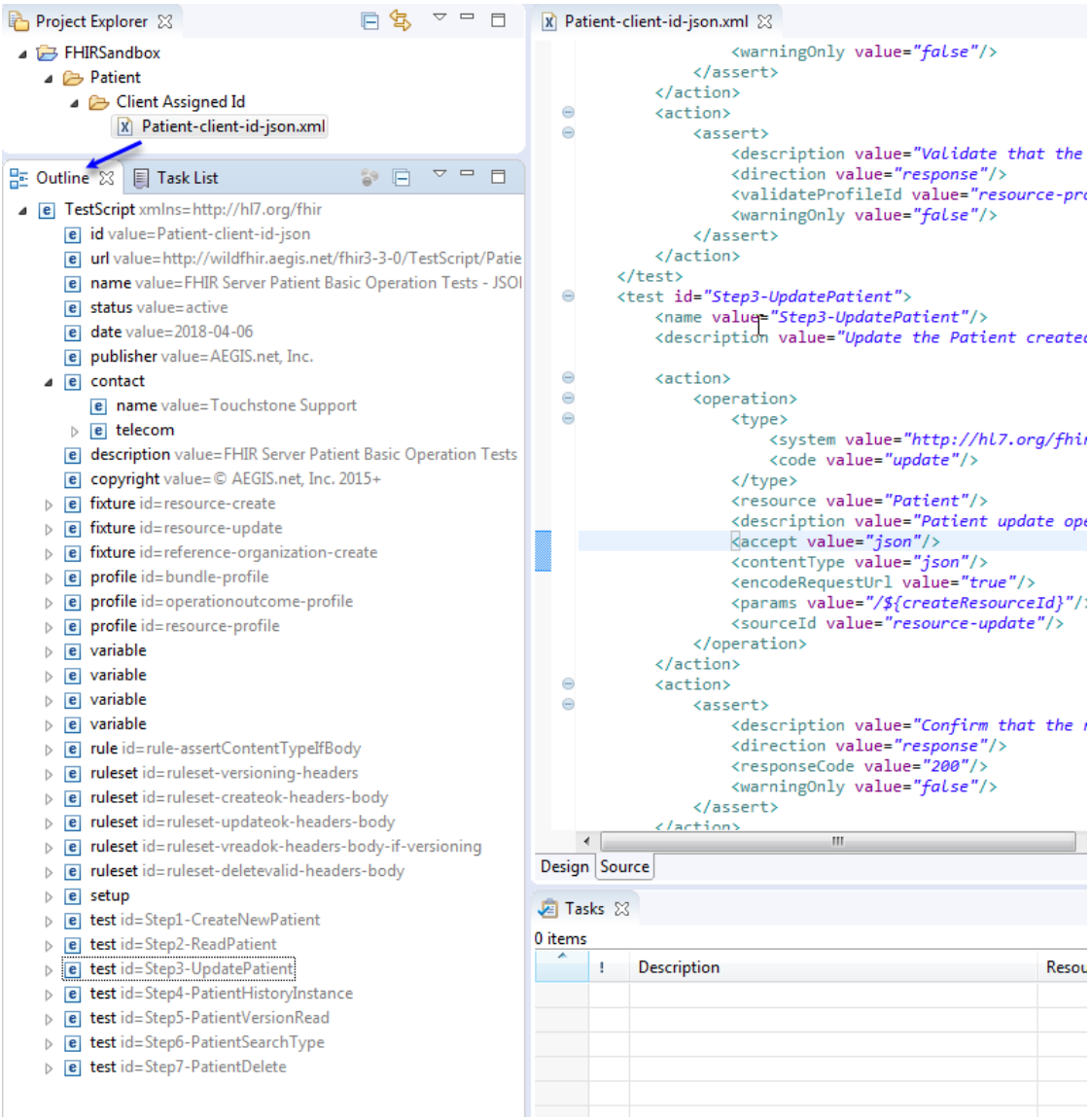


Notice that the editor tab has an asterisk (*) at the left of the filename. The asterisk indicates that the editor has unsaved changes.

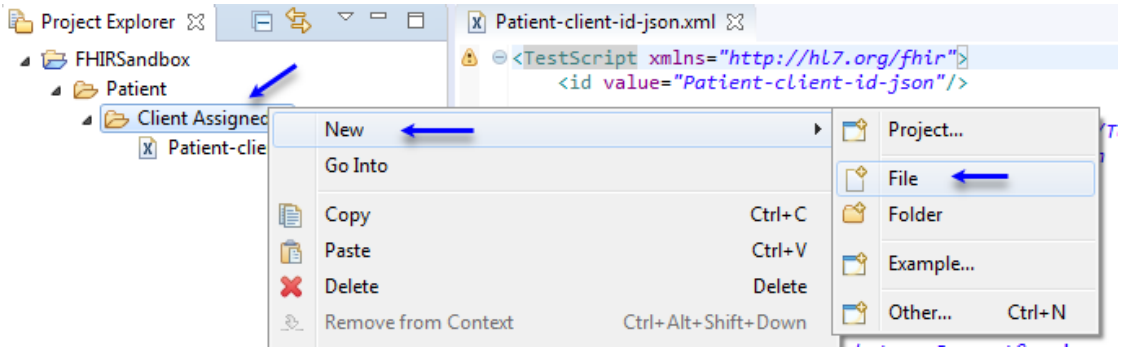


7. In the Workbench window's toolbar, click the Save button to save your work.

The Outline panel shows the high-level structure of the TestScript file:



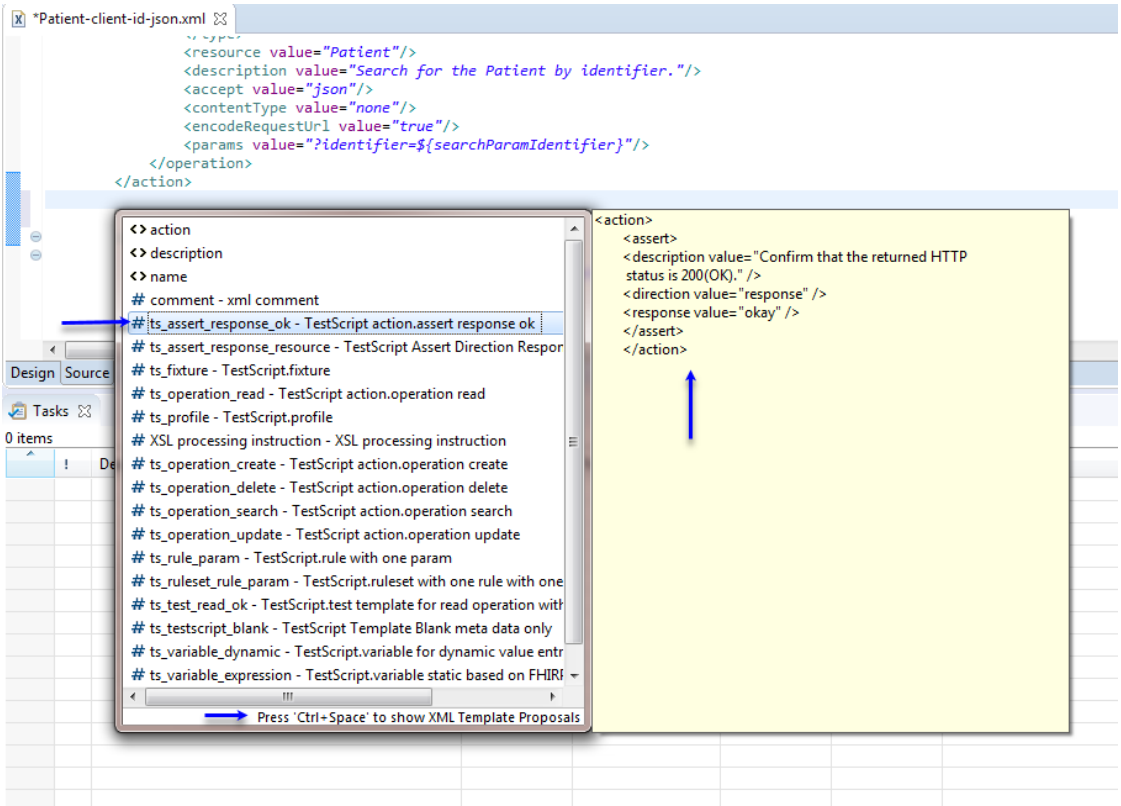
Note that a new file can also be created by using the context menu which you can get to by right-clicking the folder:



9.3.6 Code Completion

The TestScript editor provides pre-defined easy-to-use XML templates which increase reusability and speed up development of test scripts.

To access the templates, open any XML file and enter **CTRL + SPACE**. This displays the predefined list of templates that you can select for code completion:



You can use the provided templates, customize them, or create your own templates.

For example, you can work on a group of testscripts that should all contain a rule/ruleset element with a specific definition. Create a template that contains the tags for that rule/ruleset, including the appropriate attributes and attribute values for each tag. You can copy and paste the tags from a structured text editor into the template's Pattern field. Then select the name of the template from a content assist proposal list whenever you want to insert your custom rule/ruleset into an XML file.

To create a new XML template, you can take the following steps:

1. Click **Window > Preferences** and select **XML > XML Files > Templates**.
2. Click New if you want to create a completely new template.
3. Supply a new template Name and Description.
4. Specify the Context for the template. This is the context in which the template is available in the proposal list when content assist is requested.
5. Specify the Pattern for your template using the appropriate tags, attributes, or attribute values to be inserted by content assist.
6. If you want to insert a variable, click the Insert Variable button and select the variable to be inserted. For example, the date variable indicates the current date will be inserted.

7. Click OK and then Apply to save your changes.

You can edit, remove, import, or export a template by using the same preferences page. If you have modified a default template, you can restore it to its default value. You can also restore a removed template if you have not exited from the workbench since it was removed.

If you have a template that you do not want to remove but you no longer want it to appear in the content assist list, clear its check box in the table on the Templates preferences page.

9.3.7 Touchstone Integration

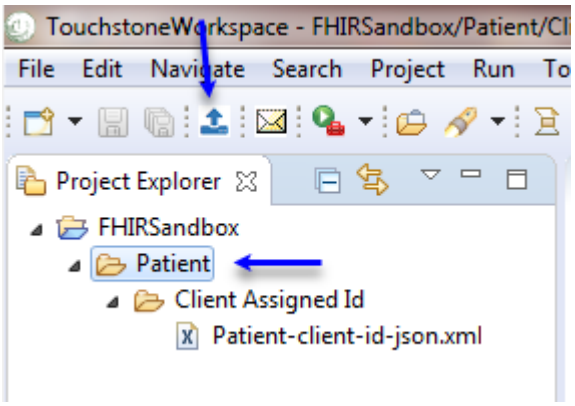
Note:

- You need to be connected to internet
- You need Starter or higher subscription level to upload test scripts to Touchstone.

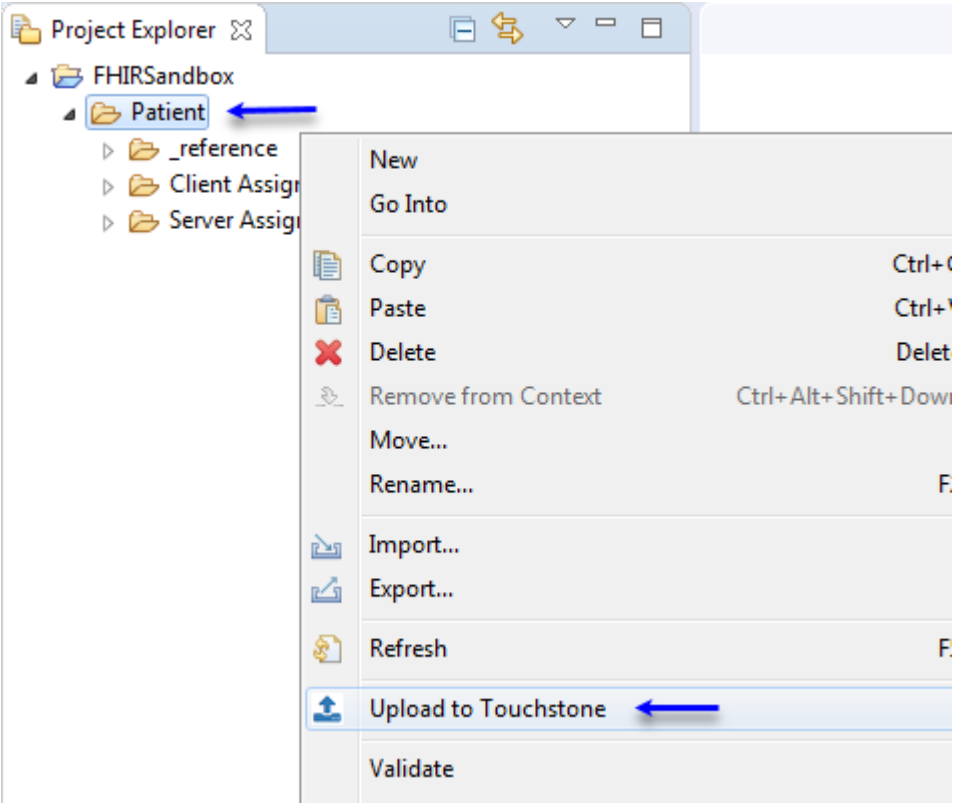
9.3.7.1 Upload to Touchstone

1. Upload to Touchstone can be initiated in one of the following ways:

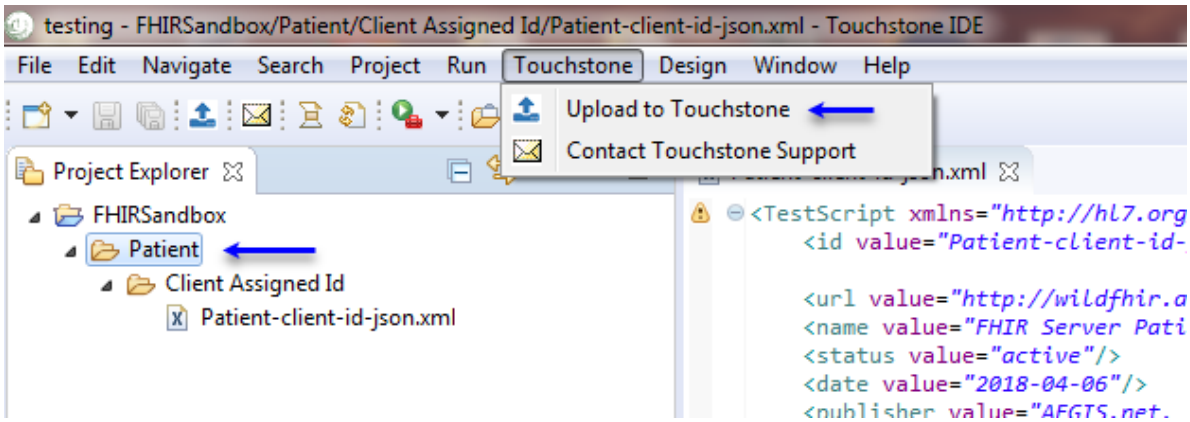
   1.1. *Toolbar upload button* - In the toolbar, click the upload icon button. The file or folder selected in the Project Explorer view will be uploaded.



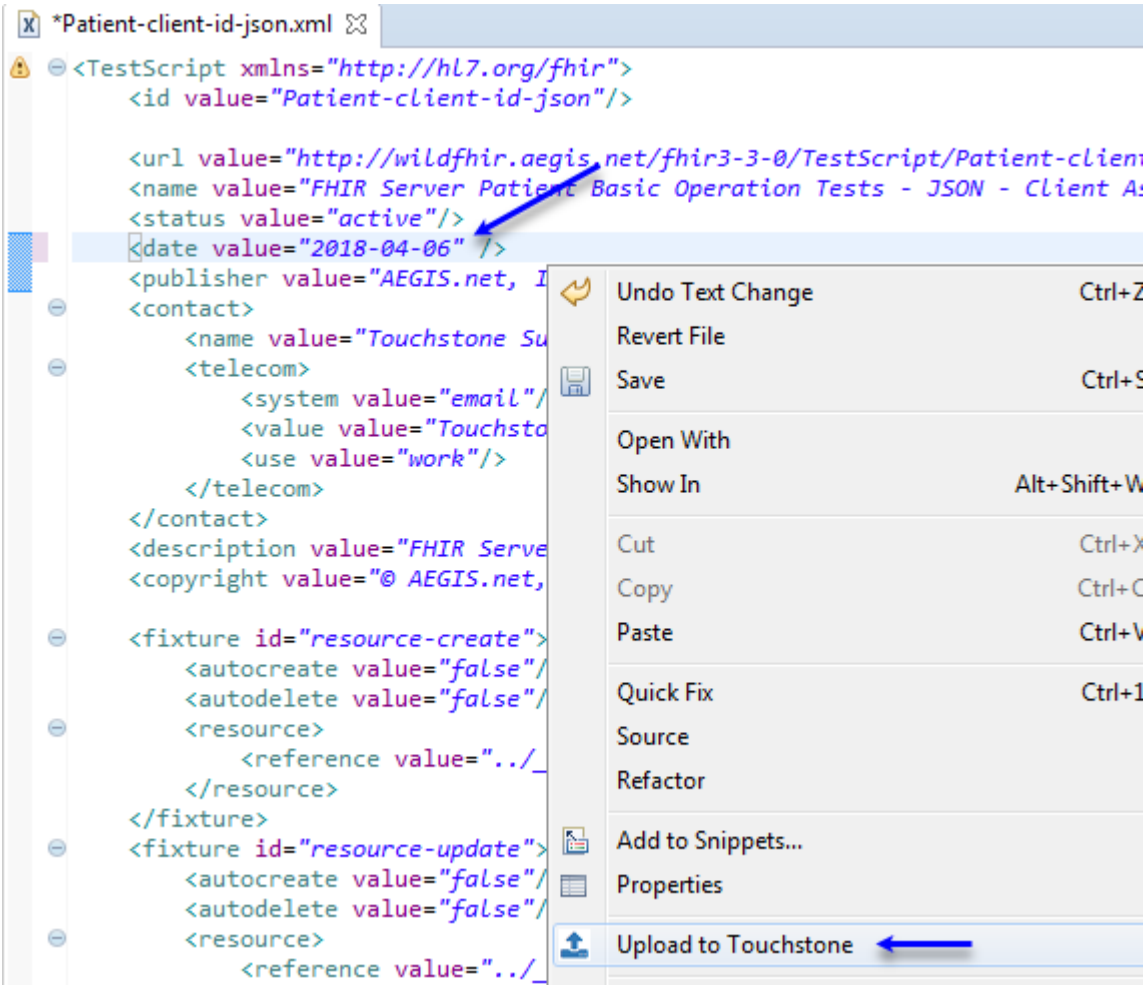
   1.2. *Popup menu* - In Project Explorer, right-click a folder or file and click on **Upload to Touchstone**:



1.3. *Touchstone menu* - Click on **Touchstone > Upload to Touchstone** on the menu bar. The file or folder selected in the Project Explorer view will be uploaded.

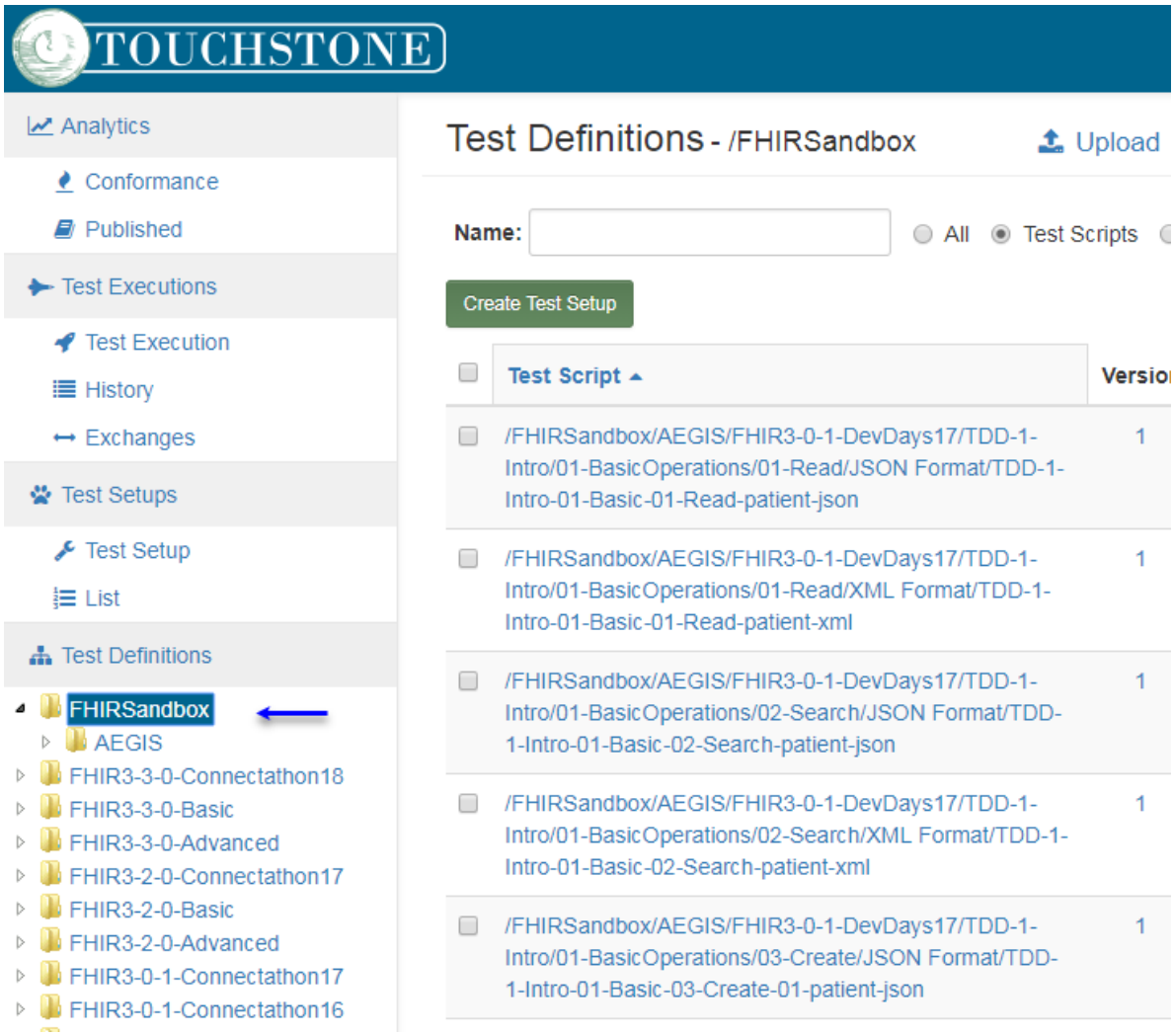


1.4. *Editor popup menu* - In the editor, right-click the content pane and click on **Upload to Touchstone**:

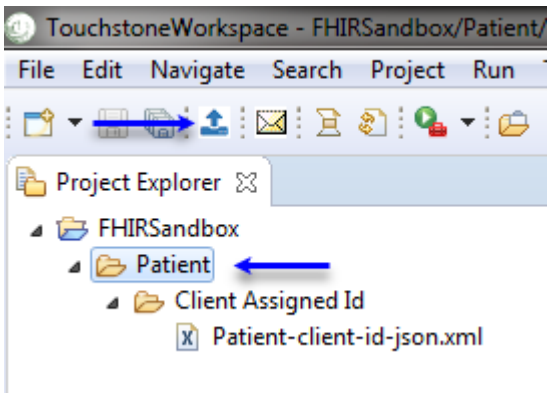


Warning: You should upload the root folder first before attempting to upload sub-folders and files.

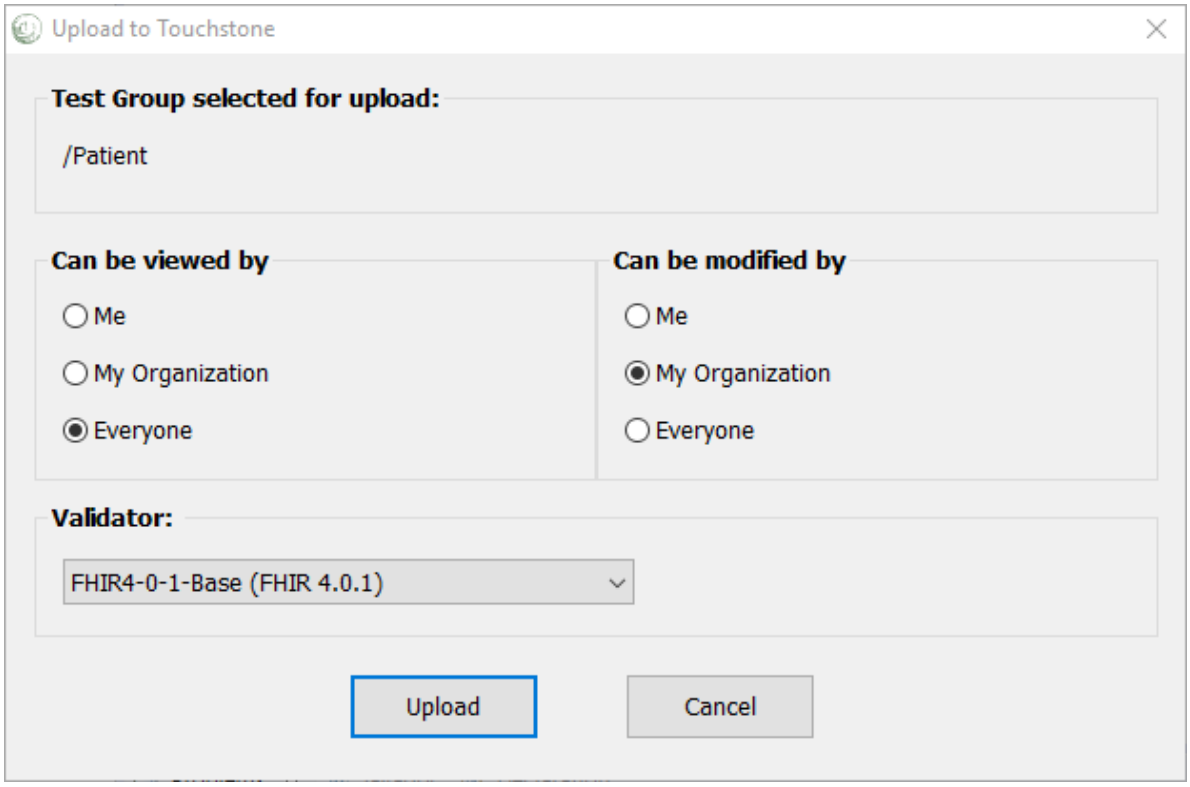
2. Notice the absence of `Patient` folder in FHIRSandbox in Touchstone UI:



3. Select the Patient folder and click on the Upload icon:

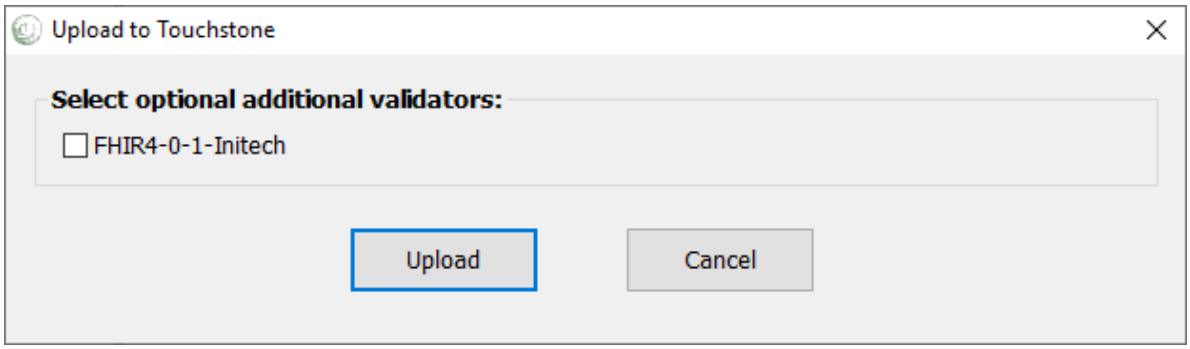


4. Select the View/Modify options and click the **Upload** button to submit or **Cancel** to cancel the request.

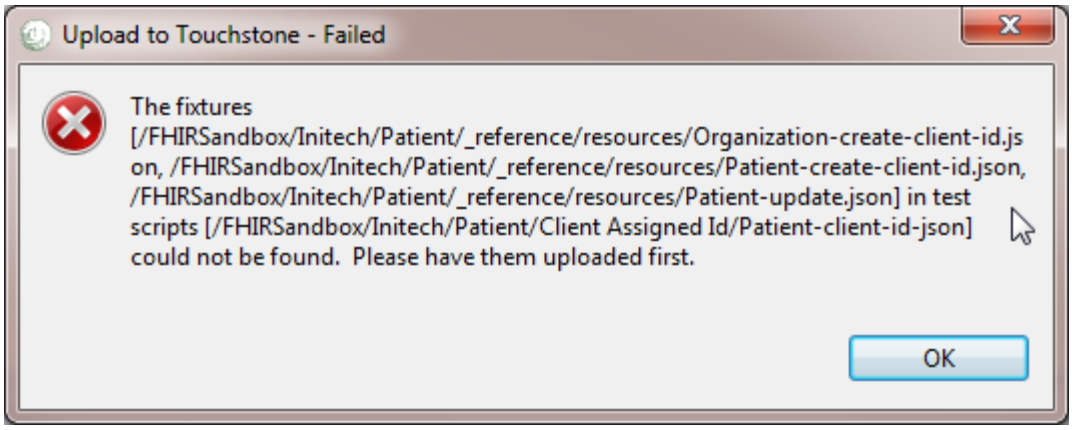


The `Can be viewed by` and `Can be modified by` options are explained in Upload Destination and Test Definition access.

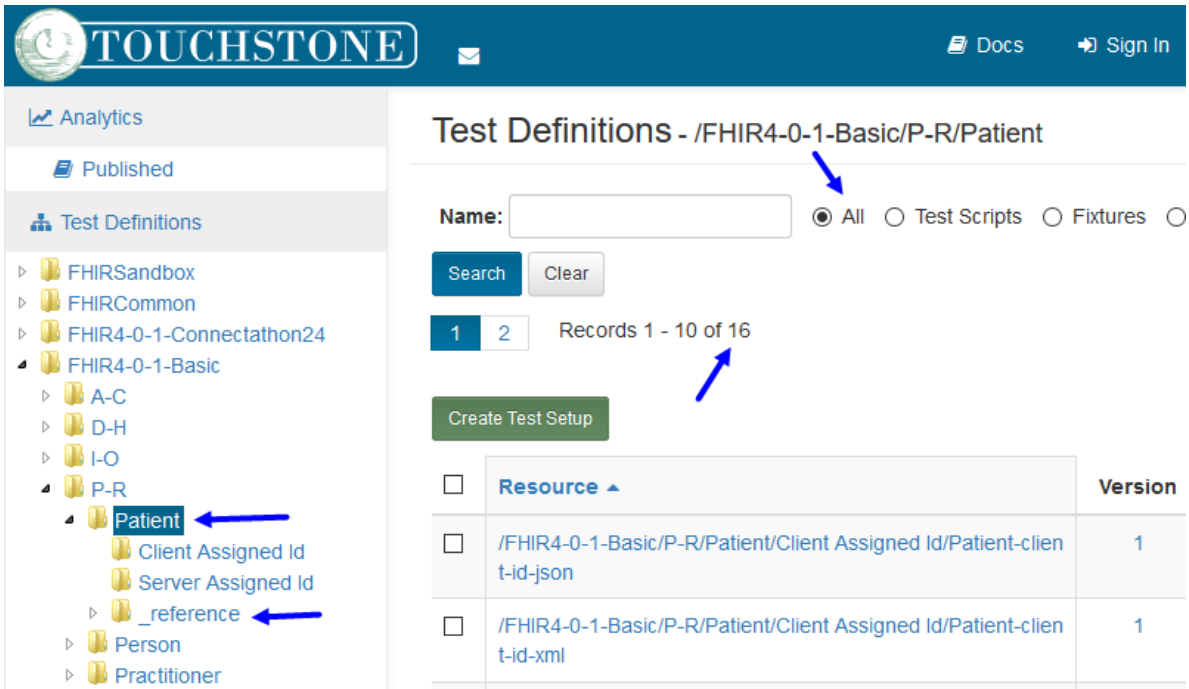
5. If the validator you selected has any additional validators linked to it, you will be prompted with another window where you can optionally choose to select zero or more validators to upload along with. If Upload is selected, the upload will run in the background:



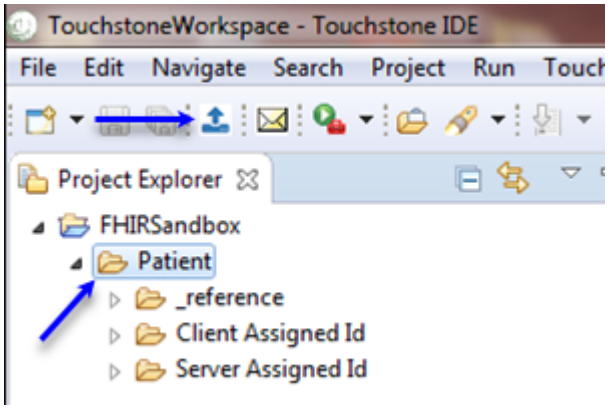
6. You will get an error from the server stating that all dependent fixtures were not included as part of the upload. Fixtures referred to by this test script should already exist in Touchstone or should be included as part of the upload.

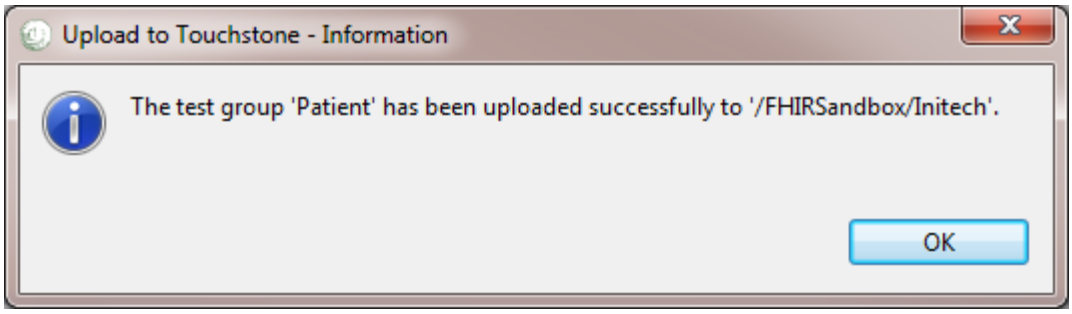


7. Current up-to-date scripts may be obtained from Test Definitions in Touchstone. You can download these example `Patient` test scripts and fixtures: Example Patient Test

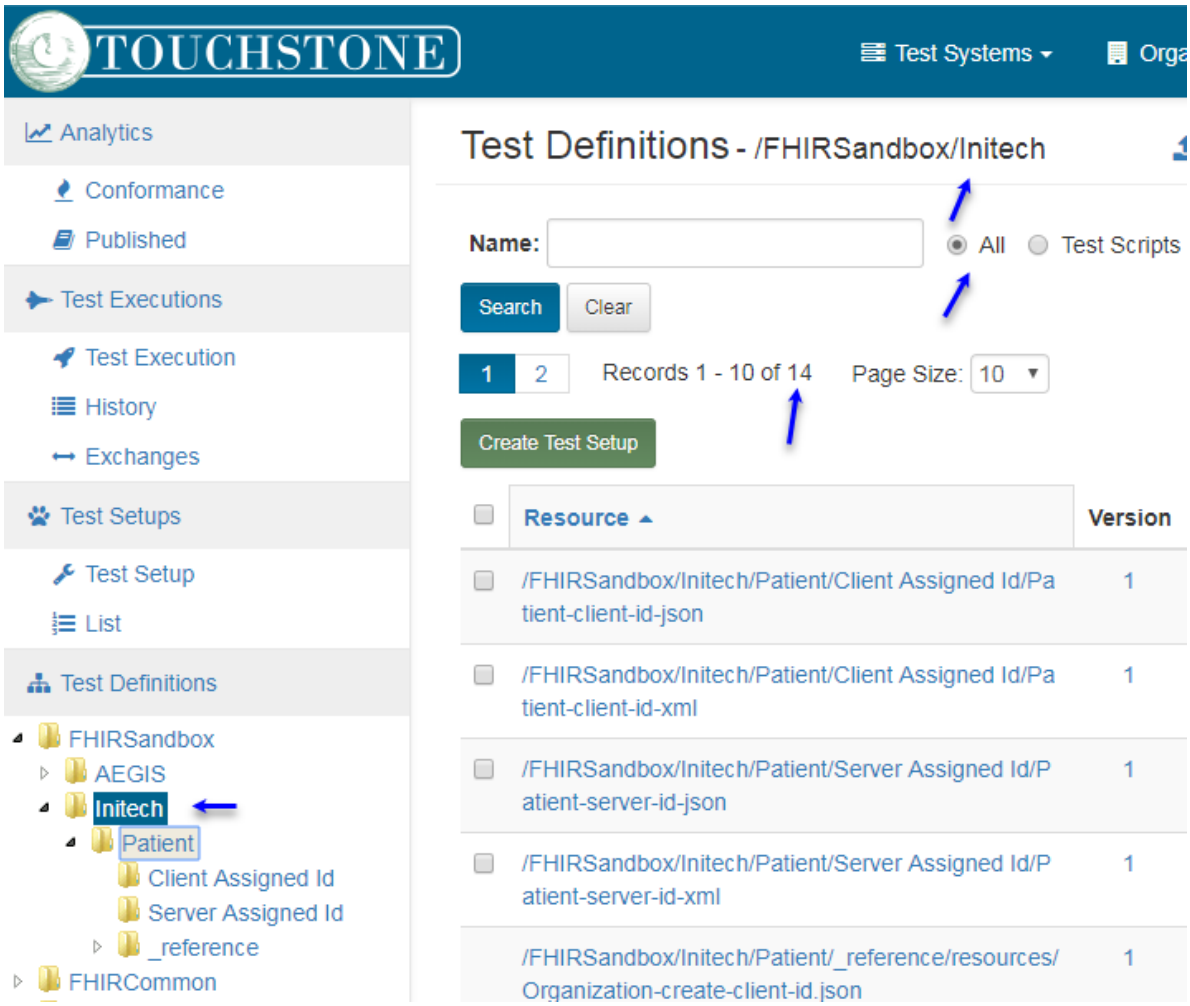


8. This time when we upload the `Patient` folder, the upload succeeds:

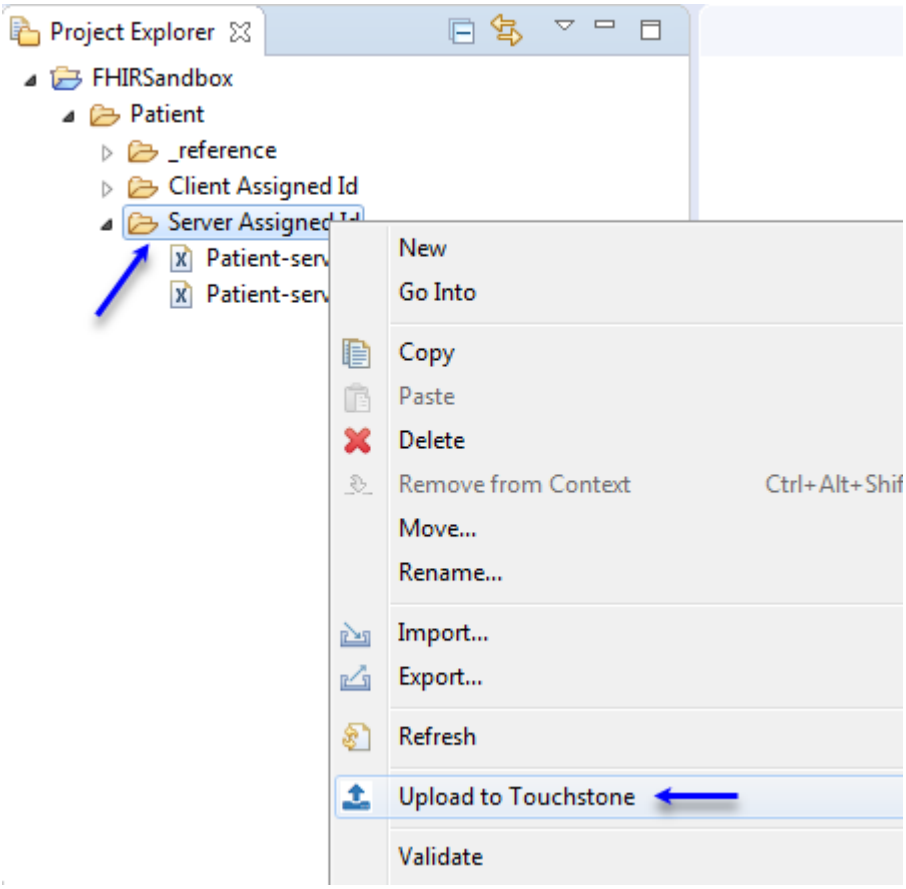


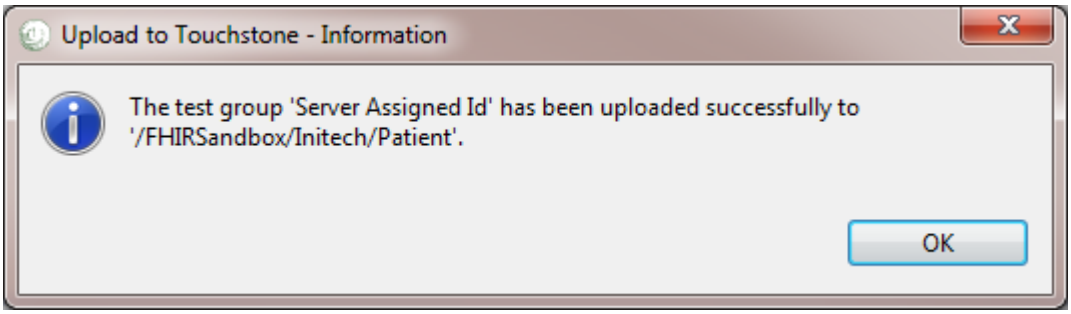


9. Notice that the test group `Patient` lands under the name of the organization the user belongs to:

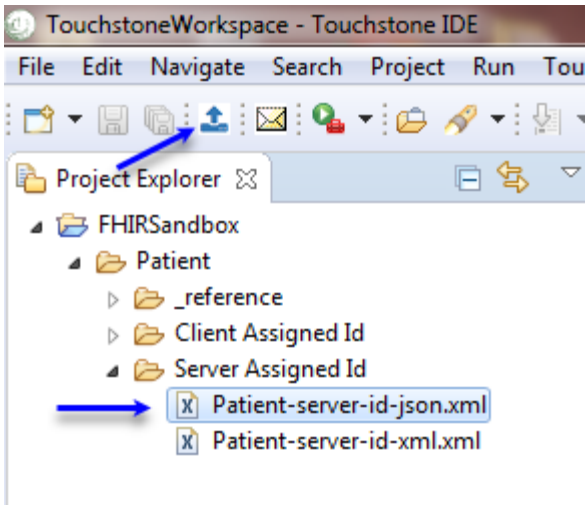


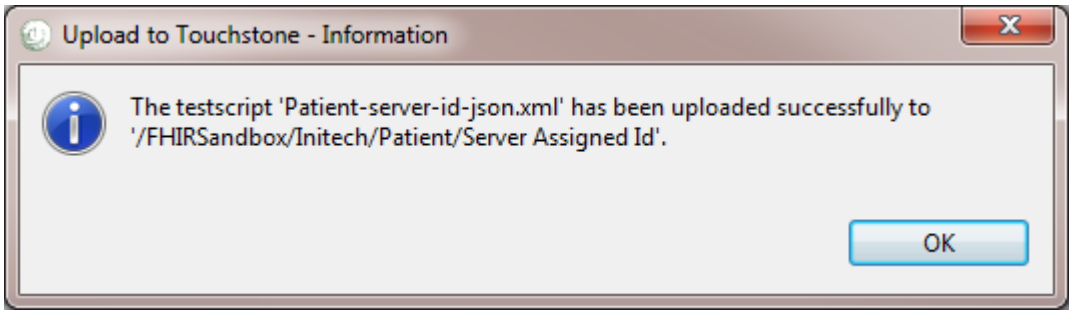
10. Now that the root folder (`Patient`) has been uploaded to Touchstone, you can make changes and create/upload individual folders within any level of the hierarchy:



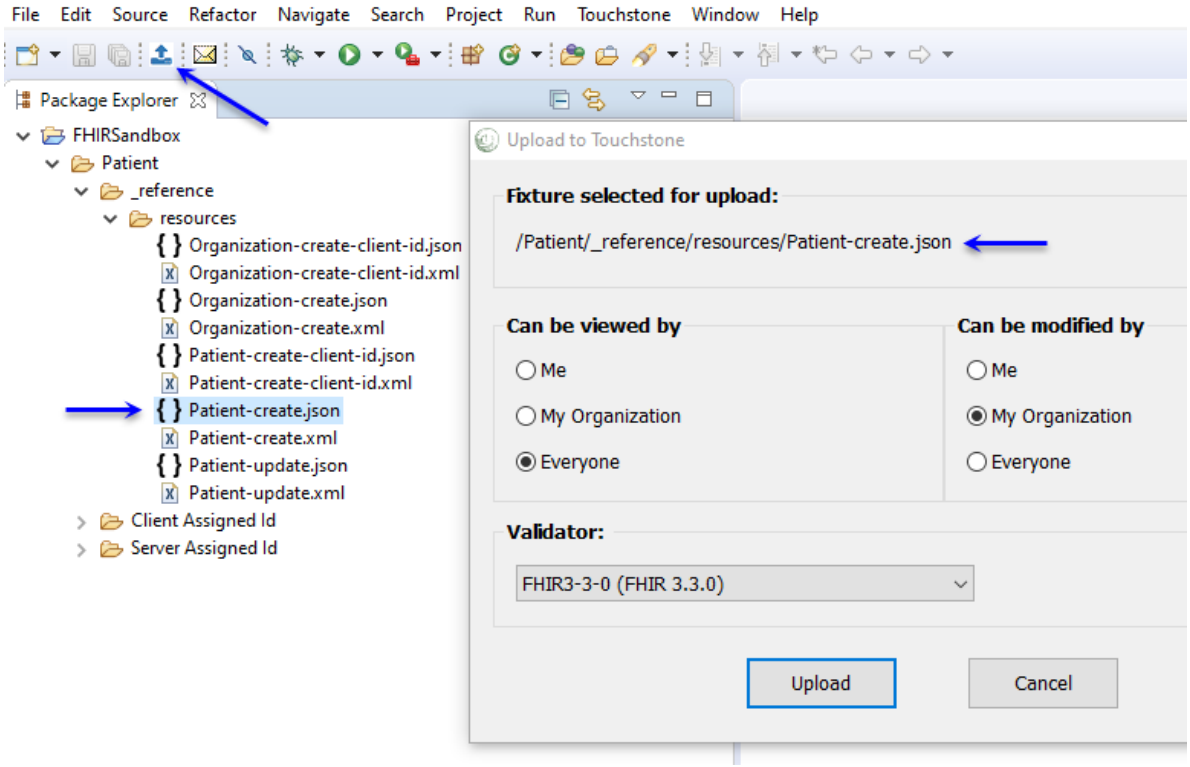


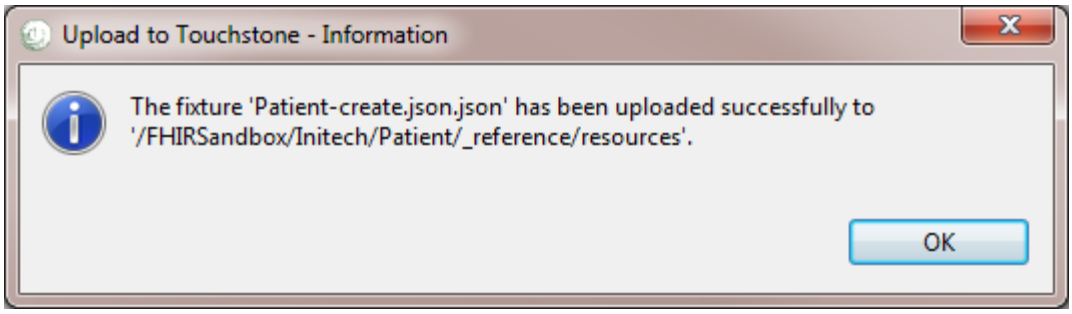
11. You can also upload changes to individual test scripts and create new test scripts:



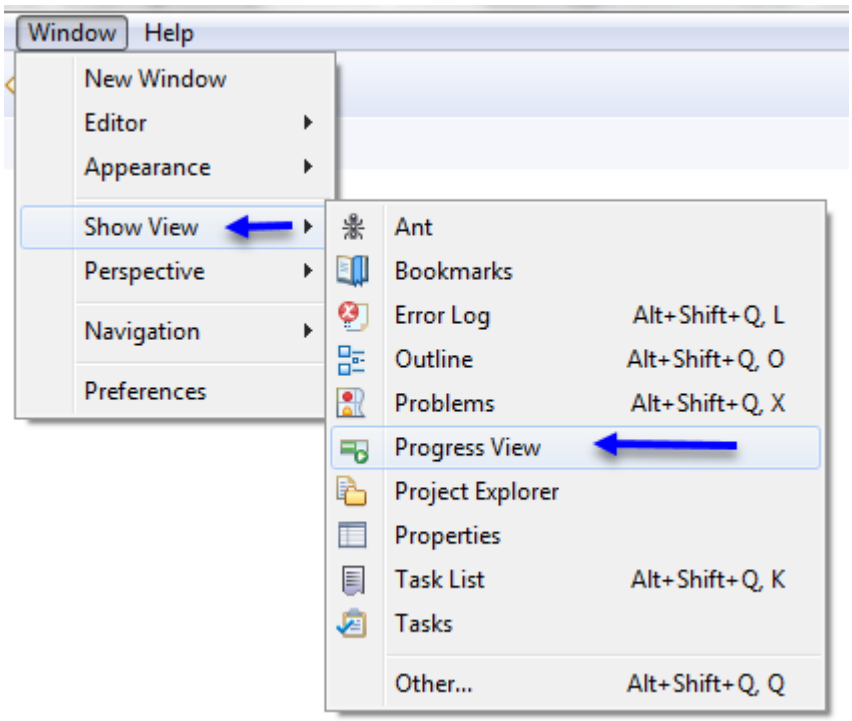


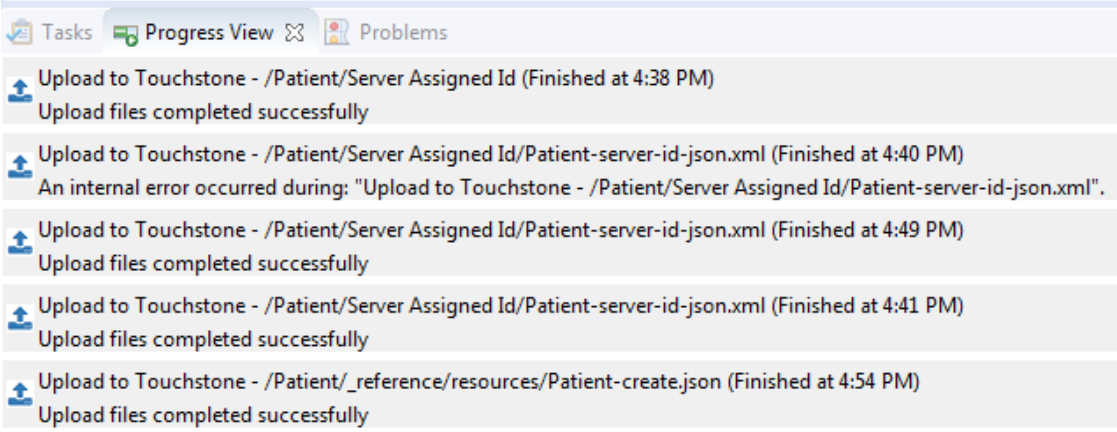
12. You can do the same with individual fixtures:





13. You can see the history of completed upload jobs in progress view:



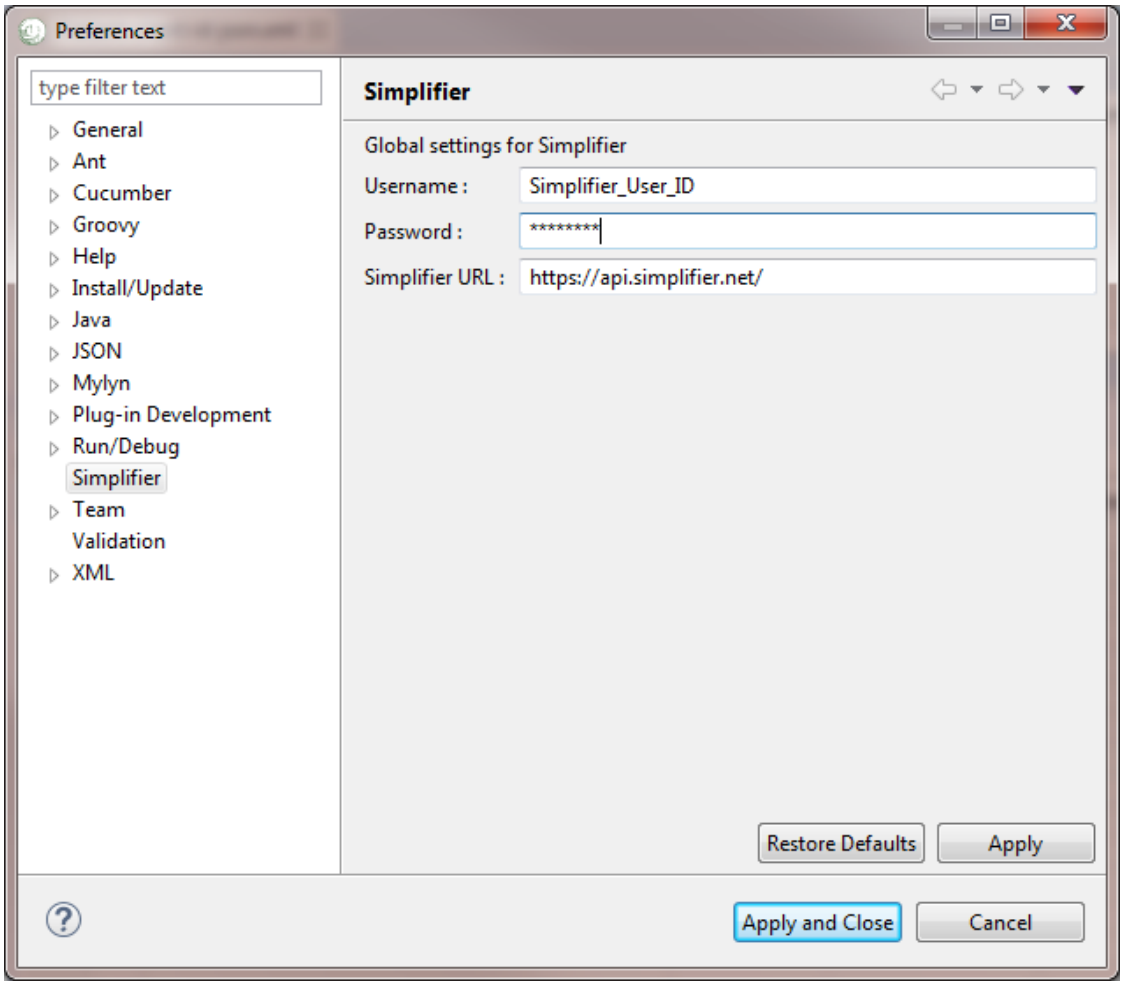


9.3.8 Simplifier Integration

9.3.8.1 Simplifier Preferences

To upload or download files you need to configure the Simplifier user credentials in preferences page `Windows -> Preferences -> Simplifier`

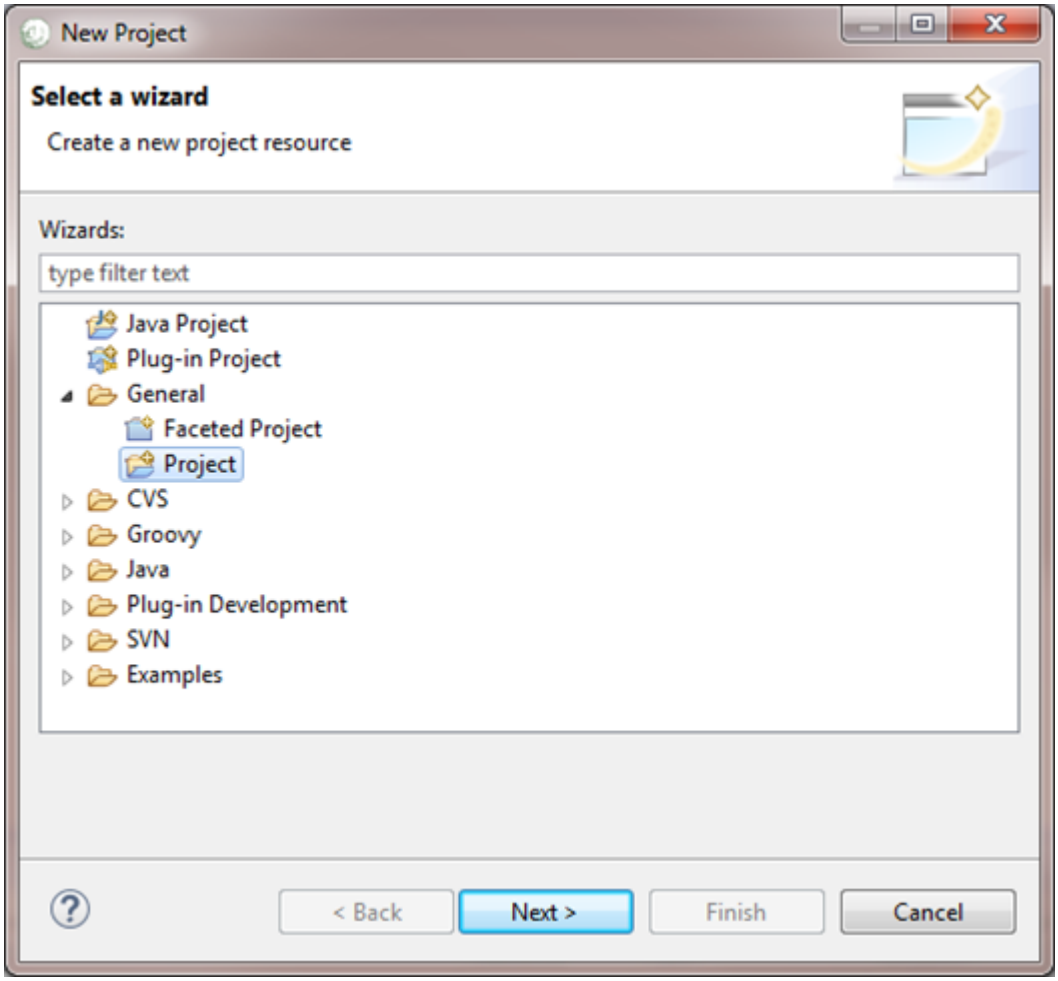
Enter your Simplifier user credentials and click *Apply and Close* button to save the preferences. This will be used to authenticate and authorize the requests in Simplifier.



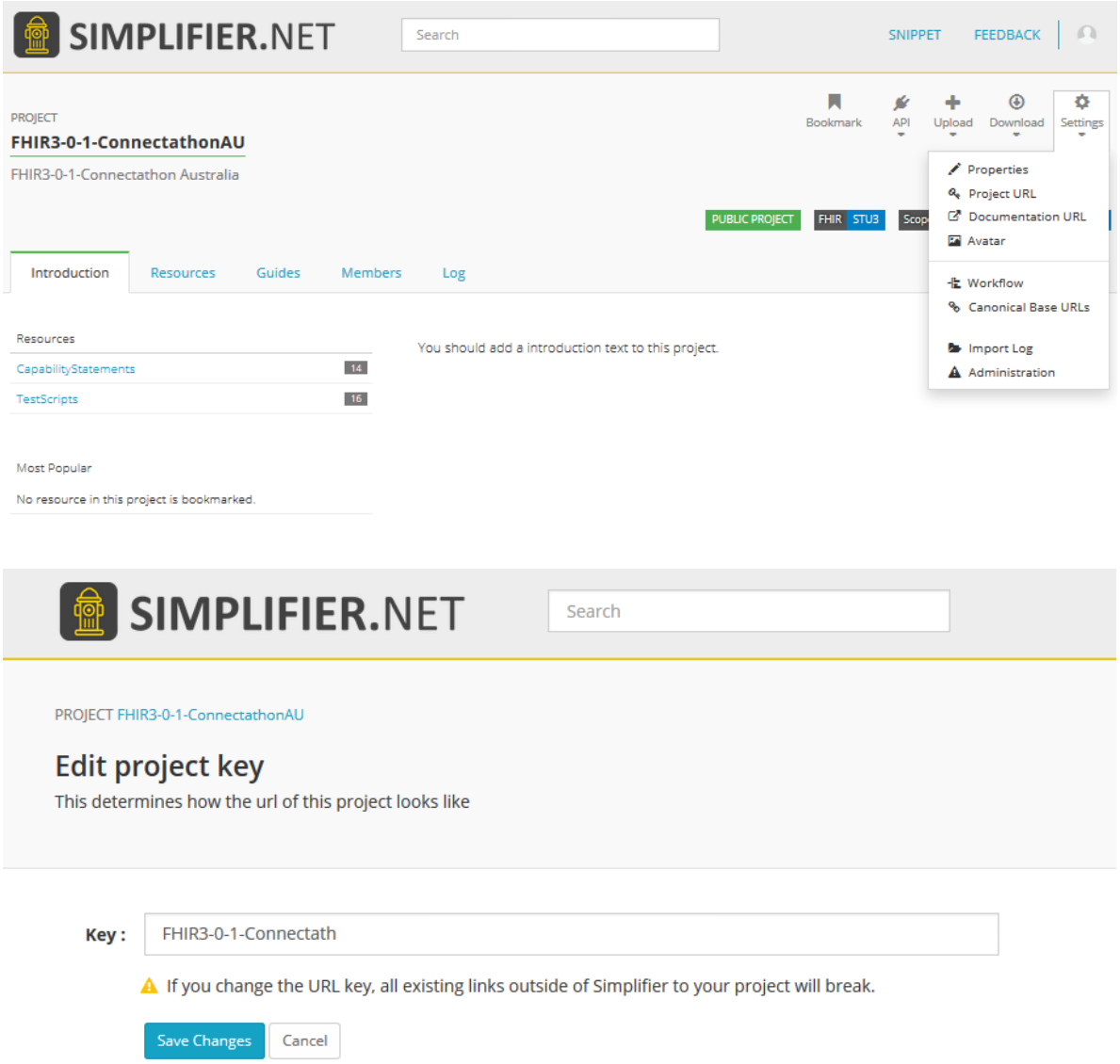
9.3.8.2 Create New Project

Start by creating a simple project as follows:

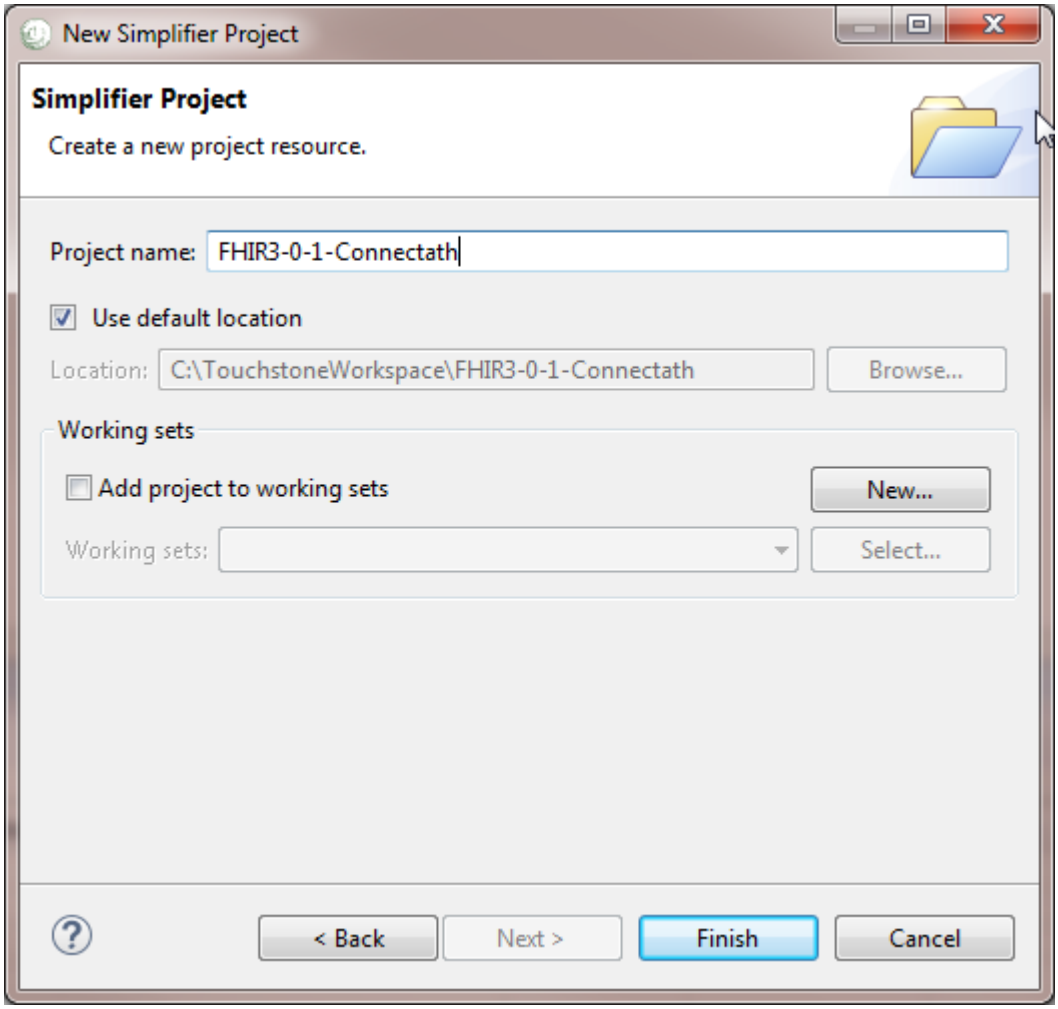
1. From the menu bar, select `File > New > Project...`



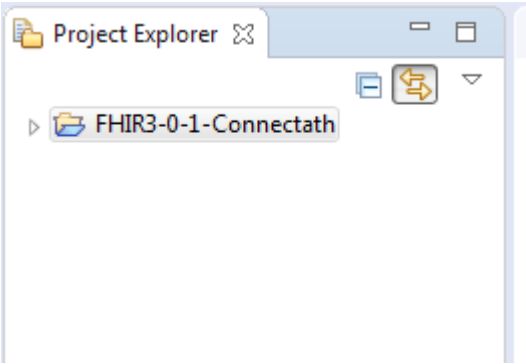
2. In the New Project wizard, select `General > Project` then click **Next**.
3. In the **Project name** field, enter the Simplifier Project key as the name of your new project e.g. **FHIR3-0-1-Connectath**. The Simplifier project key can be found in Simplifier account page `Settings > Project URL`



4. Leave the box checked to use the default location for your new project. Click Finish when you are done.



In the navigation view, you will see the **FHIR3-0-1-Connectath** project we just created.

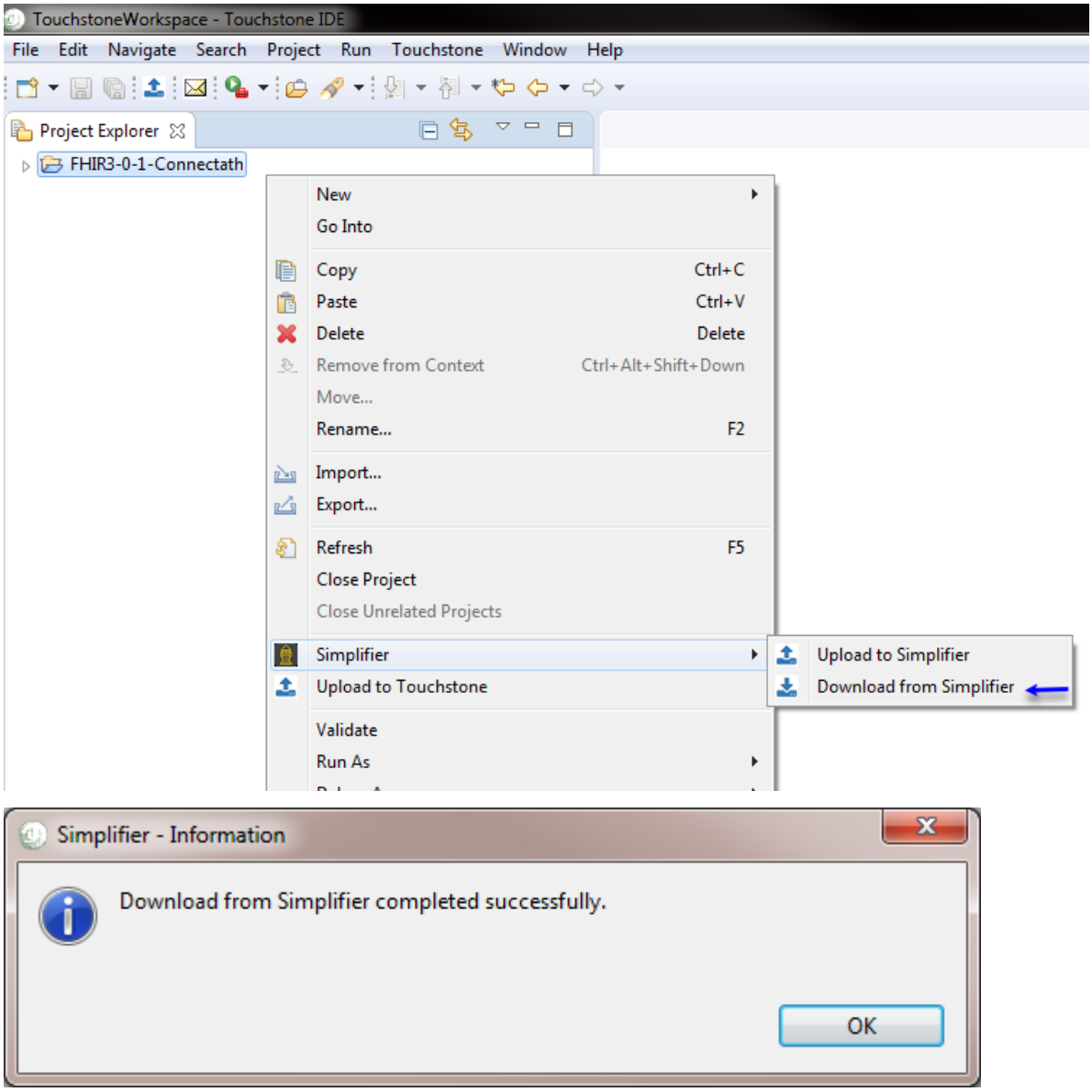


Note:

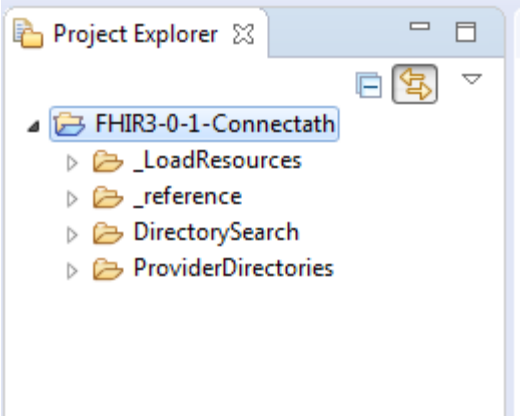
- You need to be connected to internet
- You need to have a Simplifier account with active projects.
- You need to configure the Simplifier user credentials in the editor before uploading or downloading files from Simplifier.

9.3.8.3 Download from Simplifier

1. In the Project Explorer view select the new project `FHIR3-0-1-Connectath` > `Right Click` > `Simplifier` > `Download from Simplifier`. If authentication is successful the files will be downloaded from Simplifier:

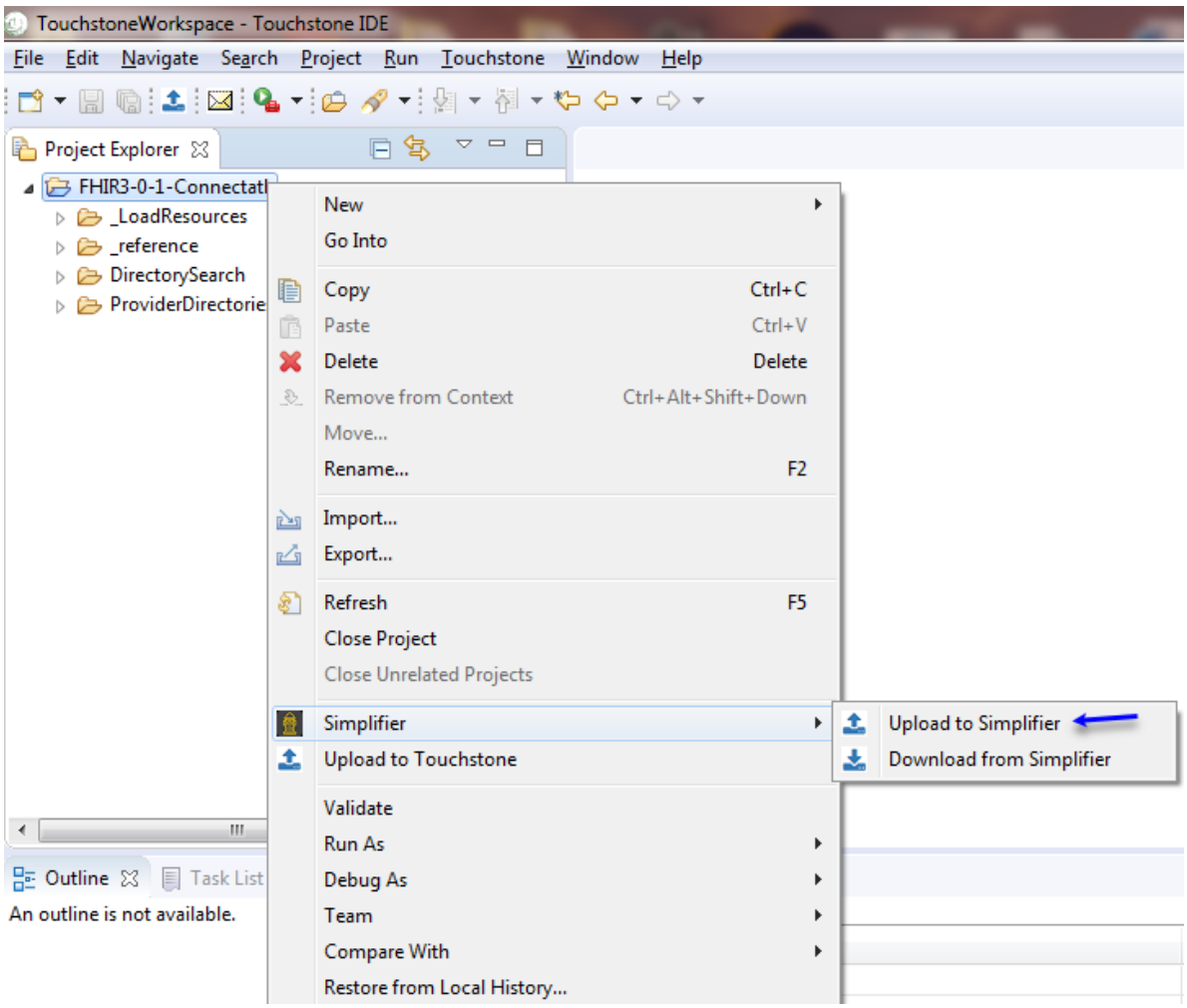


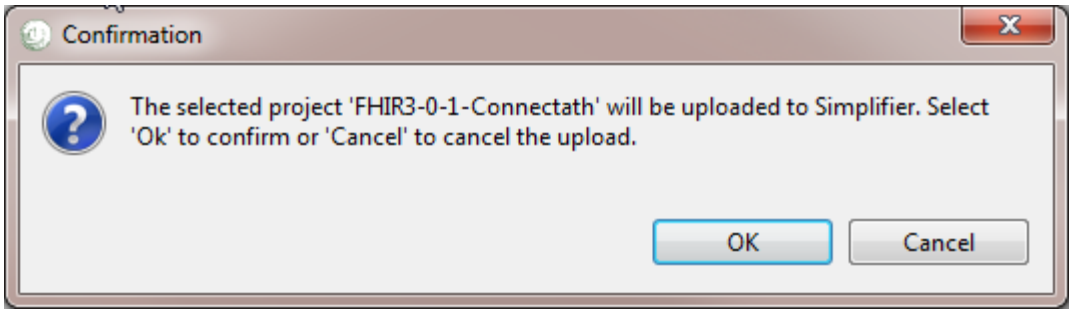
2. Refresh project **FHIR3-0-1-Connectath** and you should be able to see the downloaded files



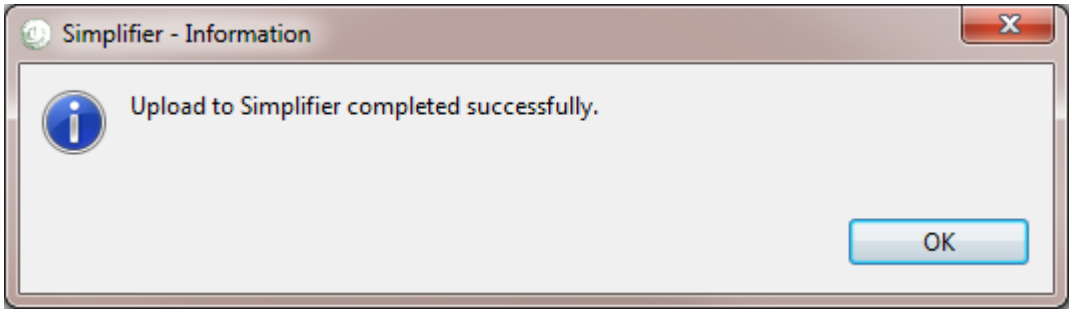
9.3.8.4 Upload to Simplifier

1. In the Project Explorer view select the new project `FHIR3-0-1-Connectath` > `Right Click` > `Simplifier` > `Upload to Simplifier`. The confirmation dialog window is displayed:

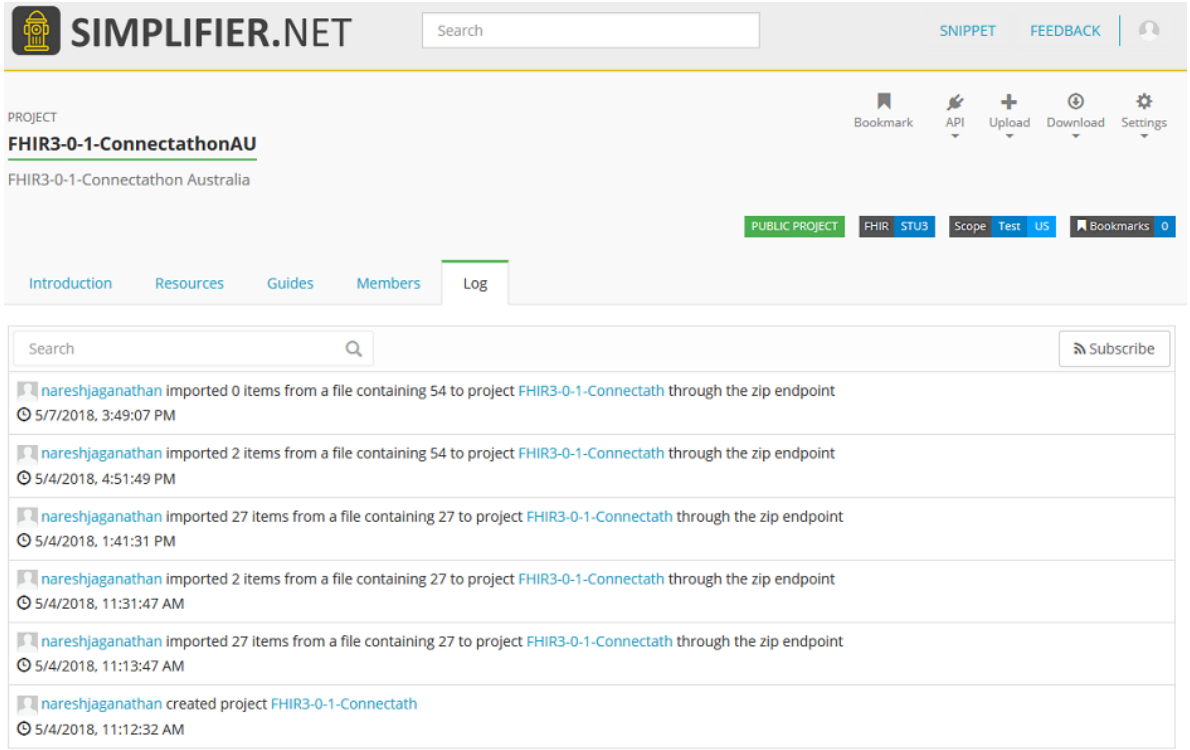




2. Select **Ok** to submit or **Cancel** to cancel the upload process.
3. If authentication is successful the files are uploaded to Simplifier project.



4. You can verify the uploads on the Log tab in Simplifier as shown below.



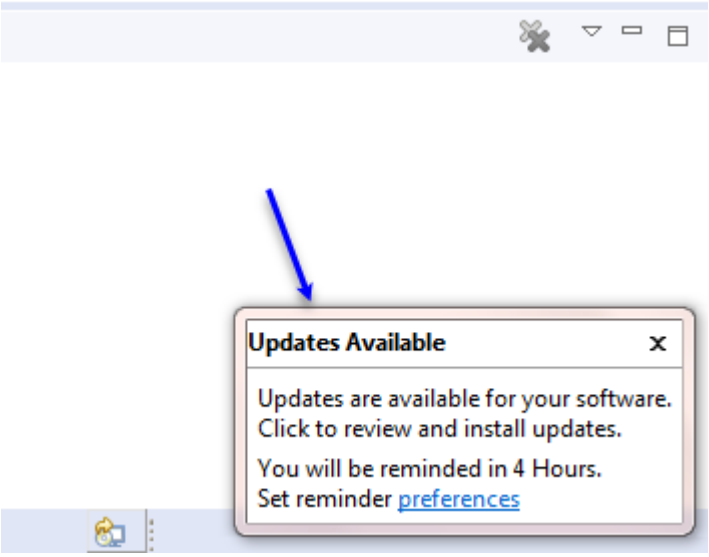
9.3.9 Updating TestScript Editor

You can update the TestScript Editor as follows:

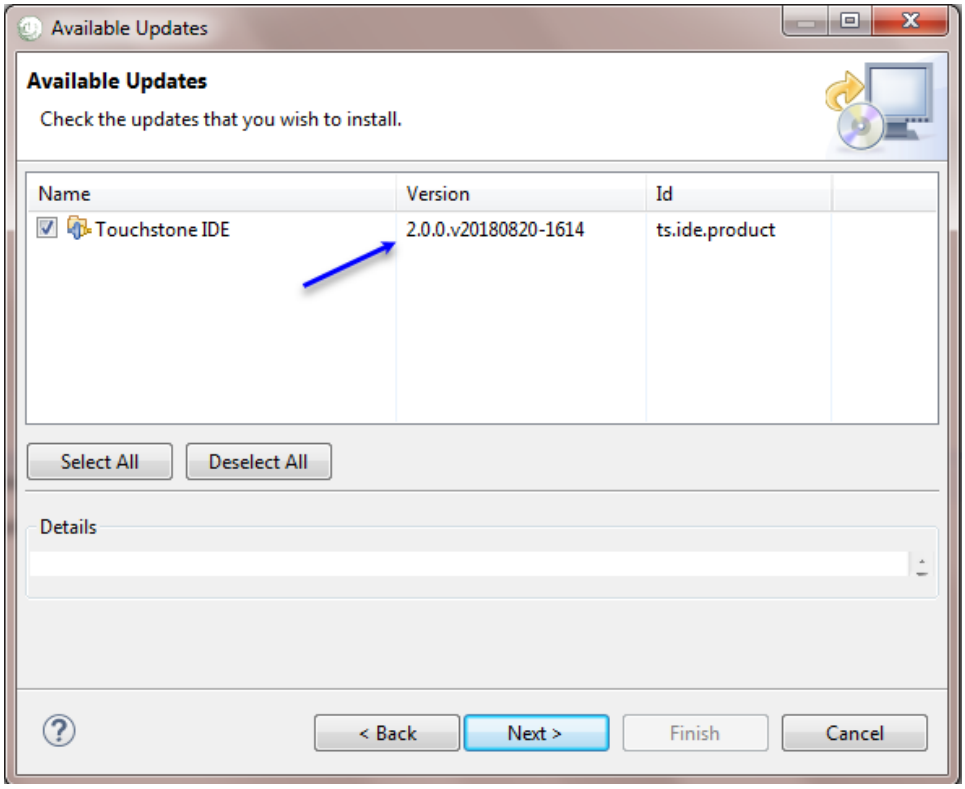
1. Check for Updates: (Recommended)

To check to see whether there are updates for the TestScript Editor in your system (requires Internet access):

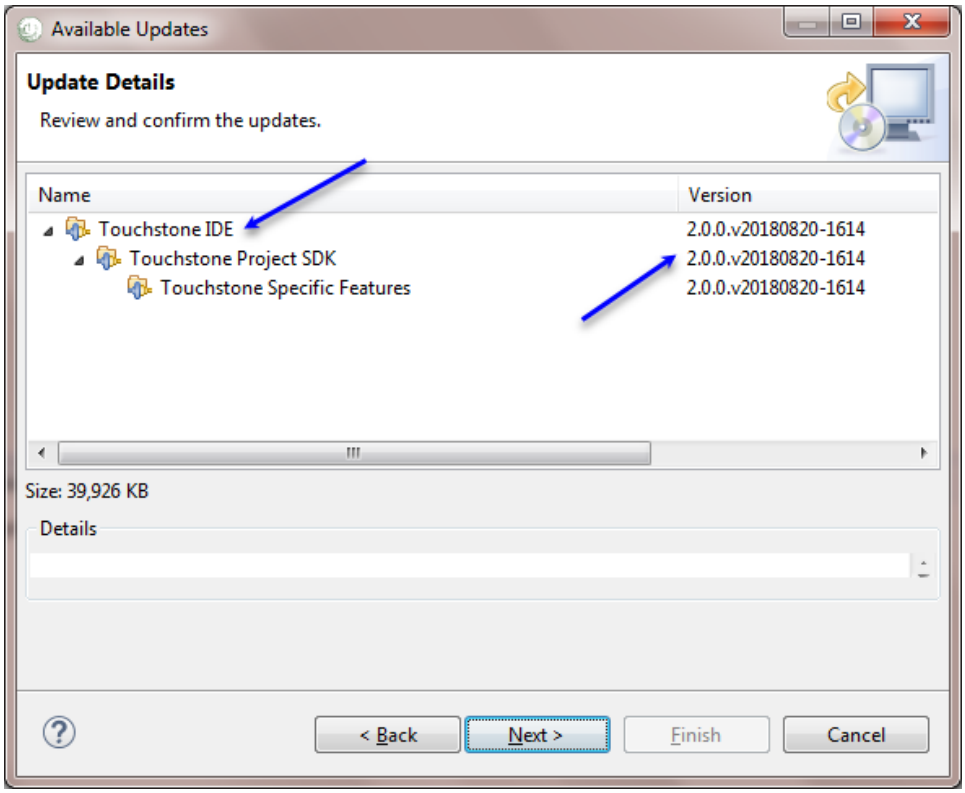
1. Click command link `Help > Check for Updates` or by clicking the **Updates Available** notification shown below. The notification is displayed when new updates are available on startup and reminds the user every 4 hours if not updated.



2. This will contact the Web sites defined in your Available Software Sites preferences to look for upgrades. If upgrades are available, they will be presented in the Available Updates wizard as shown below.

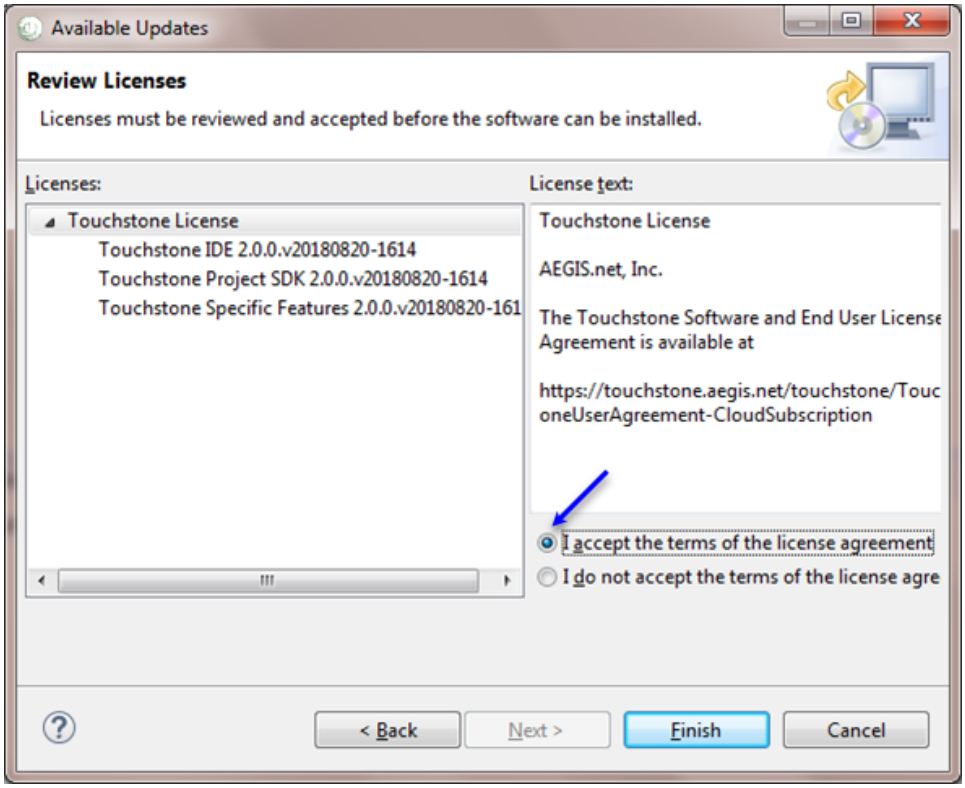


3. Check the updates that you wish to install. Click Next to see the details of the updates available as shown below.

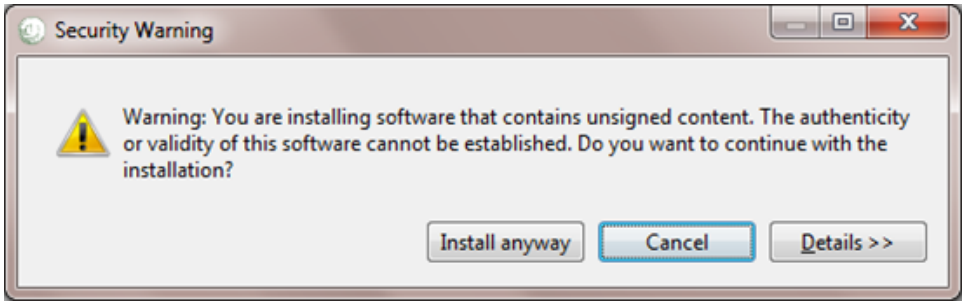


4. Click Next to review the license for selected items. If the terms of all these licenses are acceptable,

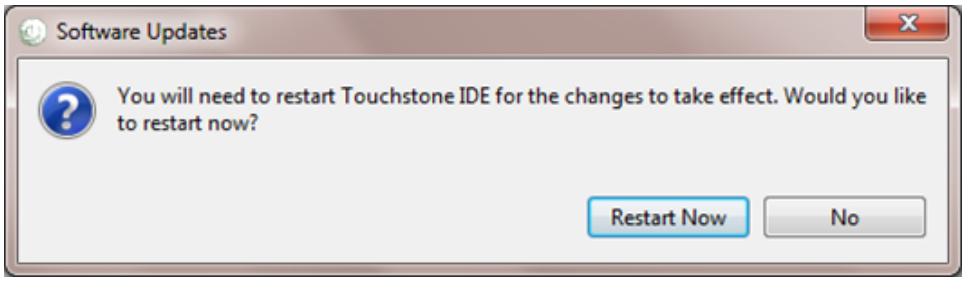
check "I accept the terms in the license agreements and click Finish to install the updates.



5. Since TestScript Editor updates are not digitally signed the security warning dialog will be displayed. Click **Install Anyway** to continue the installation.

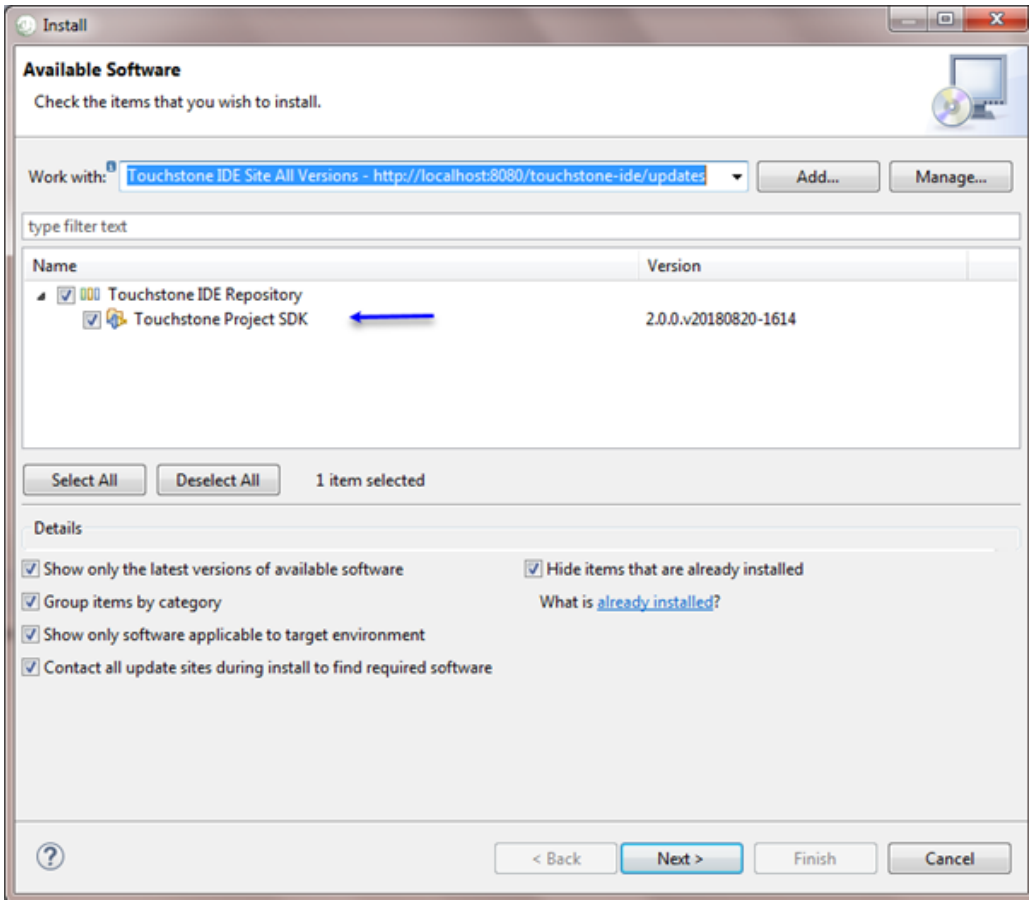


6. Once the updates are installed successfully, you will be prompted to restart the workbench. Click 'Restart Now' to restart the Workbench for the changes to take effect.

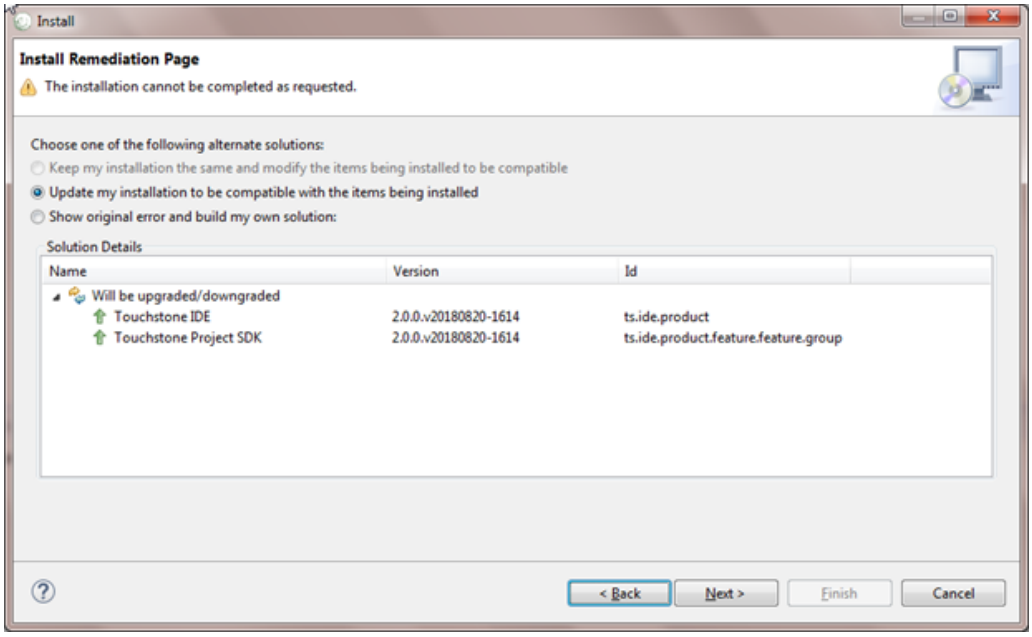


2. Using the Install New Software Wizard The Install New Software Wizard allows you to add new or update software to your installation. To install new or update an existing software:

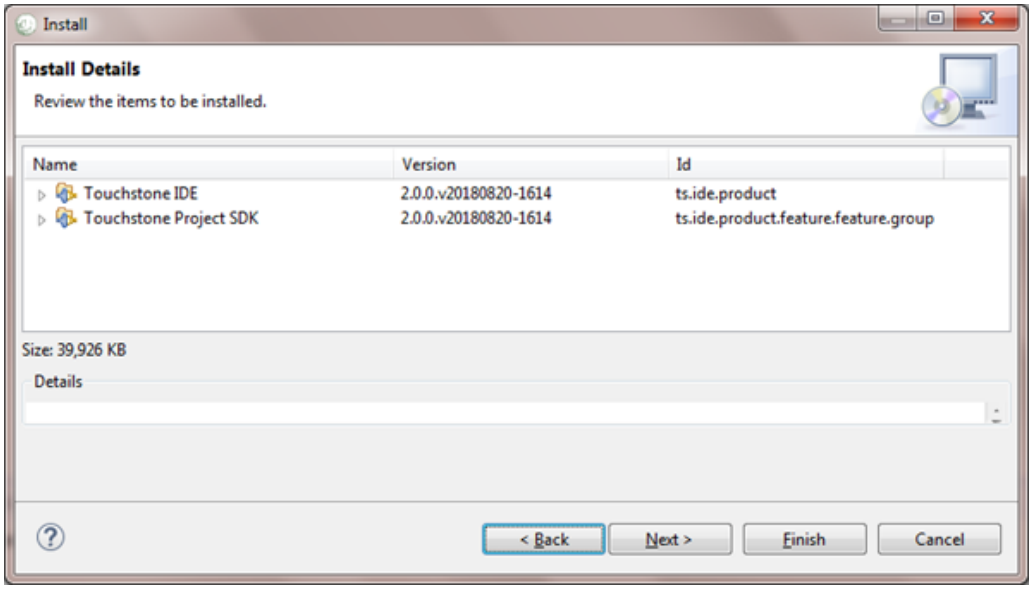
1. Click `Help > Install New Software...`. This wizard shows you the items that are available for installation.
2. Select the TestScript Editor update site using the **Works With** combo at the top of the page. By default the items in the site are grouped by category and the latest version of each item is shown.
3. As you browse the available software, you can check the items that you wish to install.



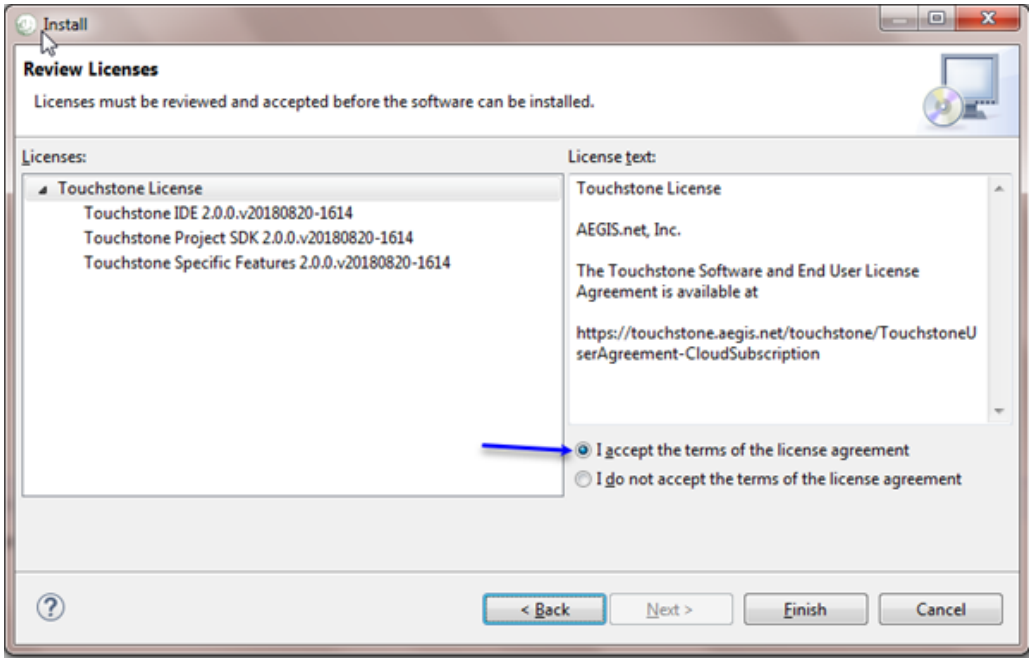
4. When you have finished making your selections, click Next to install the checked items. If the items you are installing require other software items in order to operate, those requirements will be included in your request. A checkbox at the bottom of this page controls whether all software sites will contacted when looking for requirements, or only the site shown in the Work With combo box.
5. Once you click Next, the wizard will validate your selections against your installed software, and may display the 'Install Remediation page' if the items selected are already installed. Select **Update my installation to be compatible with the items to be installed** and click Next.



6. If all of the requirements are available and there are no other installation conflicts, clicking Next will show the Install Details page. The items to be installed will be listed. Expanding each item will show what additional items will be required to complete the install. You will see an estimated size of the installation at the bottom of the page.

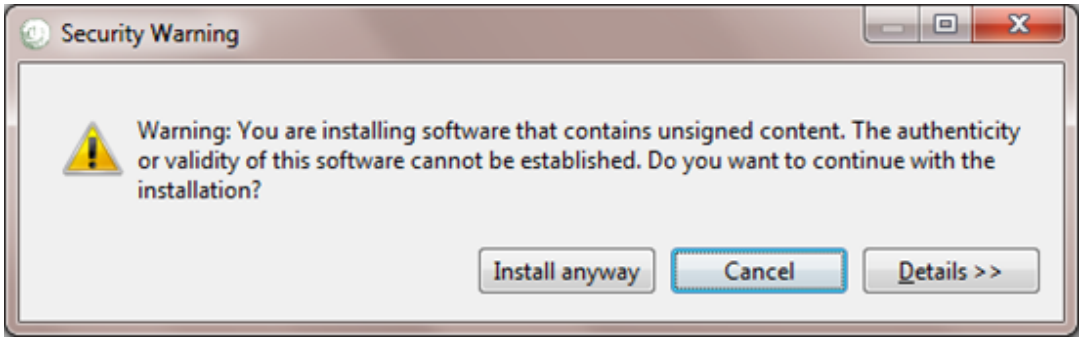


7. If the selected items have license agreements to be reviewed, you must click Next. Carefully review the license agreements for the items you wish to install. If the terms of all these licenses are acceptable, check "I accept the terms in the license agreements." Do not proceed to download the features if the license terms are not acceptable.

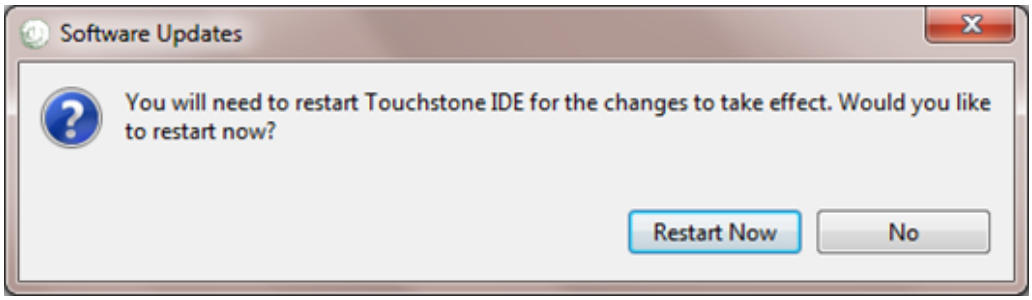


8. **If the license agreements are acceptable, click Finish. This will begin the download and installation of the new softwa**

    1. Since TestScript Editor updates are not digitally signed the security warning dialog will be displayed. Click **Install Anyway** to continue the installation.



9. Once the updates are installed successfully, you will be prompted to restart the workbench. Click 'Restart Now' to restart the Workbench for the changes to take effect.



9.3.10 References

Eclipse Workbench Documentation

XML Editor Documentation

SVN Plugin Documentation

GIT Plugin Documentation

9.4 Best Practices

9.4.1 Version Control

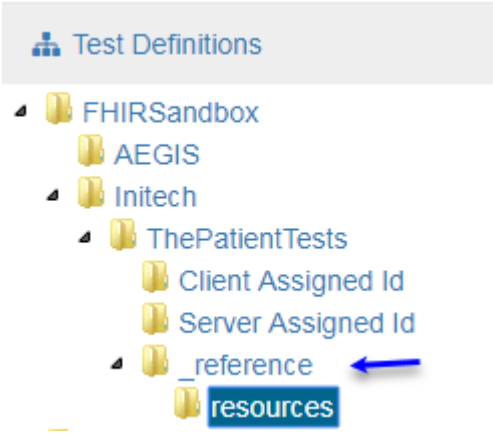
Although Touchstone does detect changes to uploaded test scripts and its dependenct resources, the versioning in Touchstone is rudimentary. You should not rely on Touchstone as the sole repository for your test scripts.

It is **highly recommended** to manage your test scripts in a proper Version Control system (e.g. Git, Subversion, TFVC, etc.) outside of Touchstone. These tools are specifically engineered to manage versioning, conflicts, branching, concurrent commits, etc.

In the event that your test group gets corrupted in Toucshtone through concurrent uploads by different users, you could re-upload the test group using the right version in your Version Control system.

9.4.2 Location

It is recommended to store the fixtures and rules for your test scripts in a separate folder and call the folder "_references". This will make it easier to tell apart these definitions from test scripts when browsing.



9.4.3 Paths to resources

For resources that your test scripts are depending on and that are specific to the test scripts in the test group you're uploading, it is highly recommended to use relative paths to reference those resources. Doing so will allow you to change the location of your test groups much more easily.

If you used absolute paths, then you'd have to go through all your test scripts and change the paths of the referenced fixtures. That can be time-consuming and error prone.

In the test script snippet below where a fixture is defined, we're using a relative path:

```
<fixture id="resource-create">
    <autocreate value="false"/>
    <autodelete value="false"/>
    <resource>
        <reference value="../_reference/resources/Patient-create-client-id.json"/>
    </resource>
</fixture>
```

If you're defining resources that are going to be used across many test groups (especially test groups shared among different Org Group members), it would be better to define the resources in a separate test group and use absolute paths to refer to those resources.

In the test script snippet below where a rule is defined, we're using an abolute path. The rule is used across many test groups in Touchstone.

```
<rule id="rule-assertContentTypeIfBody">
    <resource>
        <reference value="/FHIRCommon/_reference/rule/AssertHeaderIfBody.groovy"/>
    </resource>
    <param>
```

9.5 Exclusions

By default, Touchstone will treat uploaded files with ".json", ".xml", ".groovy", ".sch", and ".xslt" extensions as test definitions.

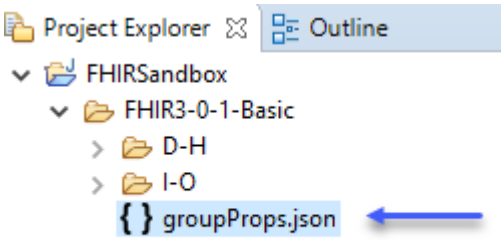
- JSON and XML test definitions are parsed during upload and are expected to be in proper format or the upload will fail.
- Files with ".groovy", ".sch", and ".xslt" extensions are treated as rule definitions and are expected to contain certain content in its headers or the upload will fail.
- Uploaded test definitions that contain FHIR resources will be validated periodically using the FHIR Validation Engine and will be flagged for validation errors.

All these restrictions would prevent end users from:

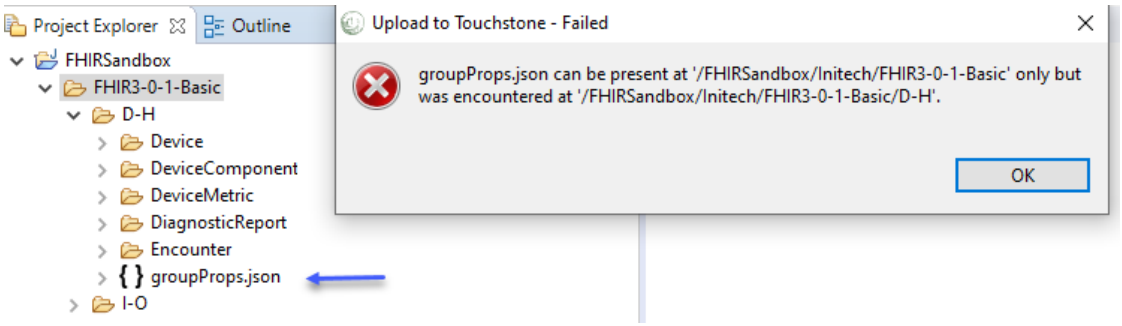
- Using invalid JSON and XML in their fixtures for negative testing.
- Storing Schematron and XSLT files alongside their test definitions for schema-validation and transformation purposes.
- Temporarily excluding test definitions from validation by a validation engine.
- etc.

9.5.1 Props Location

Touchstone provides the ability to exclude folders and files from upload, parsing, and validations via exclusions in **groupProps.json**. This file must reside at the root of the main test group. For example, if the main test group that's being uploaded is FHIR3-0-1-Basic, then groupProps.json file must reside at FHIR3-0-1-Basic:



If the file is incorrectly located one-level deeper (e.g. at FHIR3-0-1-Basic/D-H), then attempting to upload the test group will produce the following error:



9.5.2 Props format

Regular Expressions are supported in the exclusions. There are four types of exclusions:

1. **excludeFromUpload** -> Completely prevents test definitions from ending up in Touchstone.
2. **excludeFromParsing** -> Test definitions are uploaded to Touchstone but they're not parsed or validated.
3. **excludeFromValidation** -> Test definitions are uploaded to Touchstone and are parsed but they're not periodically validated using external Validaiton Engine.
4. **excludeFromConformance** -> Test definitions are uploaded to Touchstone and are parsed but they're excluded from getting included in conformance evaluations.

- Test definitions that are specified in **excludeFromUpload** do *not* need to be specified in **excludeFromParsing** and **excludeFromValidation** as they'll be excluded from upload completely.
- Test definitions that are specified in **excludeFromParsing** do *not* need to be specified in **excludeFromValidation** as they'll neither be parsed nor validated.

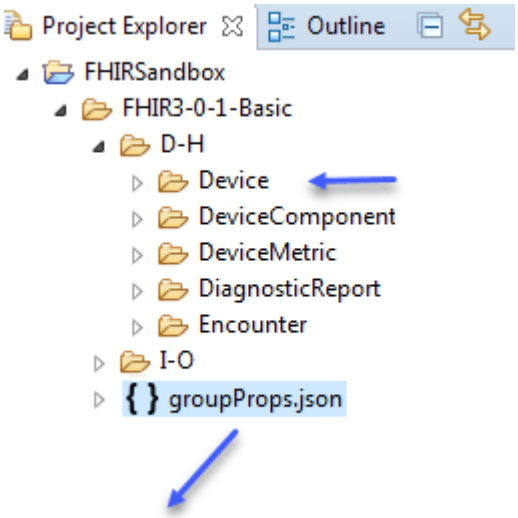
> **Warning:** Minimize the number of entries in **excludeFromUpload**, **excludeFromParsing**, and **excludeFromValidation** as each entry is evaluated separately during upload. Upload could take long if there are too many entries. It's better to agree upon a naming convention in your organization and use a regular expression that matches the agreed-upon convention.

We will cover various scenarios via examples next.

9.5.3 Upload Exclusions

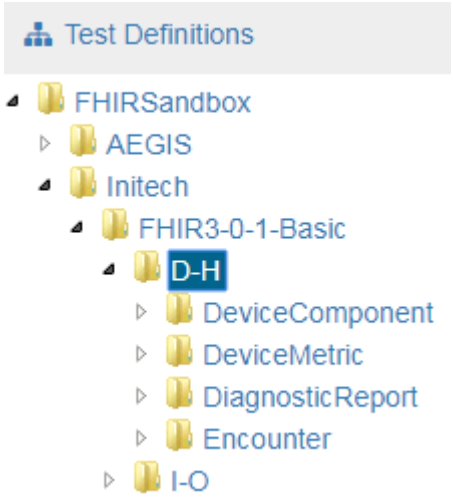
Upload exclusions are specified using the **excludeFromUpload** key in groupProps.json. Values can be specified using regular expressions. Matching folders and files will be filtered out from the upload and will not end up in Touchstone.

To exclude the Device test group from getting uploaded to Touchstone:




```
{
   "excludeFromUpload": [
      ".*/FHIR3-0-1-Basic/D-H/Device/.*"
   ]
}
```

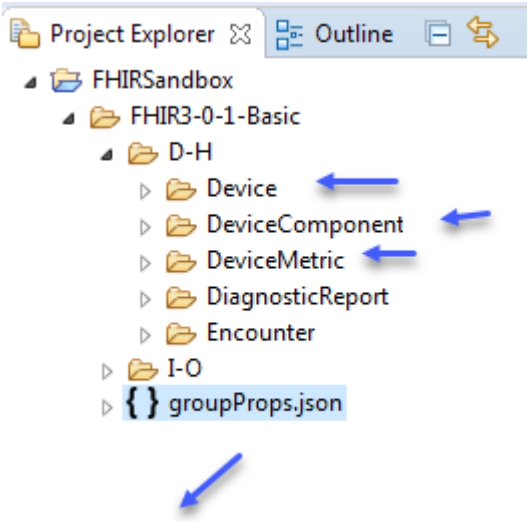
After the FHIR3-0-1-Basic test group is uploaded, the Device sub group will not be there in Touchstone:



Note: After the groupProps.json file ends up in Touchstone, you can upload sub-groups and files individually as usual i.e. You do not have to upload the main test group (FHIR3-0-1-Basic in this case) every time. Touchstone will enforce the exclusions based on the last version of groupProps.json in the system for the

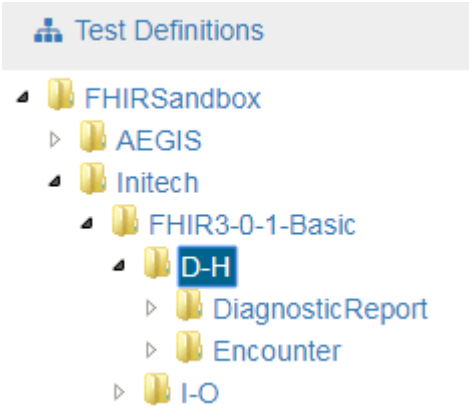
main test group. If you need to update the exclusions, you can re-upload the main test group (FHIR3-0-1-Basic in this case) which includes the updated groupProps.json. You can alternatively upload the updated groupProps.json file alone for the new exclusions to take effect.

To exclude test groups that start with "Device" from getting uploaded to Touchstone:

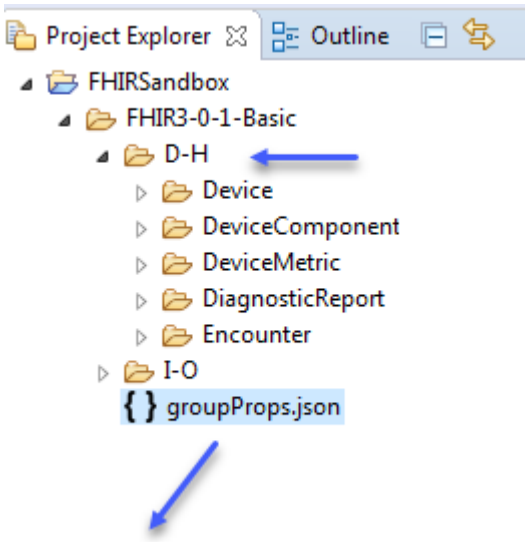



```
{
   "excludeFromUpload": [
      ".*/FHIR3-0-1-Basic/D-H/Device.*"
   ]
}
```

After the FHIR3-0-1-Basic test group is uploaded, the "Device", "DeviceComponent", and "DeviceMetric" sub groups will not be there in Touchstone:

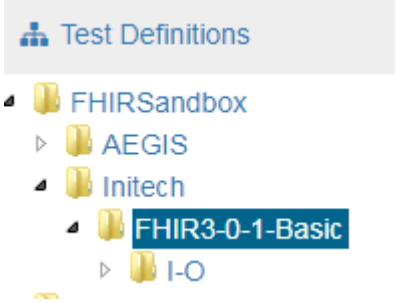


To exclude "D-H" sub group from getting uploaded to Touchstone:

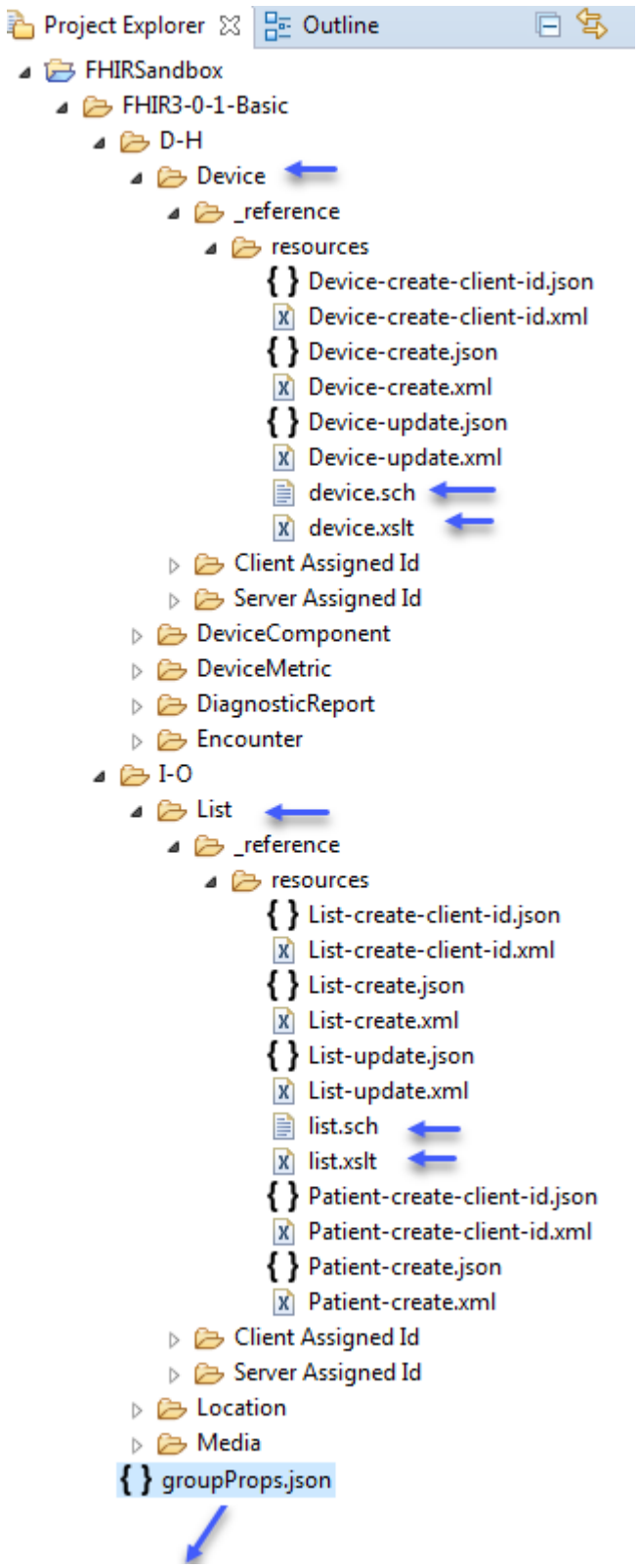



```
{
   "excludeFromUpload": [
      ".*/D-H/.*"
   ]
}
```

After the FHIR3-0-1-Basic test group is uploaded, the "D-H" sub group will not be there in Touchstone:



To exclude ".sch" and ".xslt" files in Devices sub group from getting uploaded to Touchstone:




```
{
   "excludeFromUpload": [
```

(continues on next page)

(continued from previous page)

```
      ".*/Device/.*\\.sch",
      ".*/Device/.*\\.xslt"
   ]
}
```

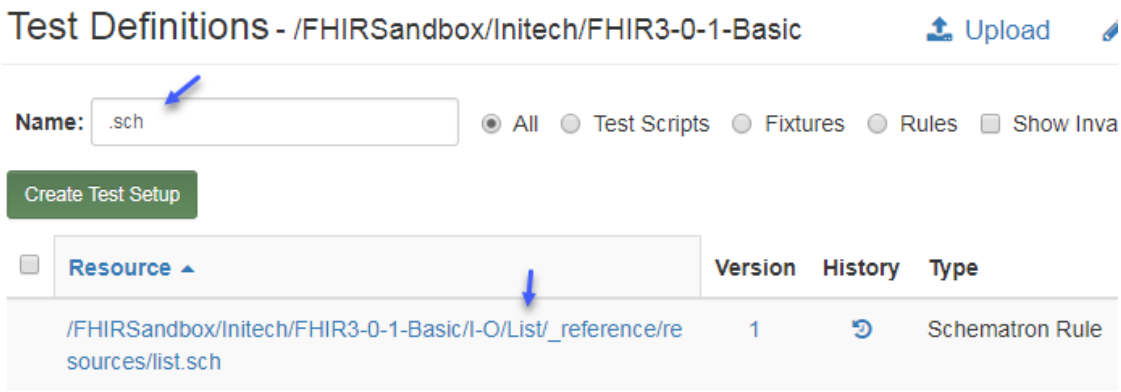
or

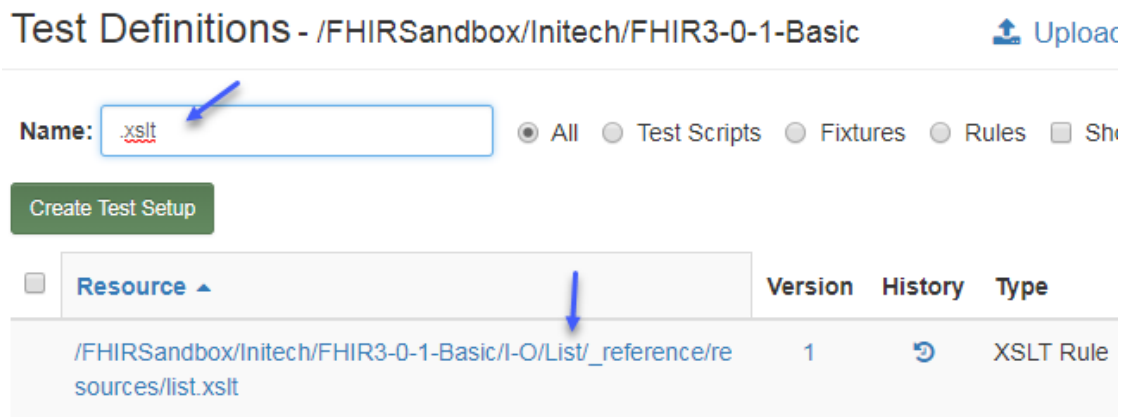
```
{
   "excludeFromUpload": [
      ".*/Device/.*\\.(sch|xslt)"
   ]
}
```

The second one above is better as it involves fewer evaluations during upload and thereby slightly better performance.

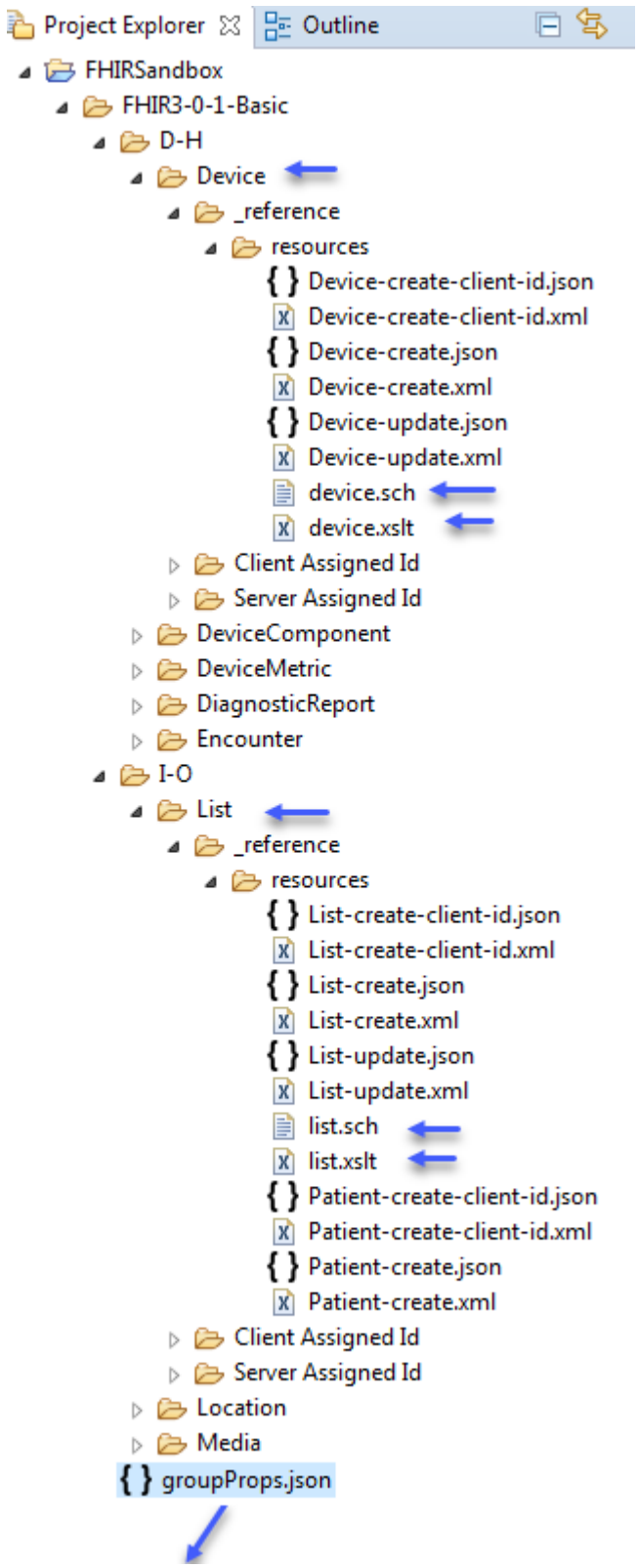
> **Note:** Certain characters must be escaped when included in a json string. The regex backslash must be escaped using another backslash, hence the double backslash in the example.

After the FHIR3-0-1-Basic test group is uploaded, the ".sch" and ".xslt" files in Device sub group will not be there in Touchstone but those in List sub group will be:





To exclude all ".sch" and ".xslt" files from getting uploaded to Touchstone:




```
{
   "excludeFromUpload": [
```

(continues on next page)

(continued from previous page)

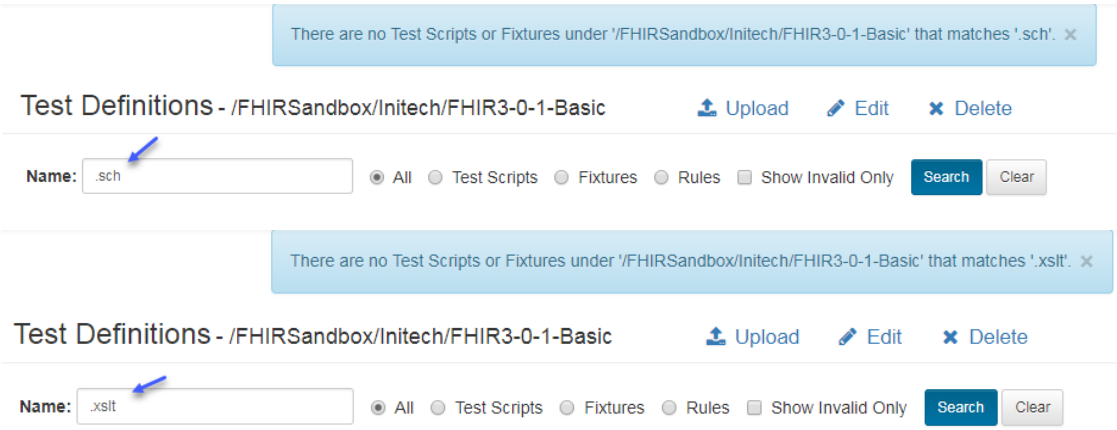
```
        ".*\\.sch",
        ".*\\.xslt"
    ]
}
```

or

```
{
    "excludeFromUpload": [
        ".*\\.(sch|xslt)"
    ]
}
```

The second one above is better as it involves fewer evaluations during upload and thereby slightly better performance.

After the FHIR3-0-1-Basic test group is uploaded, the ".sch" and ".xslt" files will not be anywhere under FHIR3-0-1-Basic in Touchstone:



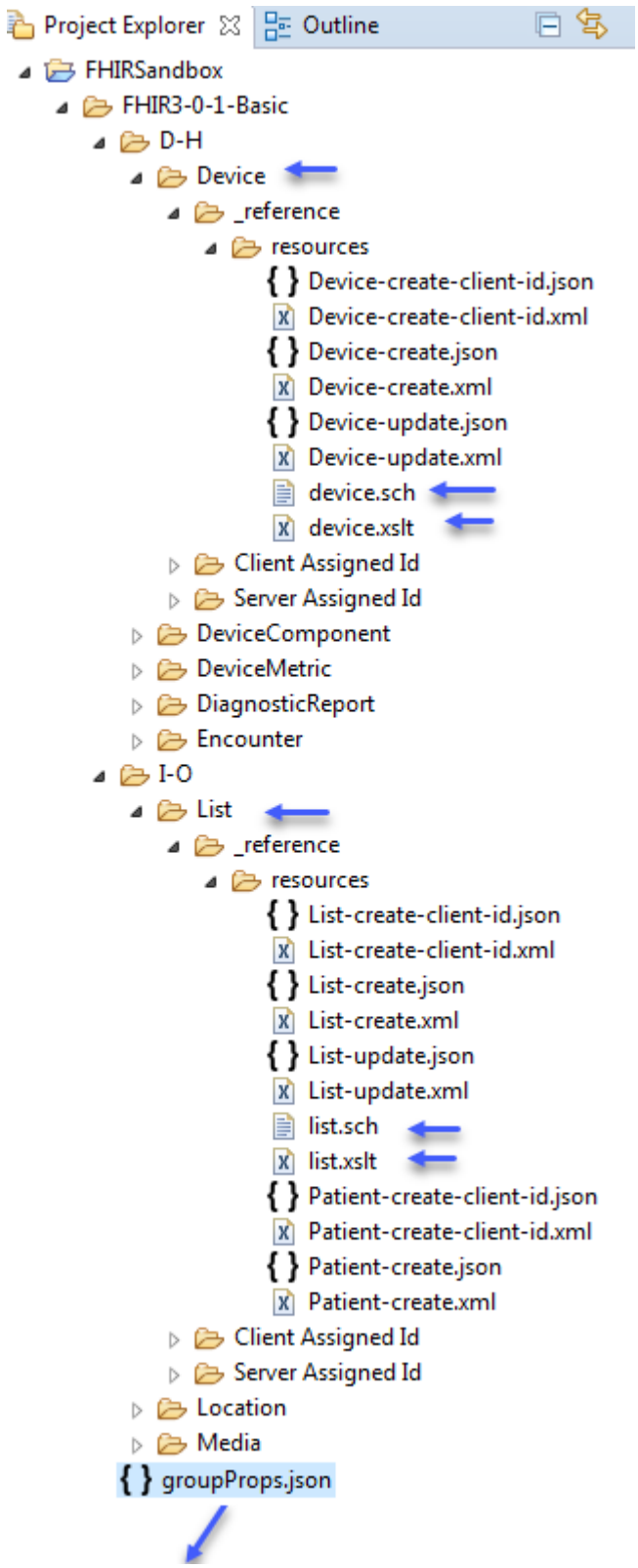
9.5.4 Parsing Exclusions

Parsing exclusions are specified using the **excludeFromParsing** key in groupProps.json. Values can be specified using regular expressions. Matching folders and files will still end up in Touchstone but will be filtered out from parsing and periodic validations.

If your folders contain Groovy, Schematron and XSLT files, for example, and you want Touchstone to **not** treat them as rule definitions, then you can specify a regular expression that matches those files in **excludeFromParsing**.

You can do the same with JSON and XML fixtures that are not even proper JSON or XML and you want to use them for negative testing.

To exclude ".sch" and ".xslt" files in all sub groups from getting parsed during upload (and treated as rule definitions):



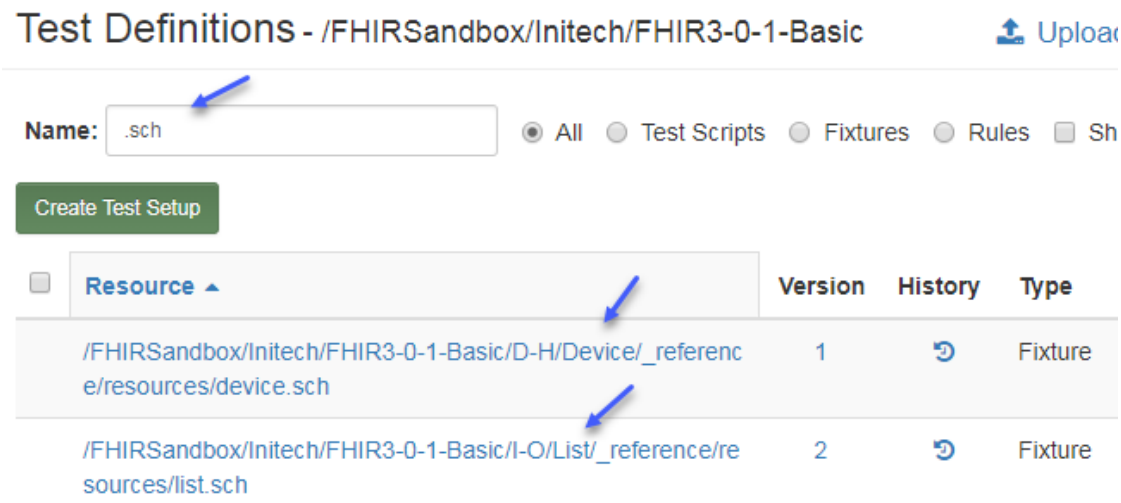

```
{
   "excludeFromParsing": [
```

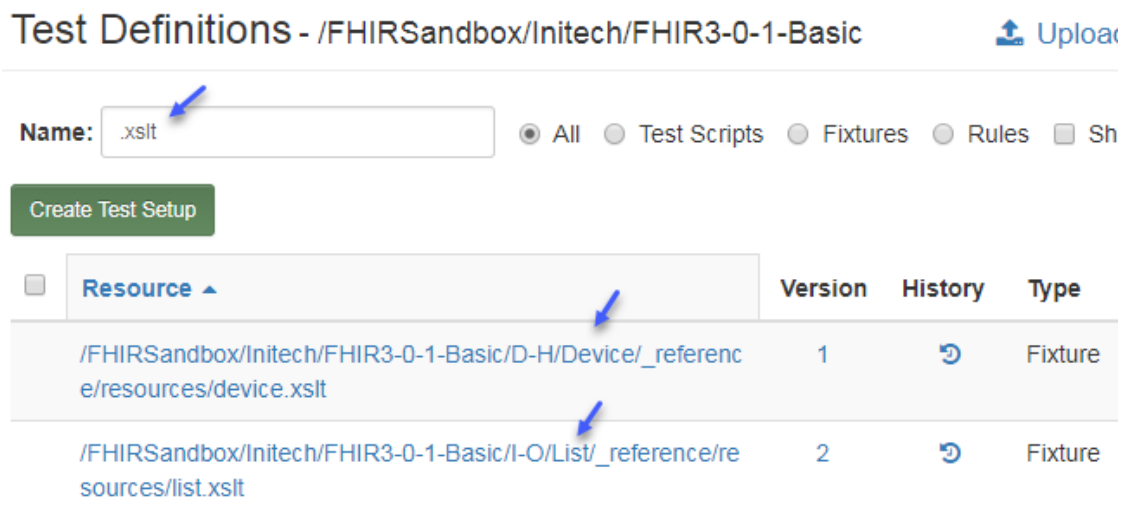
(continues on next page)

(continued from previous page)

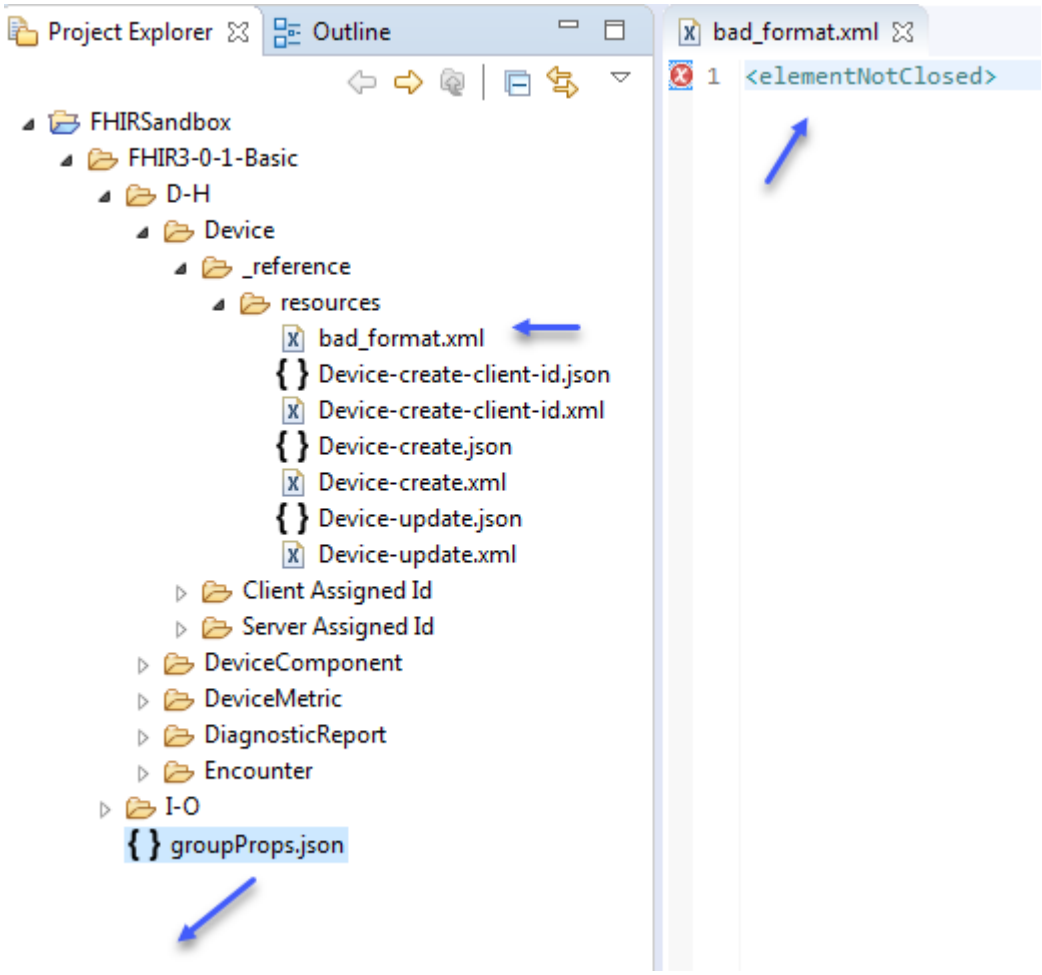
```
        ".*\\.sch",
        ".*\\.xslt"
    ]
}
```

After the FHIR3-0-1-Basic test group is uploaded, the ".sch" and ".xslt" files do get included in the upload but they will not be parsed and treated as rule definitions:



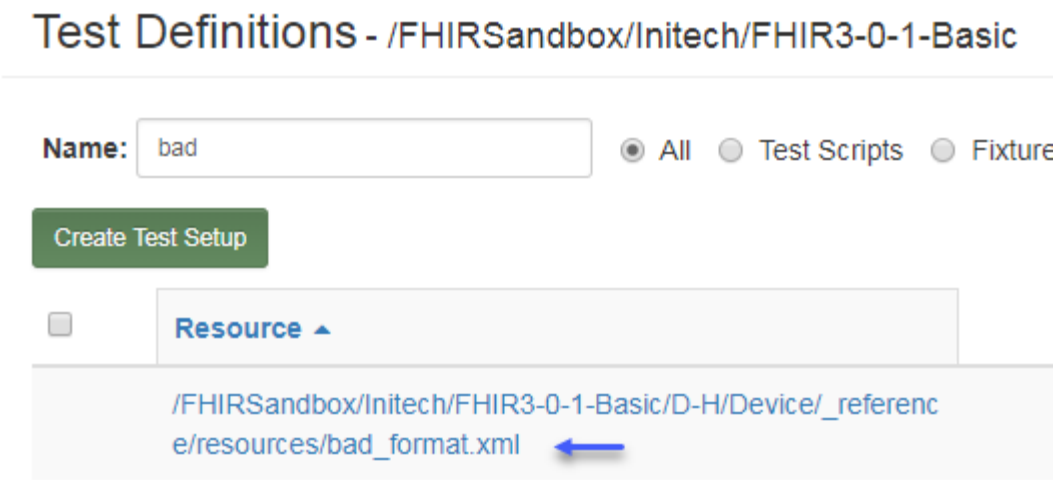


To exclude certain files from getting parsed and checked for valid JSON or XML:



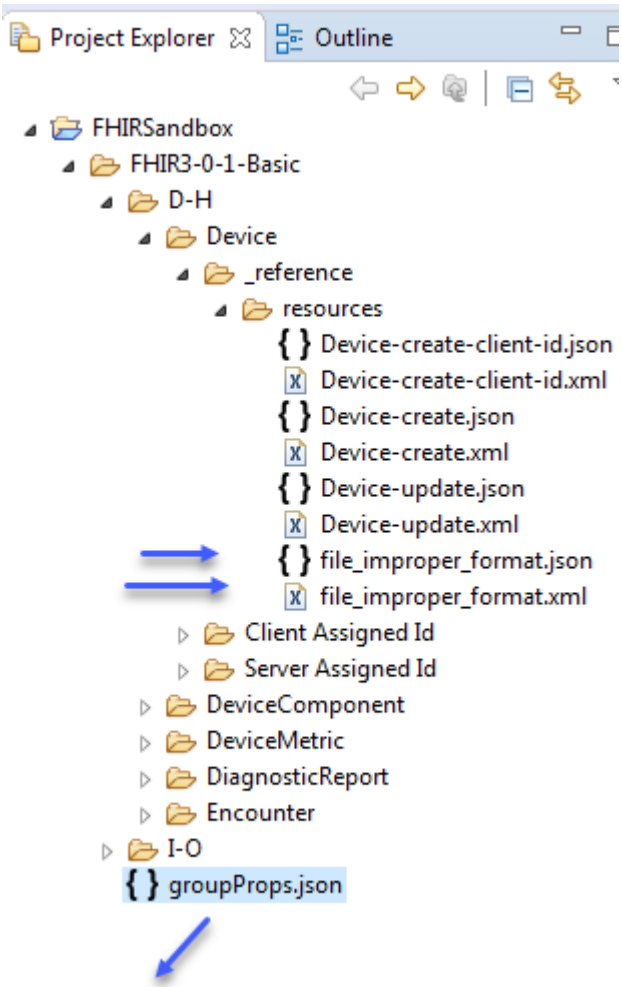

```
{
   "excludeFromParsing": [
      ".*/FHIR3-0-1-Basic/D-H/Device/_reference/bad_format.xml"
   ]
}
```

After the FHIR3-0-1-Basic test group is uploaded, the “bad_format.xml” file does get included in the upload but it will not be parsed and checked for proper XML. It will also avoid periodic validations



Warning: Minimize the number of entries in excludeFromParsing as each entry is evaluated separately during upload. Upload could take long if there are too many entries. It's better to agree upon a naming convention in your organization and use a regular expression that matches the agreed-upon convention. For example, all such invalid formats could end in "_improper_format.json" or "_improper_format.xml". Then you could exclude all these files from parsing by specifying a single regular expression that matches all those files:

To exclude all files that end in '_improper_format' from getting parsed and format-checked:



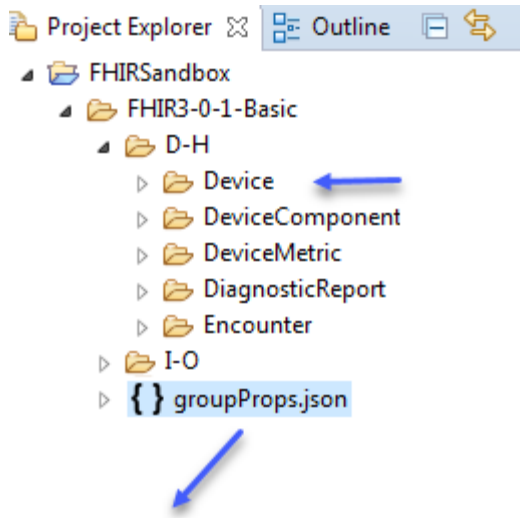

```
{
   "excludeFromParsing": [
      ".*_improper_format\\.(json|xml)"
   ]
}
```

9.5.5 Validation Exclusions

Validation exclusions are specified using the **excludeFromValidation** key in groupProps.json. Values can be specified using regular expressions. Matching folders and files will still end up in Touchstone but will be filtered out from periodic validations. JSON and XML files will be parsed though during upload and will be expected to contain proper JSON or XML.

This feature is useful if you want to avoid test definitions from getting marked for validation failures by external Validation Engine.

To exclude all test definitions in the Device test group from getting periodically validated:




```
{
   "excludeFromValidation": [
      ".*/FHIR3-0-1-Basic/D-H/Device/.*"
   ]
}
```

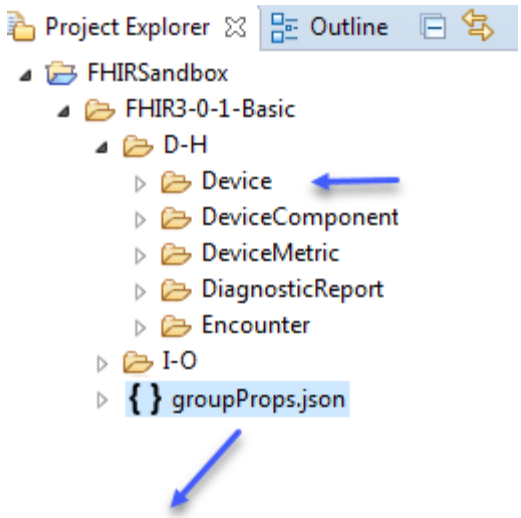
9.5.6 Conformance Exclusions

Conformance exclusions are specified using the **excludeFromConformance** key in groupProps.json. Values can be specified using regular expressions. Matching folders and files will still end up in Touchstone but will be filtered out from getting included in Conformance results. JSON and XML files will be parsed though during upload and will be expected to contain proper JSON or XML.

This feature is useful if you're testing a new operation in your test scripts and Touchstone is unable to map it to its known operation list or if you simply don't want certain test scripts from getting included in conformance results (but still want it as part of the test groups).

Note that changes to the content of such test scripts will still increment the Conformance Suite version if they're part of the test group referenced by the suite.

To exclude all test definitions in the Device test group from getting included in Conformance results:



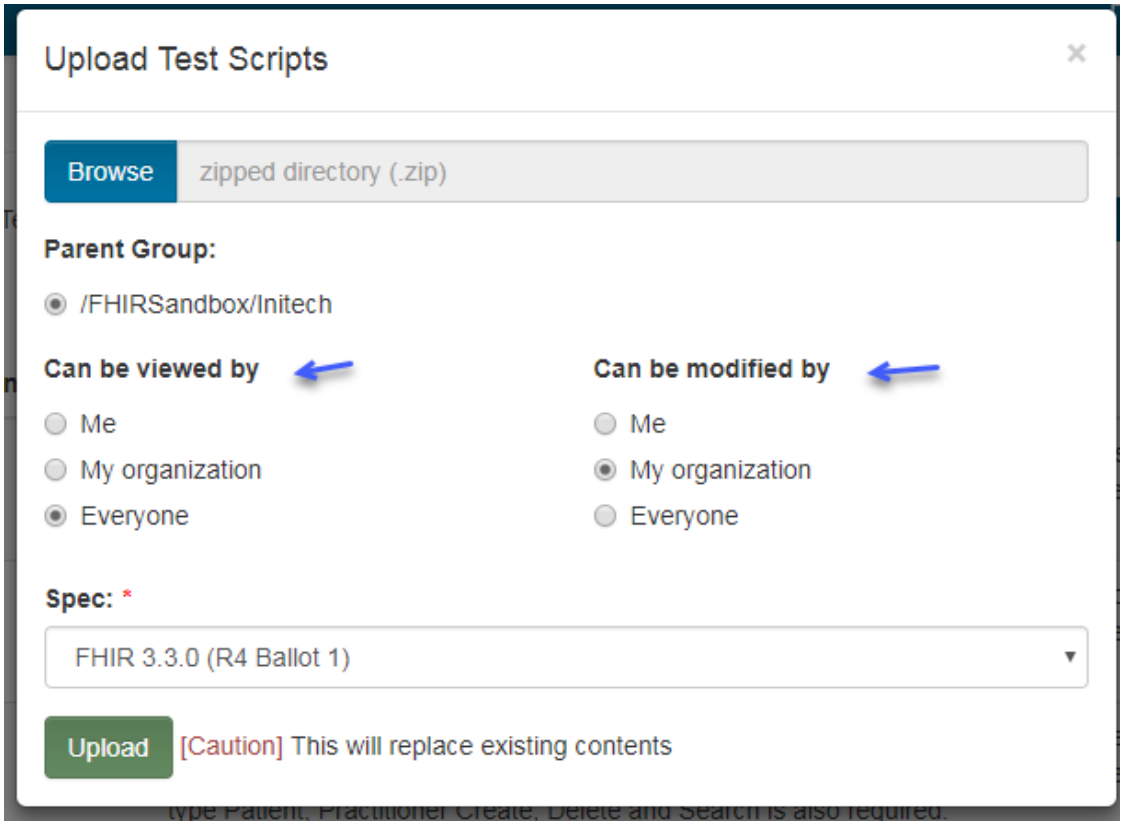

```
{
   "excludeFromConformance": [
      ".*/FHIR3-0-1-Basic/D-H/Device/.*"
   ]
}
```

9.6 Test Groups

Test Groups can be changed and deleted on the UI.

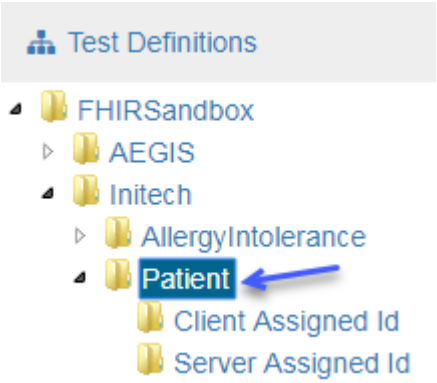
9.6.1 Access

Access rights to test groups and its descendant test definitions can be specified during upload:

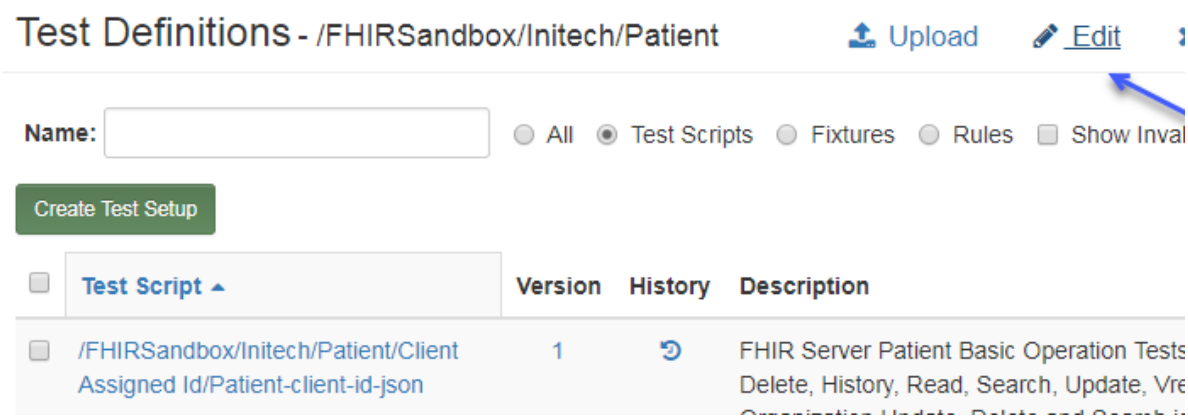


They can also be changed without having to re-upload the test group:

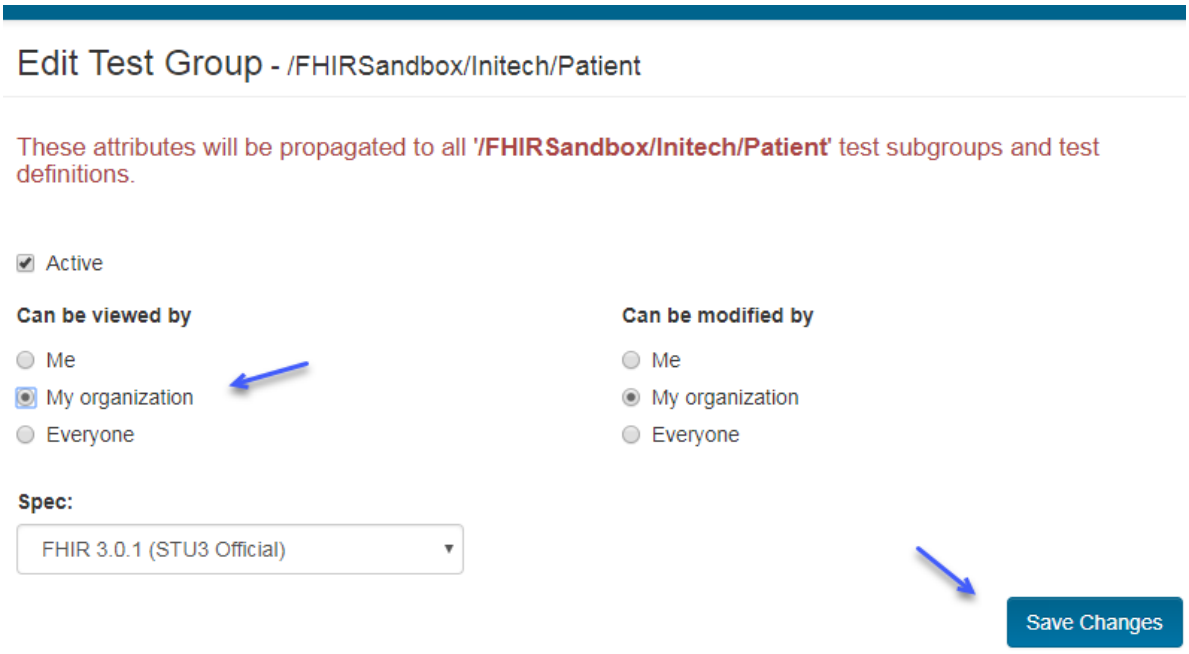
1. Select the test group



2. Click on Edit



3. Change the access and click on Save Changes

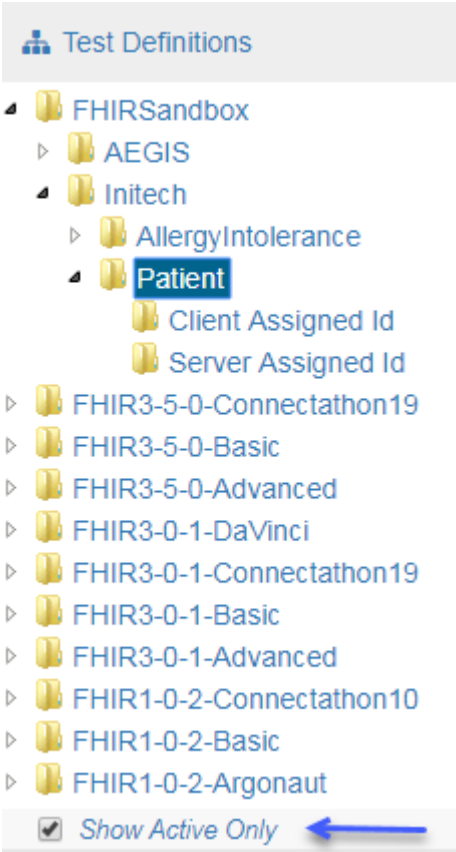


Note: These attributes will propagate to all its descendant sub groups and test definitions.

For details on controling access to test groups at the Org Group level, please refer to Test Definition Access.

9.6.2 Deactivate

Test groups can be deactivated so they don't show up in the Test Definitions tree when user has selected "Show Active Only":

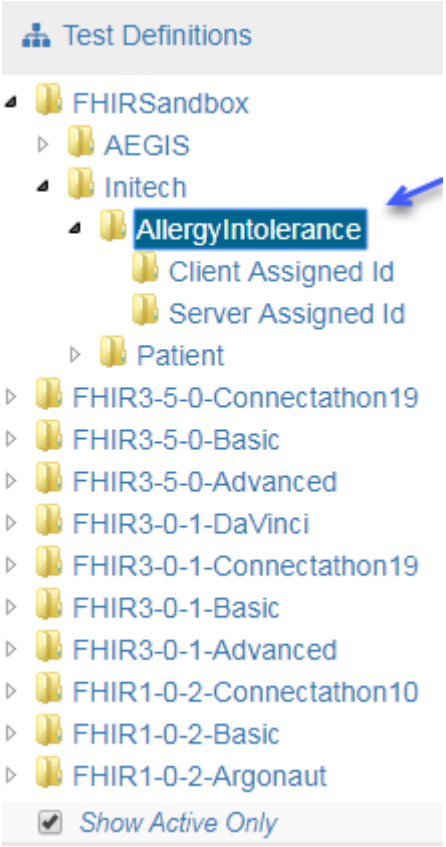


Note: It is recommended to use access rights to restrict access to test definitions that are under development and not the Active flag. If a test group is under active development and you do not want users to use the test scripts just yet, you can restrict View Access to "Me" or "My Organization", for example.

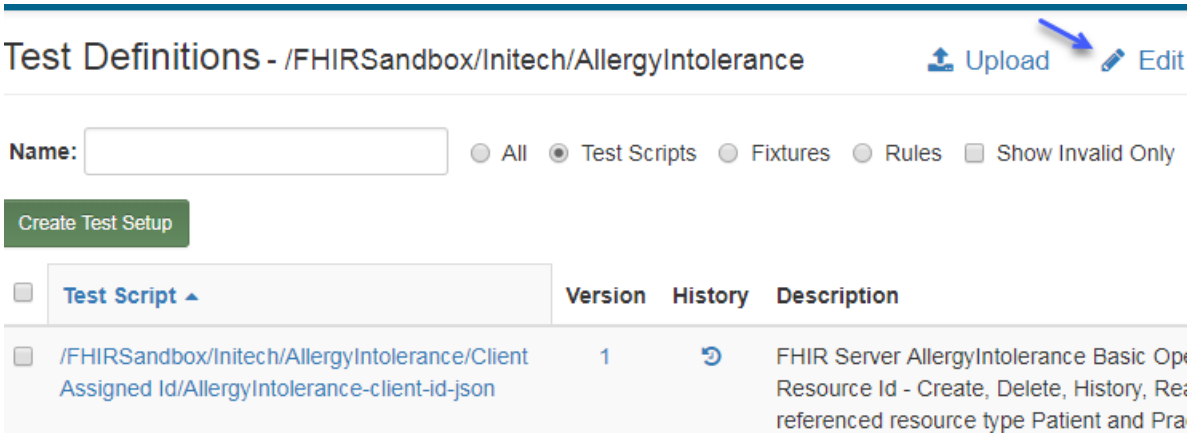
Deactivation is meant for test groups that were in use for a while but are no longer actively supported.

To deactivate a test group:

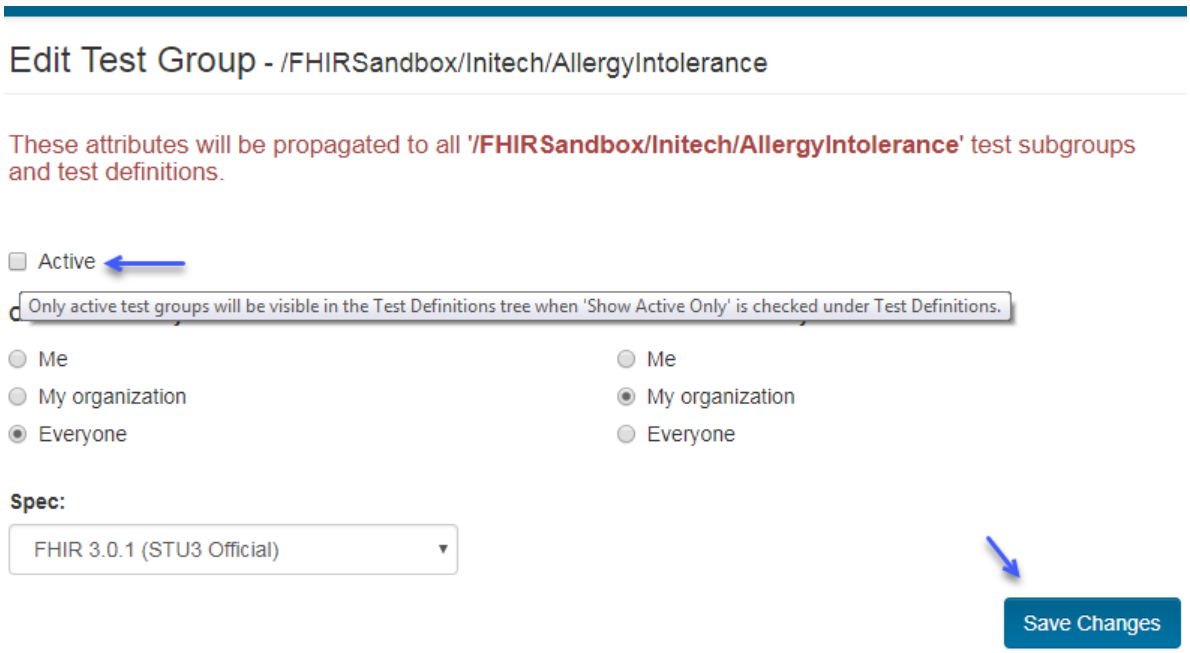
1. Select the test group



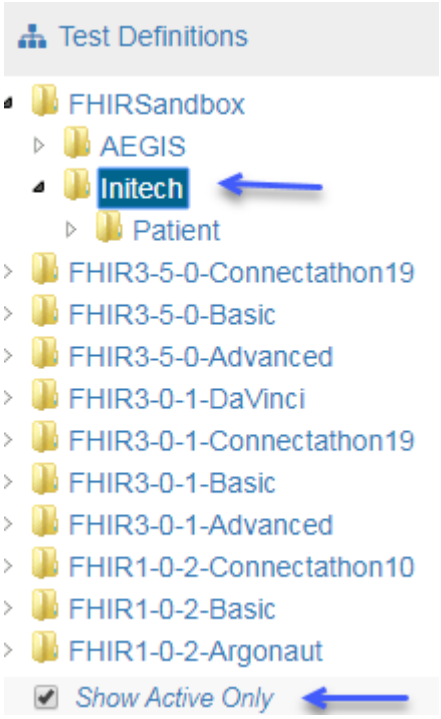
2. Click on Edit



3. Uncheck Active Flag.



4. Note that the AllergyIntolerance test group is no longer visible on the UI.

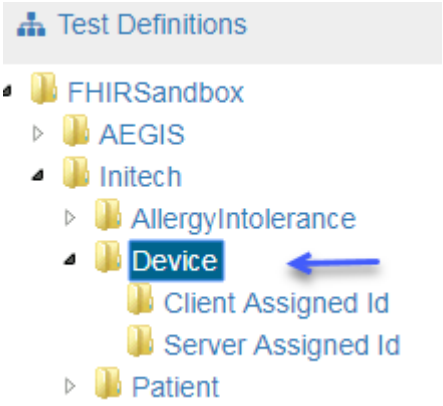


9.6.3 Deletion

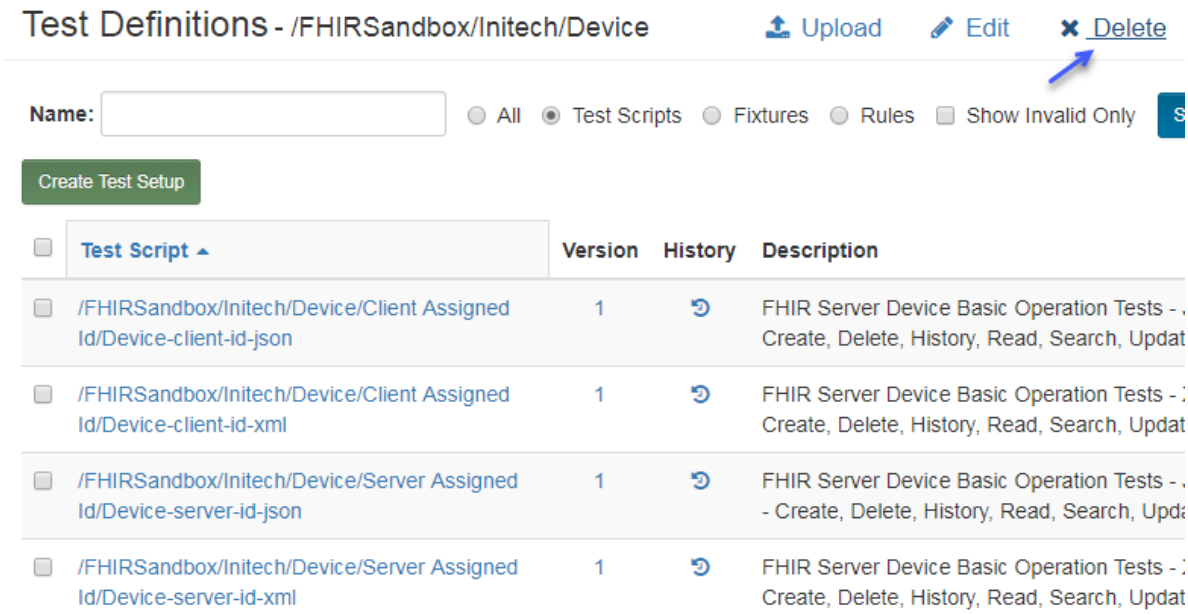
> **Warning:** This will permanently delete the test group along with all sub groups, test scripts, fixtures, etc.

Users that have write access to a test group can delete it on the UI:

1. Select the test group



2. Click on Delete



3. Confirm deletion.

9.7 Placeholders

The Touchstone Placeholders are special predefined TestScript variables and may be used where standard TestScript variables are allowed: static fixtures, dynamic fixtures, operation.params, operation.requestHeader.value, operation.url, and assert.value element values.

These Touchstone Placeholders fall into two (2) categories

- Predefined User Unique Data Values
- Functions for Dynamic Data Generation

9.7.1 Predefined User Unique Data Values

The predefined user unique data value placeholders are strings of varying lengths from 1 to 20 characters. They are defined of three (3) types:

- Character only; for example, ${C6}
- Digits only; for example, ${D9}
- Characters + Digits; for example, ${CD14}

Users may view their assigned placeholder variable values in Touchstone via the user name menu item ${} My Placeholders.

Guaranteed Value Uniqueness - All predefined user unique data value placeholder values with a character length of 6 or more are guaranteed to be unique.

9.7.2 Functions for Dynamic Data Generation

There are three (3) available functional constructs for dynamic data generation:

- CURRENTDATE and CURRENTDATETIME
- DATE and DATETIME (relative to dynamic variable)
- UUID, UUID-ST, UUID-NODASH and UUID-ST-NODASH

9.7.2.1 CURRENTDATE[TIME]

These function placeholder variables provide support for date and datetime values based on the current date (today) and current datetime (now). Optional comma separated offset arguments apply date and time arithmetic providing relative date and time generated values.

Syntax/Format ${<placeholder name>[, <datetime portion code>, <offset value>]} Where,

- <placeholder name> is CURRENTDATE or CURRENTDATETIME
- <datetime portion code> is the character indicating what portion of the date or datetime value will be offset. Valid characters are derived from the following Java SimpleDateFormat pattern string "yyMMddHHmmss"
- <offset value> is the signed integer value used to adjust the date or time
- <datetime portion code>, <offset value> may be repeated for more complex arithmetic

Examples

Placeholder Example	Description
${CURRENTDATE}	The current date (today)
${CURRENTDATETIME}	The current date and time (today)
${CURRENTDATE,d,-10}	The date 10 days before the current date
${CURRENTDATETIME,d,10}	The date and time 10 days after the current date and time
${CURRENTDATE,y,-1}	The date 1 year before the current date
${CURRENTDATETIME,H,10}	The date and time 10 hours after the current date and time

9.7.2.2 DATE[TIME]

These function placeholder variables provide support for date and datetime values based on a dynamic variable user input date or datetime value. Optional comma separated offset arguments apply date and time arithmetic providing relative date and time generated values.

Syntax/Format ${<placeholder name>, <relative value>[, <datetime portion code>, <offset value>]} Where,

- <placeholder name> is DATE or DATETIME
- **<relative value> is a dynamic variable defined in the current TestScript that holds the relative date or datetime value**
  - The dynamic variable must be defined in TestScript without path/expression; its value will be entered by the user during Test Setup
- <datetime portion code> is the character indicating what portion of the date or datetime value will be offset. Valid characters are derived from the following Java SimpleDateFormat pattern string "yyMMddHHmmss"
- <offset value> is the signed integer value used to adjust the date or time
- <datetime portion code>, <offset value> may be repeated for more complex arithmetic

Examples

Placeholder Example	Description
${DATE, medicationDate}	The value of the user entered medicationDate will be used as-is
${DATETIME, medicationDateTime}	The value of the user entered medicationDateTime will be used as-is
${DATE, medicationDate, d, -10}	The value of the user entered medicationDate minus 10 days will be used
${DATETIME, medicationDateTime, M, -1}	The value of the user entered medicationDateTime minus 1 month will be used

9.7.2.3 UUID[-ST|-NODASH|-ST-NODASH]

These function placeholder variables provide support for generation of UUID values with predefined formatting.

Syntax/Format ${<placeholder name>} Where,

- <placeholder name> is UUID, UUID-ST, UUID-NODASH or UUID-ST-NODASH

Examples

Placeholder Example	Description
${UUID}	A single generated UUID string value without the standard prefix; e.g., '3ed6eb79-fc68-443a-996f-08167f5bdef0'
${UUID-ST}	A single generated UUID string value including the standard prefix; e.g., 'urn:uuid:3ed6eb79-fc68-443a-996f-08167f5bdef0'
${UUID-NODASH}	A single generated UUID string value without the standard prefix and dash characters removed; e.g., '3ed6eb79fc68443a996f08167f5bdef0'
${UUID-ST-NODASH}	A single generated UUID string value including the standard prefix and dash characters removed; e.g., 'urn:uuid:3ed6eb79fc68443a996f08167f5bdef0'

9.7.3 Usage

The Touchstone Placeholder, both predefined values and functions, are typically defined in static fixtures used as an operation request payload; for example, create or update. Touchstone's TestScript Execution interface provides both a Raw and Resolved view of static fixtures where the pre and post test execution contents can be examined.

Raw Example - Patient.name, Patient.birthDate

```
<name>
        <use value="official"/>
        <family value="Smith${C7}"/>
        <given value="John${C6}"/>
        <birthDate value="${CURRENTDATE,d,-7}"/>
</name>
```

Resolved Example - Patient.name, Patient.birthDate

```
<name>
        <use value="official"/>
        <family value="SmithMqCERSQ"/>
        <given value="JohnXRCnsc"/>
        <birthDate value="2021-01-27"/>
</name>
```

9.8 Rule Authoring

9.8.1 Rule Basics

The TestScript's rule element can be used to reference complex validation logic that goes beyond what the basic TestScript assert element supports. As such, rules are recommended only when TestScript assert cannot be used in its basic form.

Touchstone Rules-Engine supports rules written in the following languages:

- Groovy
- XSLT
- Schematron

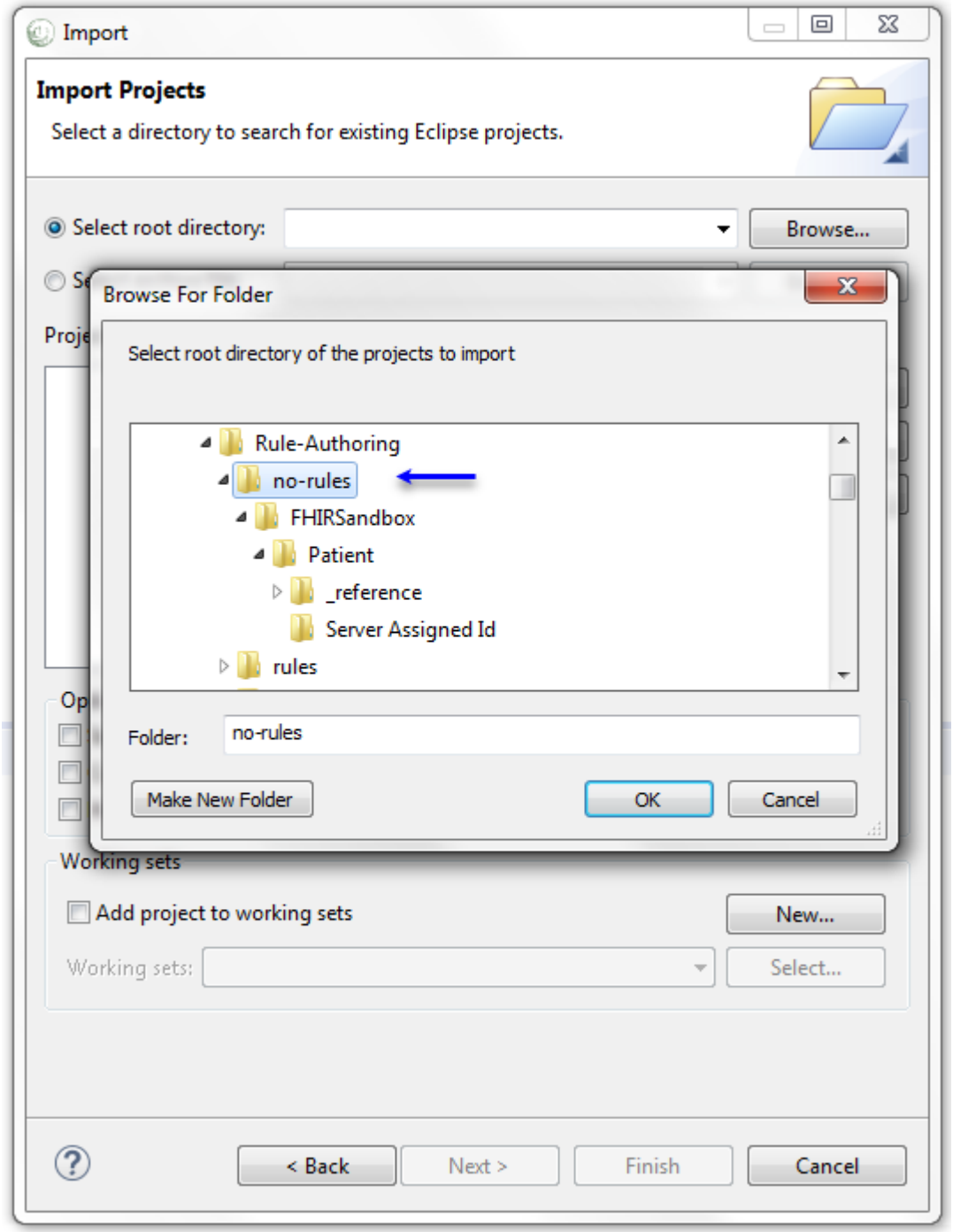
Support for additional languages may be added in the future. Unless you plan on executing test scripts against a test system that only supports XML, it is highly recommended to write rules in Groovy as XSLT and Schematron rules can only be evaluated against requests and responses whose content is in XML while Groovy supports JSON as well.

The examples in this guide will be in Groovy. For more information on how to write rules in XSLT and Schematron, please refer to XSLT and Schematron.

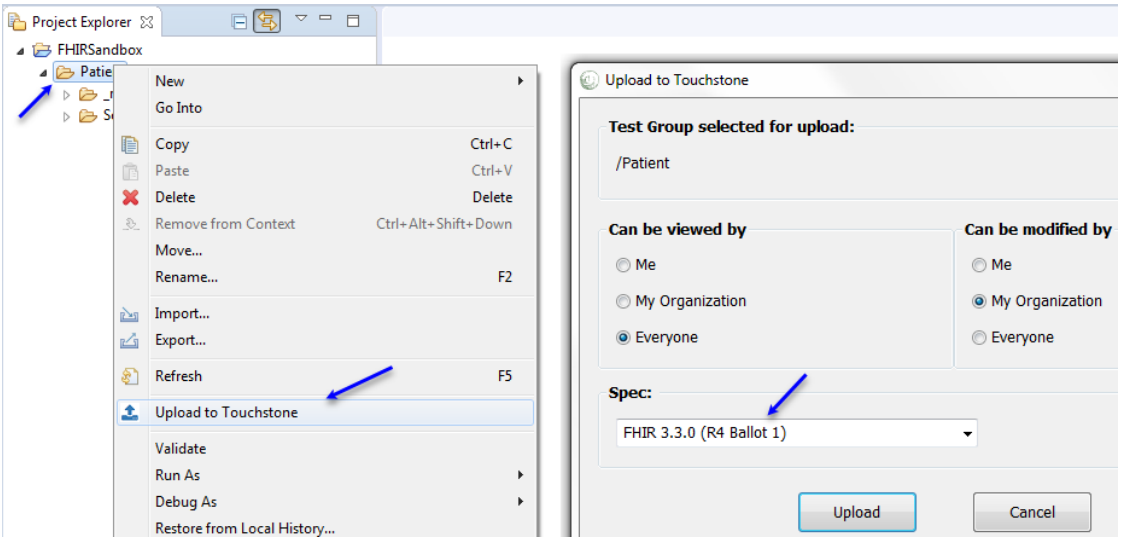
9.8.1.1 Definition

We will start with a simple test script that has no rules and gradually add rules. You can download all of the examples used in this section from here.

Open an instance of TestScript Editor and import the project in the example "**no-rules**" directory within the zip file:



Upload the Patient folder to Touchstone:



Execute the newly uploaded script successfully in Touchstone. You can use WildFHIR 3.3.0 server as the target server.

The first operation in `RegisterNewPatient` test of Patient-server-id-json.xml test script is a create operation followed by a basic assertion:

```
<action>
   <operation>
      <type>
         <system value="http://hl7.org/fhir/testscript-operation-codes"/>
         <code value="create"/>
      </type>
      <resource value="Patient"/>
      <description value="Create patient with server assigned resource id."/>
      <accept value="json"/>
      <contentType value="json"/>
      <encodeRequestUrl value="true"/>
      <sourceId value="patient-create"/>
   </operation>
</action>
<action>
   <assert>
      <description value="Confirm that the returned HTTP status is 200(OK) or␣
↪201(Created)."/>
```

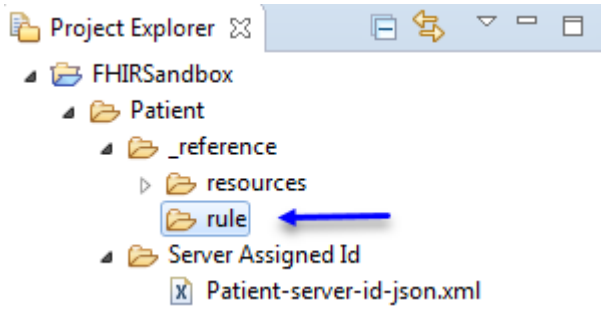
(continues on next page)

(continued from previous page)

```
      <direction value="response"/>
      <operator value="in"/>
      <responseCode value="200,201"/>
      <warningOnly value="false"/>
   </assert>
</action>
```

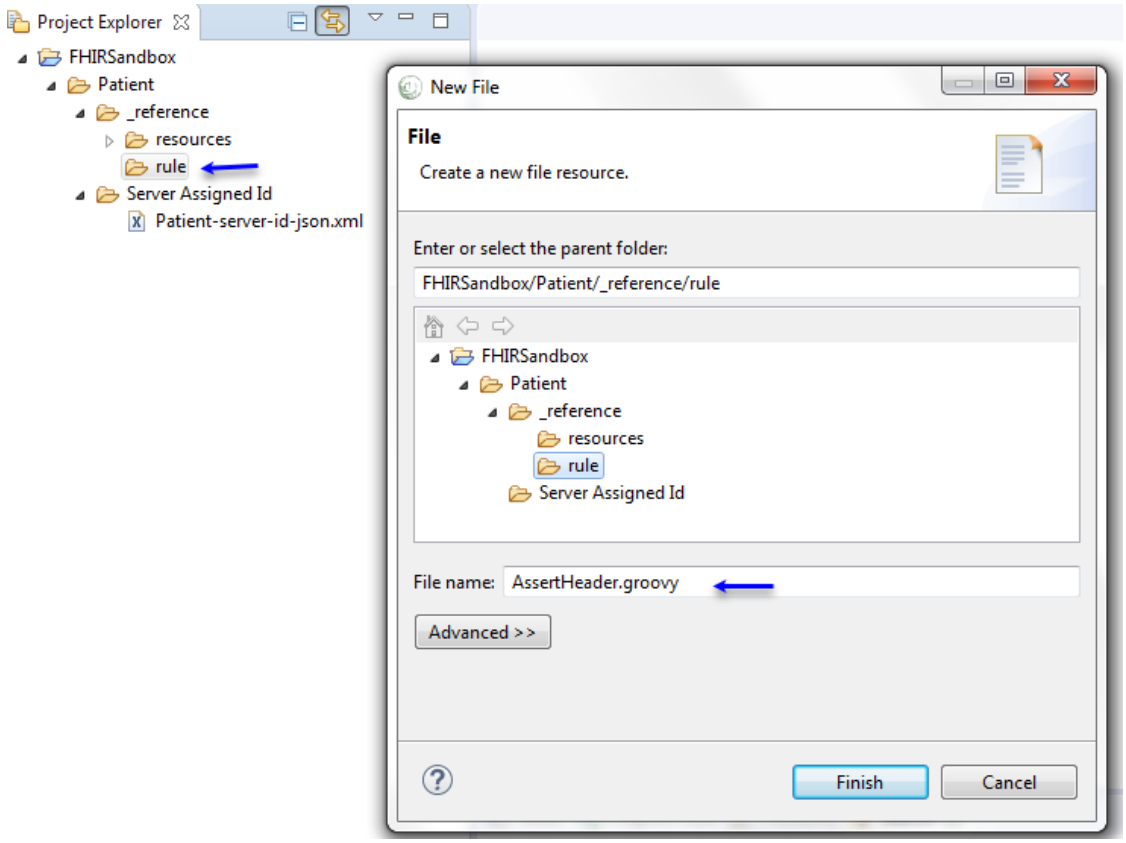
According to the FHIR specification, if a server supports versioning then it should return an ETag header with the versionId in the create operation response. Although the basic TestScript assertion supports verification of arbitrary response headers, we will create a rule to perform this verification for demonstration purposes.

Create a folder called 'rule' under the _reference folder:



This folder will host all the rule definitions that are specific to the Patient test group.

Create a new file called 'AssertHeader.groovy' under this folder:



Keep the rule contents empty for now.

9.8.1.2 Declaration

To use a rule in a test script assert, you must first declare the Touchstone IG in the test script's metadata:

```
 <meta>
     <profile value="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript"/>
 </meta>

.. warning:: If this metadata is not in your testscript Touchstone cannot evaluate any␣
↪rule/ruleset asserts.
```

Then you can declare the `assertETag` rule before the url definition in `Patient-server-id-json.xml` test script as follows:

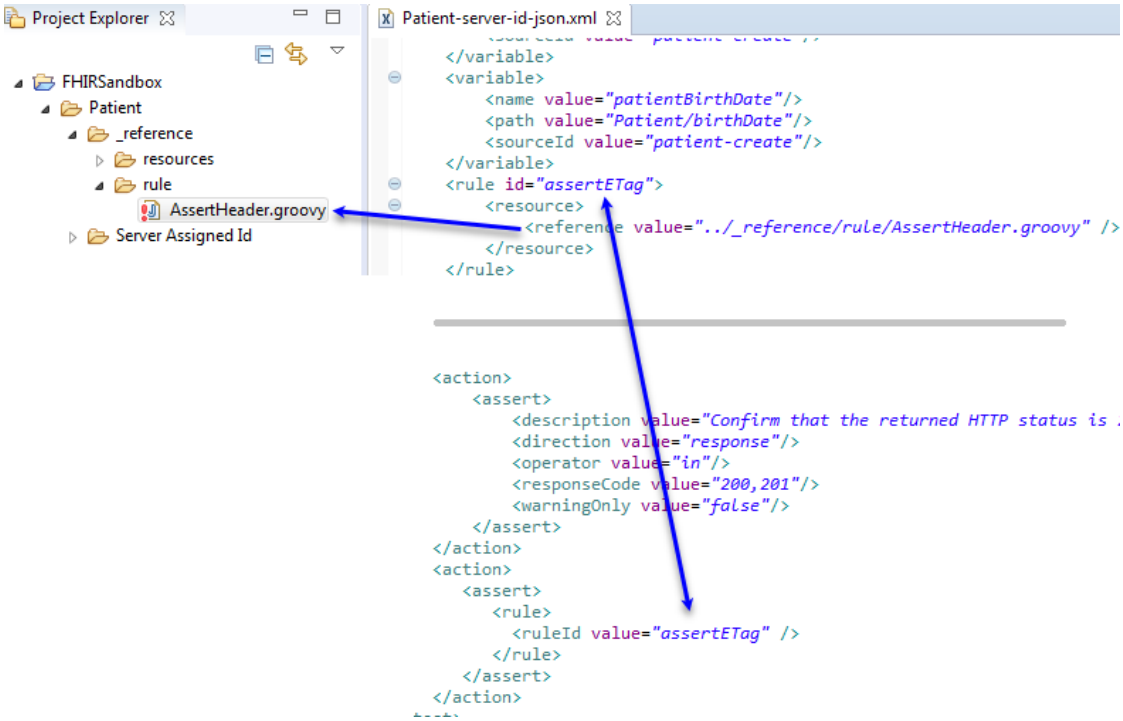
```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/
↪testscript-rule">
    <extension url="ruleId">
        <valueId value="assertETag"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/rule/AssertHeader.groovy"/>
    </extension>
</extension>
```

9.8.1.3 Assertion

We can now use the rule in rule assertions. Add the following rule assertion to the `RegisterNewPatient` test in `Patient-server-id-json.xml` test script:

```
<assert>
    <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-assert-rule">
        <extension url="ruleId">
            <valueId value="assertETag"/>
        </extension>
    </extension>
    <warningOnly value="false" />
</assert>
```

Notice we're giving a relative path to the actual rule definition. The rule id `assertETag` has to match the one used in the rule assertion. Defining the rule at the top of the test script allows us to reuse this rule in many tests within this test script.



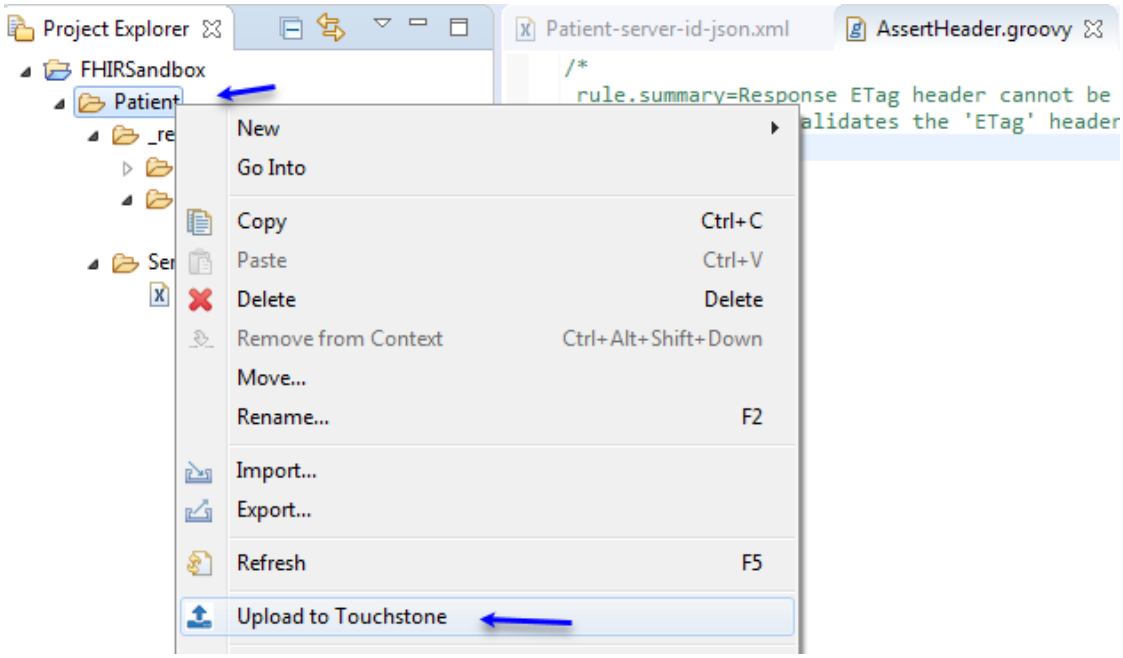
9.8.1.4 Summary and Description

Let's now add some content to the AssertHeader.groovy rule.

The summary and description lines communicate the intent of the rule assertion to the end-user. They are declared in Groovy comments at the top of the rule definition as follows:

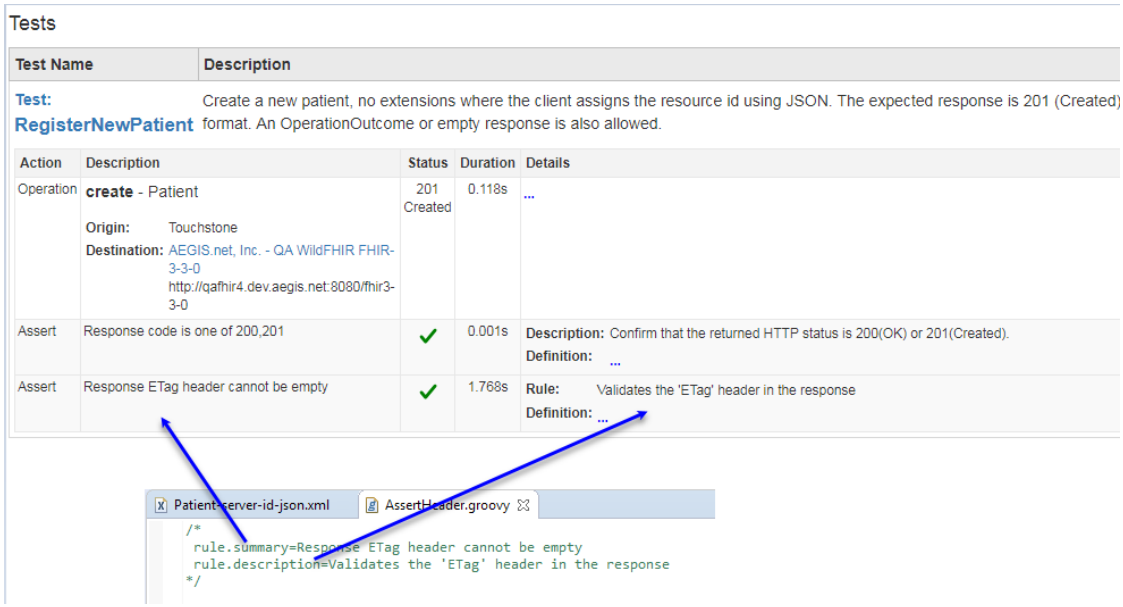
```
/*
 rule.summary=Response ETag header cannot be empty
 rule.description=Validates the 'ETag' header in the response
*/
```

Go ahead and upload the Patient folder to Touchstone:



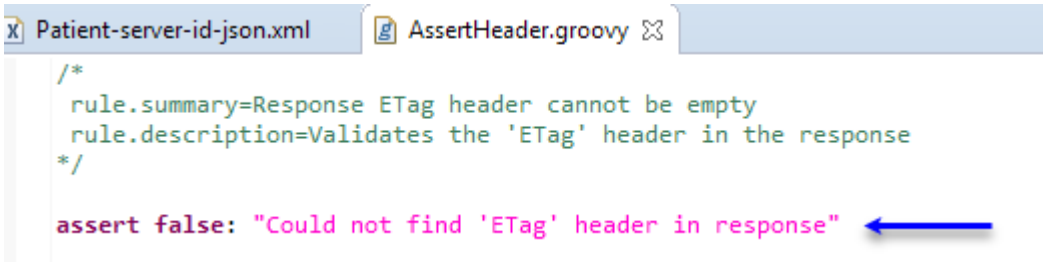
Execute the Patient-server-id-json test script on the UI:

Here's how the summary and description get displayed on the TestScript Execution screen:



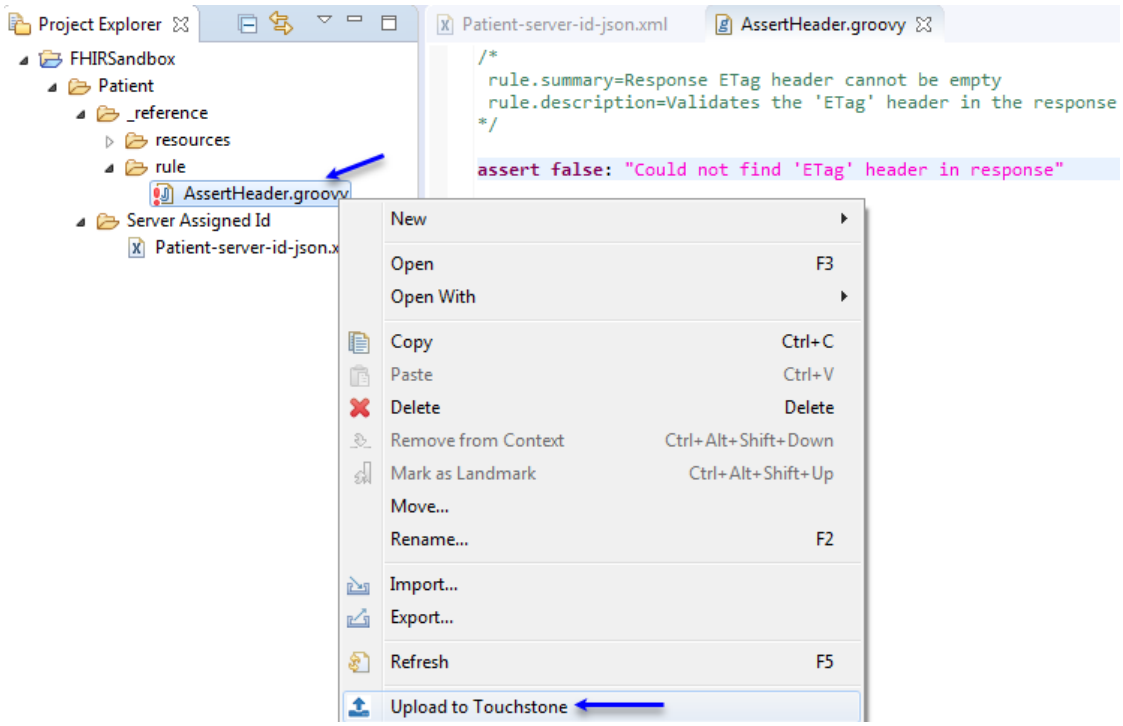
The rule did not have any logic. So the rule assertion passed.

Add the following assert to the rule content to make it fail unconditionally:

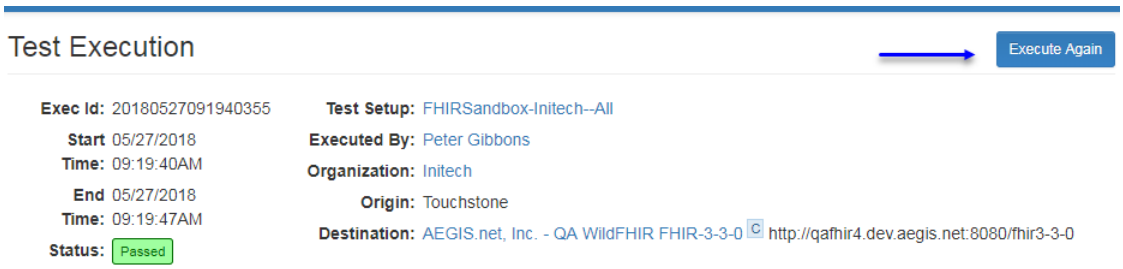


An error is raised with the message specified after the colon (i.e. "Could not find 'ETag' header in response") if the assertion fails. The "assert false" call is an unconditional failure; so we must get an error with this message.

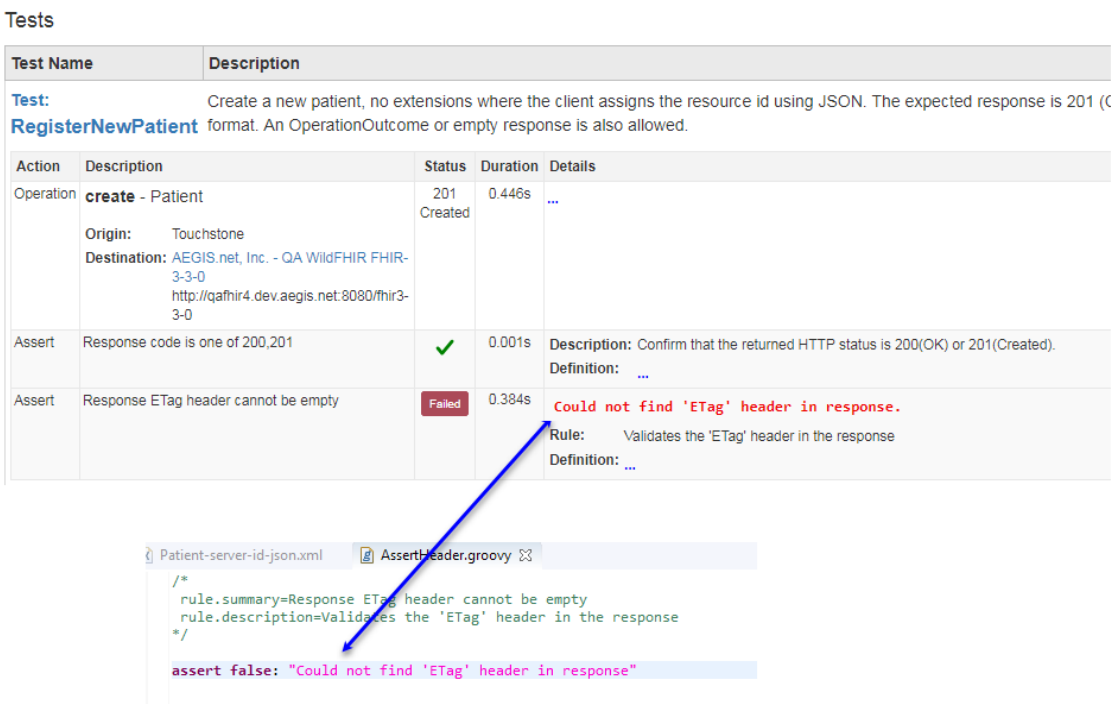
We only changed the groovy rule. Upload the groovy rule to Touchstone:



Re-execute the same test execution on Touchstone UI:



Notice the correlation between the error message in the rule and what's displayed on the UI:



Change the assert in the groovy rule to evaluate the ETag header in the response correctly:

```
assert response.header('ETag').isNotEmpty(): "Could not find 'ETag' header in response"
```

We are using the `response` binding that represents the HTTP response from the target server to grab the actual ETag header received from the server. *Bindings* are covered in the next section.

Upload the groovy rule to Touchstone and re-execute the test execution. This time, the assertion passes because the server does indeed return an ETag in the response header:

Action	Description	Status	Duration	Details
Operation	**create** - Patient **Origin:** Touchstone **Destination:** AEGIS.net, Inc. - QA WildFHIR FHIR-3-3-0 http://qafhir4.dev.aegis.net:8080/fhir3-3-0	201 Created	0.140s	**Type:** create **Resource:** Patient **URL:** http://qafhir4.dev.aegis.net:8080/fhir3-3-0/Patient **Description:** Create patient with server assigned resource id. **Definition:** ... **Request:** **Method:** POST **Path:** http://qafhir4.dev.aegis.net:8080/fhir3-3-0/Patient **Headers:** **Accept** application/fhir+json;charset=UTF-8 **Content-Type** application/fhir+json;charset=UTF-8 **Request:** Body **Response:** **Status:** HTTP/1.1 201 Created **Headers:** **Connection** keep-alive **Content-Length** 2799 **Content-Location** http://qafhir4.dev.aegis.net:8080/0/Patient/42c837b864624ea89faa13fc **Content-Type** application/fhir+json; charset=utf **Date** Sun, 27 May 2018 13:39:42 GMT **ETag** W/"1" **Last-Modified** Sun, 27 May 2018 13:39:43GMT **Location** http://qafhir4.dev.aegis.net:8080/0/Patient/42c837b864624ea89faa13fc **Server** WildFly/8 **X-Powered-By** Undertow/1 **Response:** Body
Assert	Response code is one of 200,201	✓	0.001s	**Description:** Confirm that the returned HTTP status is 200(OK) or 201(Created). **Definition:** ...
Assert	Response ETag header cannot be empty	✓	0.699s	**Rule:** Validates the 'ETag' header in the response **Definition:** ...

While it's perfectly valid to use the Groovy `assert` and construct the error messages yourself (e.g. "Could not find 'ETag' header in response"), the Touchstone Rules API provides helper methods that relieves rule authors from constructing these error messages and thereby keeps them consistent across many rule definitions. These helper methods are offered on many of the binding variables and come in the form of "assertXXXX" e.g. `assertHeaderEquals`, `assertStatusCodeEquals`, `assertContentTypeEquals`, etc. For full listing, please refer to Rules API Reference.

Let's change the assertion in AssertHeader.groovy to the following:

```
response.assertHeaderNotEmpty('ETag')
```

Upload the rule and re-execute. The results should be same as before.

9.8.1.5 Bindings

We used the `response` binding to make a header assertion in the previous section. There are many other bindings that the Rules-Engine provides access to within a Groovy rule template. They are touched upon below and are covered in more detail in Rules API Reference.

Binding	Alternative bindings	Description
exchange		The exchange taking place between a client (Touchstone or Test system) and a server Test System in Touchstone
request	**exchange.request**	The request message sent by a client (either Touchstone or another Test System in Touchstone).
requests		Map of Testscript.operation.requestId to request. If operation.requestId were specified as 'create-request', then the request could be fetched within the rule using **requests.get('create-request')**. See Multiple Assertions.
response	**exchange.response**	The response received from the server Test System.
responses		Map of Testscript.operation.responseId to request. If operation.responseId were specified as 'create-response', then the response could be fetched within the rule using **responses.get('create-response')**. See Multiple Assertions.
requestHeaders	**request.headers**	The headers of the request.
responseHeaders	**response.headers**	The headers of the response.
requestBody	**request.body**	The payload of the request.
responseBody	**response.body**	The payload of the response.
responseCode	**statusCode**,	e.g. 200. See RFC Status Codes and Touchstone

9.8.2 Parameters

We checked for the presence of 'ETag' header earlier in the AssertHeader.groovy rule. If we wanted to also check for the presence of 'LastModified' header, we could create another rule definition and have the following assert in it:

```
response.assertHeaderNotEmpty('LastModified')
```

The code is the same as we used earlier. Only the header value is different. Creating a separate rule definition for each value that you expect in an exchange would lead to an explosive number of rule definitions in the system with lots of rule logic duplication. This is hard to maintain.

It is better to have a single rule definition (AssertHeader.groovy) and pass the header as a parameter.

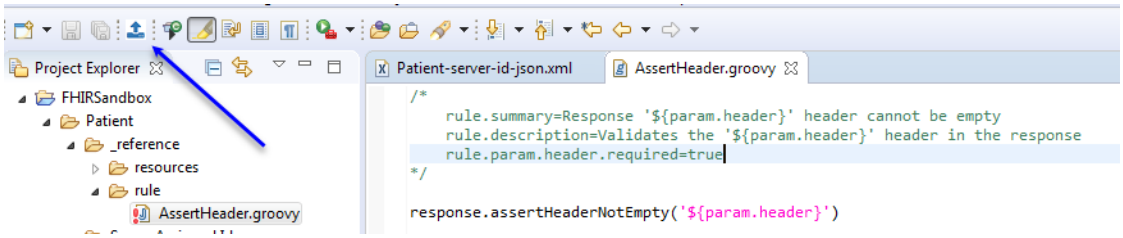
9.8.2.1 Definition

Parameters are defined in the comments section of the Groovy rule definition after the description:

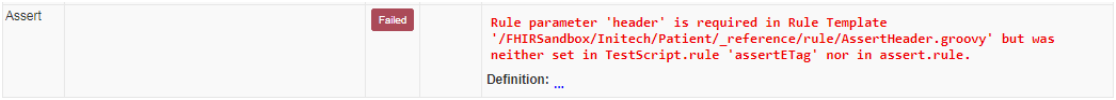
```
/*
   rule.summary=Response '${param.header}' header cannot be empty
   rule.description=Validates the '${param.header}' header in the response
   rule.param.header.required=true
*/

response.assertHeaderNotEmpty('${param.header}')
```

We're defining the 'header' parameter and making it required. The system will check what rule parameters are required by a rule definition and will ensure that the test script has supplied them before evaluating the rule. We're also using the parameter in the summary and description as '${param.header}' so the summary and description on the TestScript Execution screen is based on the parameter supplied.

Replace the contents of AssertHeader.groovy with the code above, and re-upload and re-execute.



You should get the following error:



That's because we haven't supplied the 'header' parameter from the test script yet. We'll do that next.

9.8.2.2 Supplying params

According to the specification, we could supply the parameter in two ways in the test script:

1. At the top of the test script within the rule declaration:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-rule">
    <extension url="ruleId">
        <valueId value="assertETag"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/rule/AssertHeader.groovy"/>
    </extension>
    <extension url="param">
        <extension url="name">
            <valueString value="header"/>
        </extension>
        <extension url="value">
            <valueString value="ETag"/>
        </extension>
    </extension>
</extension>
```

2. Within the rule-assertion in a test:

```
<assert>
    <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-assert-rule">
        <extension url="ruleId">
            <valueId value="assertResponseCode"/>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="header"/>
            </extension>
            <extension url="value">
                <valueString value="ETag"/>
            </extension>
        </extension>
    </extension>
    <warningOnly value="false" />
</assert>
```

You can use option 1 or 2 or both. Option 2 is useful when you need the parameter values in rule-assertion to override the ones supplied within the rule declaration. Let's use the first option i.e. declare it within the rule declaration:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/
↪testscript-rule">
    <extension url="ruleId">
        <valueId value="assertETag"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/rule/AssertHeader.groovy"/>
    </extension>
    <extension url="param">
        <extension url="name">
            <valueString value="header"/>
        </extension>
        <extension url="value">
            <valueString value="ETag"/>
        </extension>
    </extension>
</extension>
```

Upload the test script and re-execute it. You should get the following result:

Assert	Response code is one of 200,201	✓	0.000s	**Description:** Confirm that the returned HTTP status is 200(OK) or 201(Created). **Definition:** ...
Assert	Response 'ETag' header cannot be empty	✓	0.333s	**Rule:** Validates the 'ETag' header in the response **Definition:** ...

We can now easily add the rule assertion for 'Last-Modified' header without modifying AssertHeader.groovy rule definition.

Add the following rule declaration to the test script after `assertETag`:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/
↪testscript-rule">
    <extension url="ruleId">
        <valueId value="assertLastModified"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/rule/AssertHeader.groovy"/>
    </extension>
    <extension url="param">
        <extension url="name">
            <valueString value="header"/>
        </extension>
        <extension url="value">
            <valueString value="Last-Modified"/>
        </extension>
    </extension>
</extension>
```

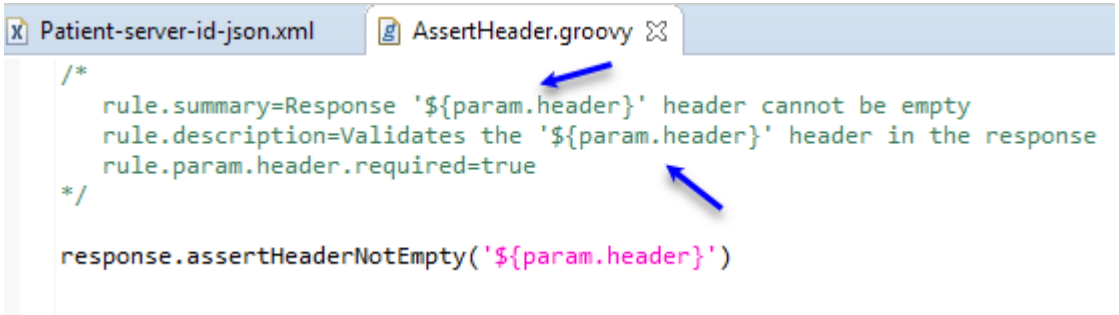
Add the following rule-asseriton to the test script after all the assertions:

```
<assert>
    <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-assert-rule">
        <extension url="ruleId">
            <valueId value="assertLastModified"/>
        </extension>
    </extension>
    <warningOnly value="false" />
</assert>
```

Re-upload and re-execute. You should get the results below. Notice how the summaries and descriptions are customized for each parameter.

Assert	Response code is one of 200,201	✓	0.000s	**Description:** Confirm that the returned HTTP status is 200(OK) or 201(Created). **Definition:** ...
Assert	Response 'ETag' header cannot be empty	✓	0.332s	**Rule:** Validates the 'ETag' header in the response **Definition:** ...
Assert	Response 'Last-Modified' header cannot be empty	✓	0.371s	**Rule:** Validates the 'Last-Modified' header in the response **Definition:** ...

That's because we used parameters in the summary and description:



9.8.2.3 Operator parameter

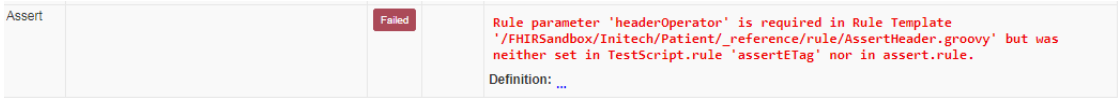
You can make the AssertHeader rule even more generic by passing the operator that the header value needs to be evaluated with.

Replace the contents of AssertHeader.groovy with the following:

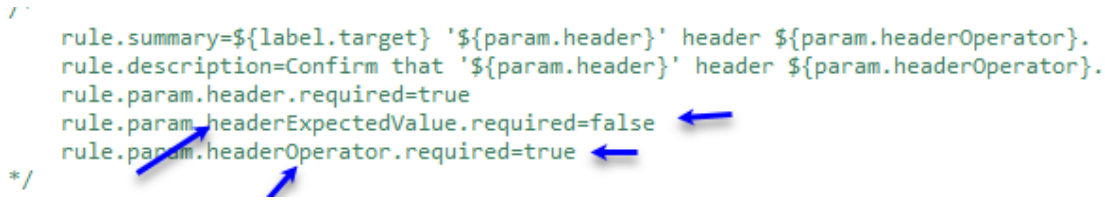
```
/*
   rule.summary=${label.target} '${param.header}' header ${param.headerOperator}.
   rule.description=Confirm that '${param.header}' header ${param.headerOperator}.
   rule.param.header.required=true
   rule.param.headerExpectedValue.required=false
   rule.param.headerOperator.required=true
*/

targetMessage.assertHeader(param.header, param.headerExpectedValue, param.
↪headerOperator);
```

See Bindings for explanation on what `targetMessage` means.

Re-upload and re-execute. You should get the following error:



Notice that there are two new parameters that can be supplied from the test script. One is required and the other optional:



Replace the rule references in `Patient-server-id-json.xml` test script with the changes below. Notice the addition of `headerOperator` parameter to both rule declarations:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/
↪testscript-rule">
    <extension url="ruleId">
        <valueId value="assertETag"/>
```

(continues on next page)

(continued from previous page)

```
    </extension>
    <extension url="path">
        <valueString value="../_reference/rule/AssertHeader.groovy"/>
    </extension>
    <extension url="param">
        <extension url="name">
            <valueString value="header"/>
        </extension>
        <extension url="value">
            <valueString value="ETag"/>
        </extension>
    </extension>
    <extension url="param">
        <extension url="name">
            <valueString value="headerOperator"/>
        </extension>
        <extension url="value">
            <valueString value="notEmpty"/>
        </extension>
    </extension>
</extension>
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/
↪testscript-rule">
    <extension url="ruleId">
        <valueId value="assertLastModified"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/rule/AssertHeader.groovy"/>
    </extension>
    <extension url="param">
        <extension url="name">
            <valueString value="header"/>
        </extension>
        <extension url="value">
            <valueString value="Last-Modified"/>
        </extension>
    </extension>
    <extension url="param">
        <extension url="name">
            <valueString value="headerOperator"/>
        </extension>
        <extension url="value">
            <valueString value="notEmpty"/>
        </extension>
    </extension>
</extension>
```

Re-upload and re-execute. You should see the results below:

Assert	Response code is one of 200,201	✓	0.000s	**Description:** Confirm that the returned HTTP status is 200(OK) or 201(Created). **Definition:** ...
Assert	Response 'ETag' header is present.	✓	0.454s	**Rule:** Confirm that 'ETag' header is present. **Definition:** ...
Assert	Response 'Last-Modified' header is present.	✓	0.318s	**Rule:** Confirm that 'Last-Modified' header is present. **Definition:** ...

You can refer to Header Rule API for details on header assertions.

9.8.2.4 Expected parameter

Let's replace the existing assertion for response code in `Patient-server-id-json.xml` test script with an assertion rule. This is not recommended practice as it's best to resort to rule-assertions when existing TestScript asserts do not meet your needs. We're doing this for demonstration purposes.

Create a new rule definition called **AssertResponseCode.groovy** and copy the following contents into it:

```
/*
   rule.summary=Response status code ${param.responseCodeOperator} ${param.responseCode}.
   rule.description=Confirm that response status code ${param.responseCodeOperator} $
↪{param.responseCode}.
   rule.param.responseCode.required=true
   rule.param.responseCodeOperator.required=true
*/

targetMessage.assertResponseCode(param.responseCode, param.responseCodeOperator)
```

The **responseCode** and **responseCodeOperator** parameters above are required and have to be supplied by the test script. The **responseCode** is the expected value in the response.

Add the following rule declaration before the assertETag rule declaration:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/
↪testscript-rule">
    <extension url="ruleId">
        <valueId value="assertResponseCode"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/rule/AssertResponseCode.groovy"/>
    </extension>
</extension>
```

Replace the existing rule assert for response code with the following:

```
<assert>
    <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-assert-rule">
        <extension url="ruleId">
            <valueId value="assertResponseCode"/>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="responseCode"/>
            </extension>
```

(continues on next page)

(continued from previous page)

```
            <extension url="value">
                <valueString value="200,201"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="responseCodeOperator"/>
            </extension>
            <extension url="value">
                <valueString value="in"/>
            </extension>
        </extension>
    </extension>
    <warningOnly value="false" />
</assert>
```

Notice that this time, we're supplying the parameters with the rule assert instead of the rule declaration.

Re-upload and re-execute. You should see the following results:

Assert	Response status code is one of 200,201.	✓	0.323s	**Rule:** Confirm that response status code is one of 200,201. **Definition:** ...
Assert	Response 'ETag' header is present.	✓	0.319s	**Rule:** Confirm that 'ETag' header is present. **Definition:** ...
Assert	Response 'Last-Modified' header is present.	✓	0.326s	**Rule:** Confirm that 'Last-Modified' header is present. **Definition:** ...

You can refer to Response Code Rule API for details on response code assertions.

9.8.3 Ruleset

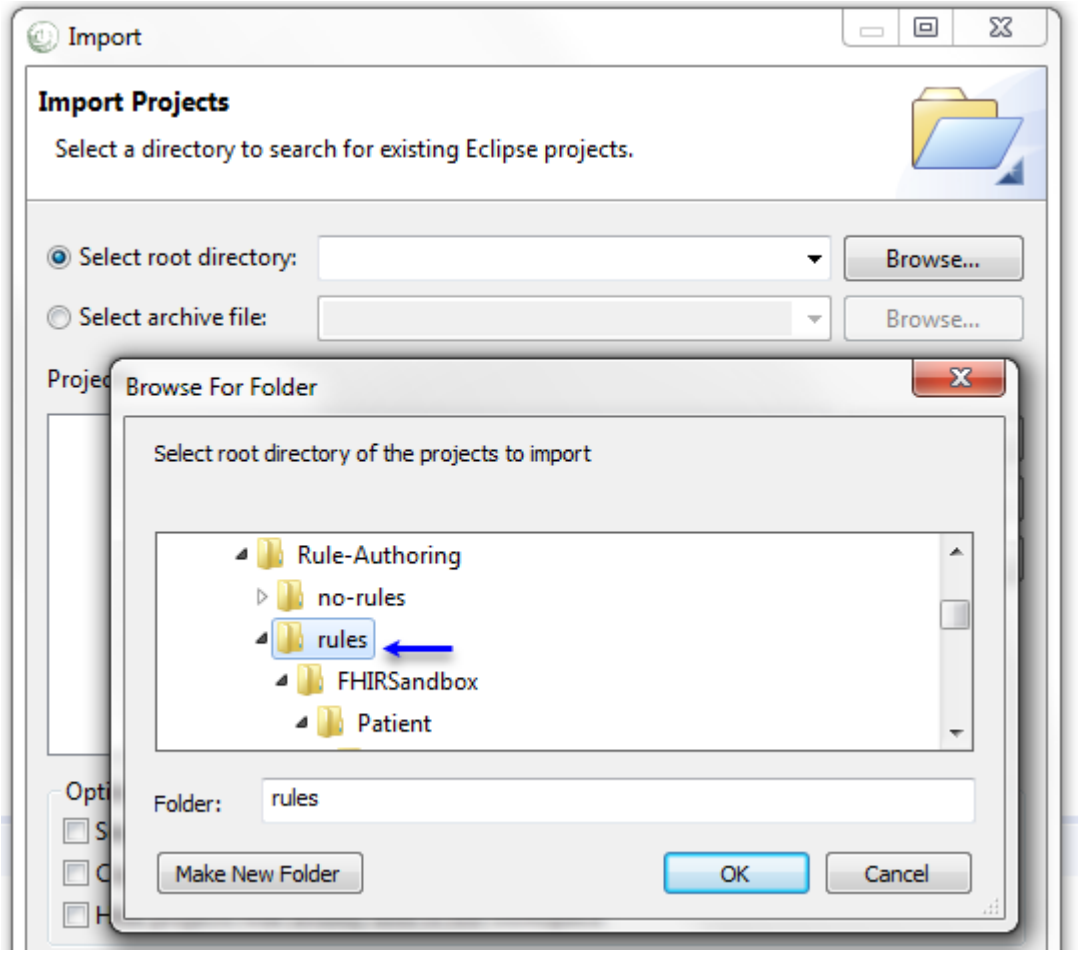
Sometimes, you may want to apply a bunch of rules as a group to a request or response message.

For example, the `ETag` and `Last-Modified` rules developed in the previous section would make sense to be applied as a group to a response message after it has been determined that the response status was "201 (Created)". Rather than putting all the logic in one rule, you can split each rule into a separate rule template and then group the rule templates in a RuleSet. The advantage of doing so is that you get separate assertions in the TestScript Execution results. That's easier to understand for user analyzing test results and is more maintainable for the rule author.

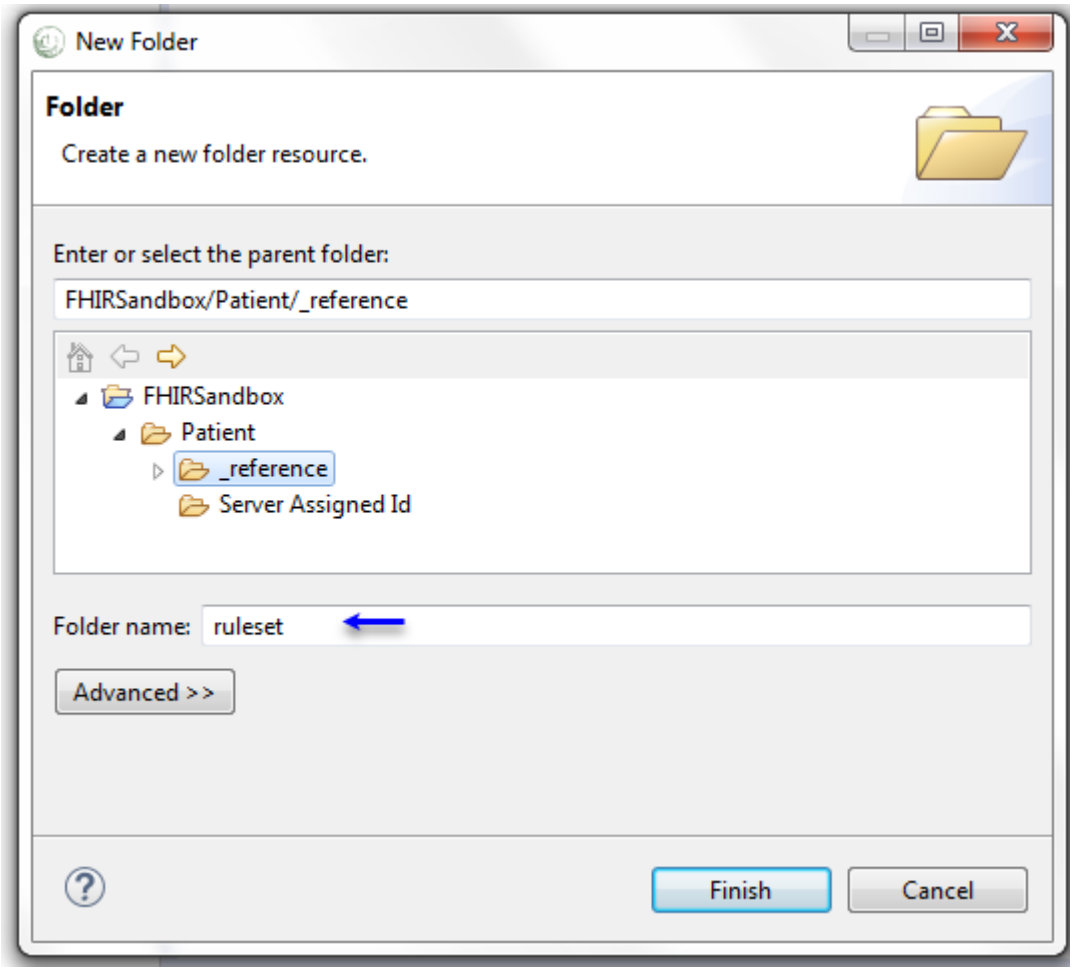
9.8.3.1 Definition

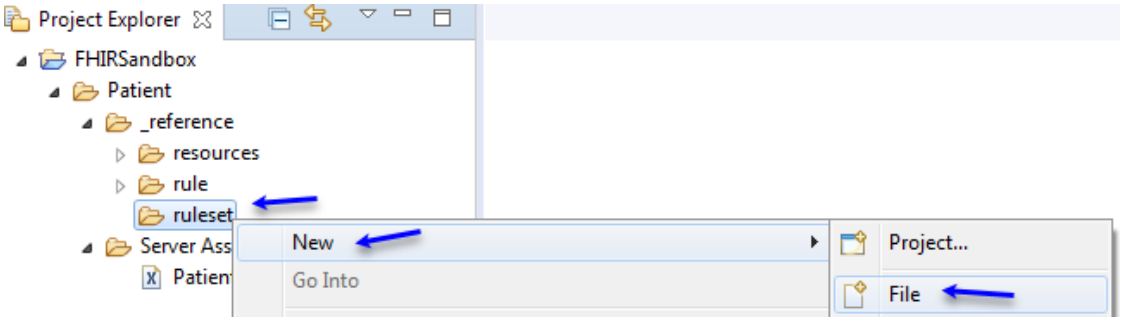
You can download all of the examples used in this section from here.

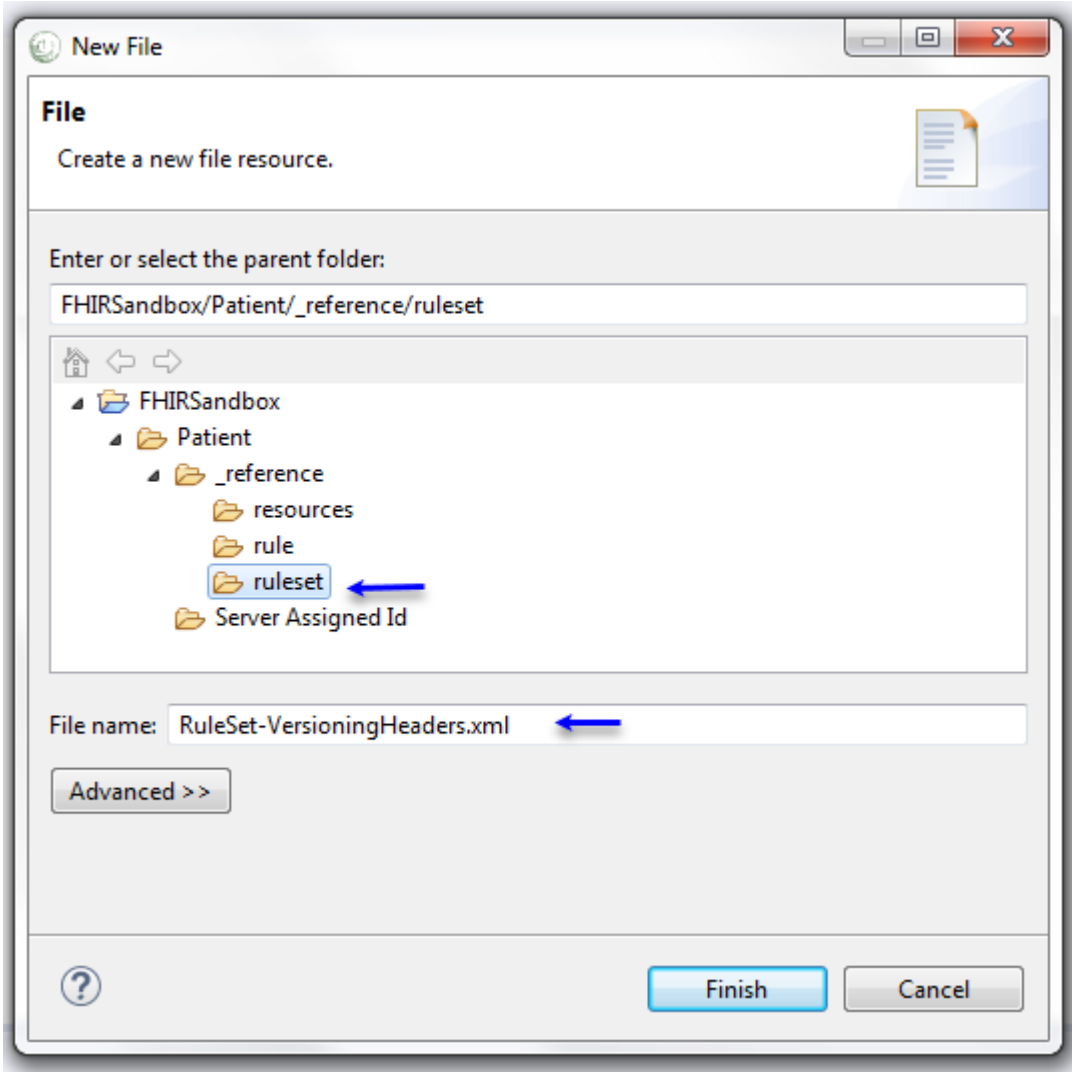
Open an instance of TestScript Editor and import the project in the example "**rules**" directory within the zip file:



Create a new directory called **ruleset** and a new file called "RuleSet-Versioning.xml" in it:


New Folder
Folder
Create a new folder resource.
Enter or select the parent folder:
FHIRSandbox/Patient/_reference
FHIRSandbox
Patient
_reference
Server Assigned Id
Folder name:
ruleset
Advanced >>
Finish
Cancel


Project Explorer
FHIRSandbox
Patient
_reference
resources
rule
ruleset
Server Ass
Patien
New
Go Into
Project...
File



Copy the following contents into the newly created **RuleSet-VersioningHeaders.xml**:

```
<RuleSet>
   <description value="Contains common rules for validating versioning-related headers."␣
↪/>
   <rule id="assertResponseCode">
      <required  value="false" />
      <reference value="../rule/AssertResponseCode.groovy"/>
   </rule>
   <rule id="assertETag">
      <required  value="true" />
      <reference value="../rule/AssertHeader.groovy"/>
   </rule>
   <rule id="assertLastModified">
      <required  value="false" />
      <reference value="../rule/AssertHeader.groovy"/>
   </rule>
</RuleSet>
```

The RuleSet contains three different rules. Two of them are marked required and one is marked optional. The optional

rules in the RuleSet can be opted out in test scripts. If the TestScript.ruleset and the assert.ruleset elements in the test script do not specify the optional rule, then the rule is not evaluated against the message. On the other hand, if neither the TestScript.ruleset nor the TestScript.test.action.assert.ruleset specifies a required rule (by the id used in the RuleSet definition), then the system will raise an error and refuse to process the rule evaluation.

The rules within a RuleSet definition can be referenced using relative paths (as is the case above) or via absolute paths. Using relative paths makes it easier for you to move test groups (and their contained resources) around and as such is the recommended practice if the rules and ruleset are applicable to the containing test group only. If they can be used across test groups, then it's recommended to use absolute paths.

Each rule within a RuleSet will be evaluated as a separate TestScript assertion during test execution as required by the FHIR spec.

The description element in a RuleSet is mandatory.

9.8.3.2 Declaration

To use a RuleSet in a TestScript assert, it must first be declared as a TestScript.ruleset.

At the top of the `Patient-server-id-json.xml` test script, let's replace the previous rule declarations with a ruleset declaration to use **RuleSet-VersioningHeaders.xml** that was defined earlier:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/
↪testscript-ruleset">
    <extension url="rulesetId">
        <valueId value="ruleset-versioning-headers"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/ruleset/RuleSet-VersioningHeaders.xml"/>
    </extension>
    <extension url="rule">
        <extension url="ruleId">
            <valueId value="assertResponseCode"/>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="responseCode"/>
            </extension>
            <extension url="value">
                <valueString value="200,201"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="responseCodeOperator"/>
            </extension>
            <extension url="value">
                <valueString value="in"/>
            </extension>
        </extension>
    </extension>
    <extension url="rule">
        <extension url="ruleId">
            <valueId value="assertETag"/>
        </extension>
```

(continues on next page)

(continued from previous page)

```
        <extension url="param">
            <extension url="name">
                <valueString value="header"/>
            </extension>
            <extension url="value">
                <valueString value="ETag"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="headerOperator"/>
            </extension>
            <extension url="value">
                <valueString value="notEmpty"/>
            </extension>
        </extension>
    </extension>
    <extension url="rule">
        <extension url="ruleId">
            <valueId value="assertLastModified"/>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="header"/>
            </extension>
            <extension url="value">
                <valueString value="Last-Modified"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="headerOperator"/>
            </extension>
            <extension url="value">
                <valueString value="notEmpty"/>
            </extension>
        </extension>
    </extension>
</extension>
```

Notice that rule declarations have been replaced with the ruleset declaration in the test script:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-ruleset">
    <extension url="rulesetId">
        <valueId value="ruleset-versioning-headers"/>
    </extension>
    <extension url="path">
        <valueString value="../_reference/ruleset/RuleSet-VersioningHeaders.xml"/>
    </extension>
    <extension url="rule">
        <extension url="ruleId">
            <valueId value="assertResponseCode"/>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="responseCode"/>
            </extension>
            <extension url="value">
                <valueString value="200,201"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="responseCodeOperator"/>
            </extension>
            <extension url="value">
                <valueString value="in"/>
            </extension>
        </extension>
    </extension>
    <extension url="rule">
        <extension url="ruleId">
            <valueId value="assertETag"/>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="header"/>
            </extension>
            <extension url="value">
                <valueString value="ETag"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="headerOperator"/>
            </extension>
            <extension url="value">
                <valueString value="notEmpty"/>
            </extension>
        </extension>
    </extension>
    <extension url="rule">
        <extension url="ruleId">
            <valueId value="assertLastModified"/>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="header"/>
            </extension>
            <extension url="value">
                <valueString value="Last-Modified"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="headerOperator"/>
            </extension>
            <extension url="value">
                <valueString value="notEmpty"/>
            </extension>
        </extension>
    </extension>
</extension>
```

9.8.3.3 Assertion

Now that we have declared the ruleset, we can use it in assertions. Replace the rule-assertions with the following ruleset-assertion in `Patient-server-id-json.xml` test script:

```
<assert>
  <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-assert-ruleset">
    <extension url="rulesetId">
      <valueId value="ruleset-versioning-headers"/>
    </extension>
  </extension>
  <description value="Complex ruleset assertion to validate expected versioning HTTP␣
↪Headers."/>
  <direction value="response"/>
  <warningOnly value="false"/>
</assert>
```

Re-upload and re-execute. You should see the following results:

Assert	Response status code is one of 200,201.	✓	0.337s	**Description:** Complex ruleset assertion to validate expected versioning HTTP Headers. **Rule:** Confirm that response status code is one of 200,201. **Definition:** ...
Assert	Response 'ETag' header is present.	✓	0.322s	**Description:** Complex ruleset assertion to validate expected versioning HTTP Headers. **Rule:** Confirm that 'ETag' header is present. **Definition:** ...
Assert	Response 'Last-Modified' header is present.	✓	0.328s	**Description:** Complex ruleset assertion to validate expected versioning HTTP Headers. **Rule:** Confirm that 'Last-Modified' header is present. **Definition:** ...

The assertResponseCode, assertETag, and assertLastModified rules were marked as required rules by the RuleSet-VersioningHeaders.xml definition (as covered in a previous section earlier). If these rules were not specified at the TestScript.ruleset, then they would have been expected to be specified in the assert.ruleset. Otherwise, the system would raise an error.

We have supplied parameters for all the rules within the RuleSet declaration at the top of the test script. These can be overwritten in individual TestScript assertions.

9.8.3.4 Overriding rules

Replace the ruleset-assertion with the following in `Patient-server-id-json.xml` test script:

```
<assert>
  <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-assert-ruleset">
    <extension url="rulesetId">
      <valueId value="ruleset-versioning-headers"/>
    </extension>
    <extension url="rule">
      <extension url="ruleId">
          <valueId value="assertResponseCode"/>
      </extension>
      <extension url="param">
          <extension url="name">
              <valueString value="responseCode"/>
```

(continues on next page)

(continued from previous page)

```
            </extension>
            <extension url="value">
                <valueString value="201"/>
            </extension>
        </extension>
        <extension url="param">
            <extension url="name">
                <valueString value="responseCodeOperator"/>
            </extension>
            <extension url="value">
                <valueString value="equals"/>
            </extension>
        </extension>
      </extension>
    </extension>
    <description value="Complex ruleset assertion to validate expected versioning HTTP␣
↪Headers."/>
    <direction value="response"/>
    <warningOnly value="false"/>
  </assert>
```

Notice that we're overriding the `responseCode` and `responseCodeOperator` parameters in the ruleset-declaration at the top of the test script with different values in this ruleset-assertion.

Re-upload and re-execute. Notice the change in the expectations and the summary/description:

9.8.4 Rules API

This section documents the APIs available to rule authors that allow them to both make assertions and also to extract data from the headers and payload of request and response messages. Many of the assertions are also availabe in FHIR TestScript. The low-level APIs though can be combined to form more complicated rule logic than can be accompished using TestScript assertions without rules.

9.8.4.1 Body

The following assertions can be performed on both `request` and `response` variables offered by the Touchstone Rules-Engine.

They would validate the presence or absence of payload in the request or response.

- **assertBodyNotEmpty()**
  - Asserts that the request or response contains a payload
  - Examples:

```
request.assertBodyNotEmpty()
```

```
response.assertBodyNotEmpty()
```

- Equivalent to these:

```
assertBodyNotEmpty(request)
```

```
assertBodyNotEmpty(response)
```

```
// operator can be passed as parameters from test script.
assertBody("notEmpty", request)
```

```
// operator can be passed as parameters from test script.
assertBody("notEmpty", response)
```

```
assert request.body != null: "Expected message body but did not find it␣
↪in request"
```

```
assert response.body != null: "Expected message body but did not find it␣
↪in response"
```

```
assert request.body: "Expected message body but did not find it in␣
↪request"
```

```
assert response.body: "Expected message body but did not find it in␣
↪response"
```

- **assertBodyEmpty()**
  - Asserts that the request or response does not contain a payload
  - Examples:

```
request.assertBodyEmpty()
```

```
response.assertBodyEmpty()
```

  - Equivalent to these:

```
assertBodyEmpty(request)
```

```
assertBodyEmpty(response)
```

```
// operator can be passed as parameters from test script.
assertBody("empty", request)
```

```
// operator can be passed as parameters from test script.
assertBody("empty", response)
```

```
assert request.body==null: "Found message body when it was not expected␣
↪in request"
```

```
assert response.body==null: "Found message body when it was not expected␣
↪in response"
```

```
assert !request.body: "Found message body when it was not expected in␣
↪request"
```

```
assert !response.body: "Found message body when it was not expected in␣
↪response"
```

9.8.4.2 Capability / Support

The assertions below can be performed on both `clientCapStmt` and `serverCapStmt` variables offered by the Touchstone Rules-Engine. These assertions confirm the presence or absence of a capability in the test system. They rely on evaluation of FHIRPath, XPath, or JSONPath expressions against the contents of the capability statement. They can be useful in ensuring that the test system supports a certain capability before other assertions are made on test execution results.

The `clientCapStmt` variable represents the capability statement of the client/origin test system and will only be set by the Rules Engine in Peer-to-Peer interactions and if the statement has been downloaded or uploaded into Touchstone. In other interactions, Touchstone serves as the client.

The `serverCapStmt` variable represents the capability statement of the server/destination test system and will be set by the Rules Engine if the statement has been downloaded or uploaded into Touchstone.

See Capability Statement and Test System List for details on how to download or upload a Capability Statement into Touchstone.

The advantage of assertSupportVia**FhirPath** assertions over assertSupportVia**NonFhirPath**, assertSupportVia**XPath**, and assertSupportVia**JsonPath** assertions is that only one FHIRPath expression is needed and will work regardless of the content type (XML or JSON) of the capability statement that was downloaded or uploaded into Touchstone. The disadvantage is that it will run significantly slower than XPath and JSONPath evaluation.

> **Warning:** Avoid using assertSupportVia**XPath**, and assertSupportVia**JsonPath** assertions as they make assumptions about the content type of the capability statement downloaded or uploaded into Touchstone. Use either assertSupportVia**FhirPath** or assertSupportVia**NonFhirPath** assertions instead as they'll work with either content type.

- **assertSupportViaFhirPath**(*fhirpath*, *expectedConfPathValue*, *operator*, *pathLabel*)
  - Asserts that the provided *fhirpath* evaluates to the provided *expectedConfPathValue* using the provided *operator* against the capability statement represented by the provided *capStmt*. The *pathLabel* is a short label/description that will be used for informational messages in place of the long *fhirpath* to describe the capability.
  - Example:

```
serverCapStmt.assertSupportViaFhirPath(
   "rest.where(mode.value='server').resource.where(type.value='Patient').
↪versioning",
    "versioned,versioned-update", "in", "Patient versioning")
```

- Equivalent to these:

```
assertSupportViaFhirPath(
   "rest.where(mode.value='server').resource.where(type.value='Patient').
↪versioning",
    "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)
```

```
boolean supports = serverCapStmt.supportsViaFhirPath(
    "rest.where(mode.value='server').resource.where(type.value='Patient
↪').versioning",
    "versioned,versioned-update", "in", "Patient versioning")

assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
boolean supports = supportsViaFhirPath(
    "rest.where(mode.value='server').resource.where(type.value='Patient
↪').versioning",
    "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)

assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
def versioning = serverCapStmt.fhirPathValue(
      "rest.where(mode.value='server').resource.where(type.value='Patient
↪').versioning")

assert versioning in ["versioned","versioned-update"]: "Expected Patient␣
↪versioning to be
      supported by server capability statement"
```

- **assertSupportViaNonFhirPath**(*xPath*, *jsonPath*, *expectedConfPathValue*, *operator*, *pathLabel*)
  - Asserts that the provided *xPath* or *jsonPath* evaluates to the provided *expectedConfPathValue* using the provided *operator* against the capability statement represented by the provided *capStmt*. The *pathLabel* is a short label/description that will be used for informational messages in place of the long *xPath* or *jsonPath* to describe the capability.

    If the capability statement downloaded or uploaded into Touchstone is in XML, then the provided *xPath* expression is used. And if the statement is in JSON, then the provided *jsonPath* expression is used.
  - Example:

```
serverCapStmt.assertSupportViaNonFhirPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning")
```

  - Equivalent to these:

```
assertSupportViaNonFhirPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
```

(continues on next page)

(continued from previous page)

```
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning"
   "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)
```

```
boolean supports = serverCapStmt.supportsViaNonFhirPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning"
   "versioned,versioned-update", "in", "Patient versioning")

assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
boolean supports = supportsViaNonFhirPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning"
   "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)

assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
def versioning = serverCapStmt.nonFhirPathValue(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning
↪")

assert versioning in ["versioned", "versioned-update"]: "Expected␣
↪Patient versioning to be
      supported by server capability statement"
```

- **assertSupportViaXPath**(*xpath*, *expectedConfPathValue*, *operator*, *pathLabel*)
  - Asserts that the provided *xpath* evaluates to the provided *expectedConfPathValue* using the provided *operator* against the capability statement represented by the provided *capStmt*. The *pathLabel* is a short label/description that will be used for informational messages in place of the long *xpath* to describe the capability.
  - Example:

```
serverCapStmt.assertSupportViaXPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning")
```

  - Equivalent to these:

```
assertSupportViaXPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)
```

(continues on next page)

(continued from previous page)

```
boolean supports = serverCapStmt.supportsViaXPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning")

assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
boolean supports = supportsViaXPath(
   "rest[mode/@value='server']/resource[type/@value='Patient']/versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)

assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
def versioning = serverCapStmt.xPathValue(
      "rest[mode/@value='server']/resource[type/@value='Patient']/
↪versioning")

assert isIn("versioned, versioned-update", versioning): "Expected␣
↪Patient versioning to be
      supported by server capability statement"
```

- **assertSupportViaJsonPath**(*jsonPath*, *expectedConfPathValue*, *operator*, *pathLabel*)
  - Asserts that the provided *jsonPath* evaluates to the provided *expectedConfPathValue* using the provided *operator* against the capability statement represented by the provided *capStmt*. The *pathLabel* is a short label/description that will be used for informational messages in place of the long *jsonPath* to describe the capability.
  - Example:

```
serverCapStmt.assertSupportViaJsonPath(
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning")
```

  - Equivalent to these:

```
assertSupportViaJsonPath(
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)
```

```
boolean supports = serverCapStmt.supportsViaJsonPath(
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning")
```

(continues on next page)

(continued from previous page)

```
assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
boolean supports = supportsViaJsonPath(
   ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].versioning
↪",
   "versioned,versioned-update", "in", "Patient versioning",␣
↪serverCapStmt)

assert supports: "Expected Patient versioning to be supported by server␣
↪capability statement"
```

```
def versioning = serverCapStmt.jsonPathValue(
      ".rest[?(@.mode=='server')].resource[?(@.type=='Patient')].
↪versioning")

assert isIn("versioned, versioned-update", versioning): "Expected␣
↪Patient versioning to be
      supported by server capability statement"
```

9.8.4.3 Content-Type

The following assertions can be performed on both `request` and `response` variables offered by the Touchstone Rules-Engine.

They would validate the `Content-Type` header in the request or response.

- **assertContentTypeContains**(*expectedValue*)
  - Asserts that the header 'Content-Type' contains the provided value
  - Example:

    ```
    response.assertContentTypeContains("xml")
    ```

  - Equivalent to these:

    ```
    assertContentTypeContains("xml", response)
    ```

    ```
    // expectedValue and operator can be passed as parameters from test␣
    ↪script.
    assertContentType("xml", "contains", request)
    ```

    ```
    assert response.header("Content-Type").contains("json"): "The actual
       value \""+response.header('Content-Type')?.value +"\" did not contain
          the expected value \"json\" for 'Content-Type' header in response."
    ```

    Notice that the raw value of the 'Content-Type' header can be grabbed using:

    ```
    response.header('Content-Type')?.value
    ```

- **assertContentTypeNotContains**(*expectedValue*)

- Asserts that the header 'Content-Type' does not contain the provided value
- Example:

```
response.assertContentTypeNotContains("xml")
```

- Equivalent to these:

```
assertContentTypeNotContains("xml", response)
```

```
// expectedValue and operator can be passed as parameters from test␣
↪script.
assertContentType("xml", "notContains", request)
```

```
assert response.header("Content-Type").notContains("xml"): "The actual
   value \""+response.header('Content-Type')?.value +"\" contained the
      expected value \"xml\" for 'Content-Type' header with operator
         'notContains' in response."
```

- **assertContentTypeEquals**(*expectedValue*)
  - Asserts that the value of header 'Content-Type' matches the provided value
  - Example:

```
response.assertContentTypeEquals("application/fhir+json;charset=UTF-8")
```

- Equivalent to these:

```
assertContentTypeEquals("application/fhir+json;charset=UTF-8", response)
```

```
// expectedValue and operator can be passed as parameters from test␣
↪script.
assertContentType("application/fhir+json;charset=UTF-8", "equals",␣
↪request)
```

```
assert response.header("Content-Type").equals("application/fhir+json;
↪charset=UTF-8"): "The
   actual value \""+response.header('Content-Type')?.value +"\" did not␣
↪match the
   expected value \"application/fhir+json;charset=UTF-8\" for 'Content-
↪Type' header in response"
```

- **assertContentTypeNotEquals**(*expectedValue*)
  - Asserts that the value of header 'Content-Type' does not match the provided value
  - Example:

```
response.assertContentTypeNotEquals("application/fhir+xml")
```

- Equivalent to these:

```
assertContentTypeNotEquals("application/fhir+xml", response)
```

```
// expectedValue and operator can be passed as parameters from test␣
↪script.
assertContentType("application/fhir+xml", "notEquals", request)
```

```
assert response.header("Content-Type").notEquals("application/fhir+xml
↪"): "The actual
   value \""+response.header('Content-Type')?.value +"\" matched the␣
↪expected value
    \"application/fhir+xml\" for 'Content-Type' header with operator
↪'notEquals'
      in response."
```

9.8.4.4 Header

The following assertions can be performed on both `request` and `response` variables offered by the Touchstone Rules-Engine.

They would validate the specified header in the request or response.

- **assertHeaderContains**(*headerName*, *expectedValue*)
  - Asserts that the provided header *headerName* contains the provided *expectedValue*.
  - Example:

```
request.assertHeaderContains("Accept","xml")
```

```
response.assertHeaderContains("Content-Type","xml")
```

  - Equivalent to these:

```
assertHeaderContains("Accept", "xml", request)
```

```
assertHeaderContains("Content-Type", "xml", response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Accept", "xml", "contains", request)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Type", "xml", "contains", response)
```

```
assert request.header('Accept').contains('xml'): "The actual value '"+
   request.header('Accept')?.value+"' did not contain the expected
      value 'xml' for 'Accept' in request"
```

```
assert response.header('Content-Type').contains('xml'): "The actual␣
↪value '"+
   request.header('Content-Type')?.value+"' did not contain the expected␣
↪value
      'xml' for 'Content-Type' in response"
```

```
assert contains('xml', request.header('Accept').value): "The actual␣
↪value '"+
   request.header('Accept')?.value+"' did not contain the expected
      value 'xml' for 'Accept' in request"
```

```
assert contains('xml', response.header('Content-Type').value): "The␣
↪actual value '"+
   request.header('Content-Type')?.value+"' did not contain the expected␣
↪value
      'xml' for 'Content-Type' in response"
```

- **assertHeaderNotContains**(*headerName*, *expectedValue*)
  - Asserts that the provided header *headerName* does not contain the provided *expectedValue*.
  - Example:

```
response.assertHeaderNotContains("Content-Type","xml")
```

  - Equivalent to these:

```
assertHeaderNotContains("Content-Type", "xml", response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Type", "xml", "notContains", response)
```

```
assert response.header('Content-Type').notContains('xml'): "The actual␣
↪value '"+
 request.header('Content-Type')?.value+"' contained the expected value
↪'xml'
   for 'Content-Type' in response"
```

```
assert notContains('xml', response.header('Content-Type').value): "The␣
↪actual value '"+
 request.header('Content-Type')?.value+"' contained the expected value
↪'xml'
   for 'Content-Type' in response"
```

- **assertHeaderEmpty**(*headerName*)
  - Asserts that the provided header *headerName* is absent or empty.
  - Example:

```
response.assertHeaderEmpty("Content-Type")
```

  - Equivalent to these:

```
assertHeaderEmpty("Content-Type", response)
```

```
// header and operator can be passed as parameters from test script.
 assertHeader("Content-Type", "empty", response)
```

```
assert response.header('Content-Type').empty(): "Found 'Content-Type'
   header when it was not expected in response"
```

```
assert empty(response.header('Content-Type').value): "Found 'Content-Type
↪'
   header when it was not expected in response"
```

- **assertHeaderNotEmpty**(*headerName*)
  - Asserts that the provided header *headerName* is present and not empty.
  - Example:

```
response.assertHeaderNotEmpty("Content-Type")
```

  - Equivalent to these:

```
assertHeaderNotEmpty("Content-Type", response)
```

```
// header and operator can be passed as parameters from test script.
assertHeader("Content-Type", "notEmpty", response)
```

```
assert response.header('Content-Type').notEmpty(): "Expected
   'Content-Type' header but did not find it in response"
```

```
assert notEmpty(response.header('Content-Type').value): "Expected
   'Content-Type' header but did not find it in response"
```

- **assertHeaderEquals**(*headerName*, *expectedValue*)
  - Asserts that the provided header *headerName* matches the provided *expectedValue*
  - Example:

```
response.assertHeaderEquals("Content-Type","text/html")
```

  - Equivalent to these:

```
assertHeaderEquals("Content-Type","text/html", response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Type", "text/html", "equals", response)
```

```
assert response.header('Content-Type').equals("text/html"): "The actual␣
↪value '"+
   request.header('Content-Type')?.value+"' did not match the expected␣
↪value
      'text/html' for 'Content-Type' in response"
```

```
assert equals("text/html", response.header('Content-Type').value): "The␣
↪actual value '"+
   request.header('Content-Type')?.value+"' did not match the expected␣
↪value
```

(continues on next page)

(continued from previous page)

```
      'text/html' for 'Content-Type' in response"
```

- **assertHeaderNotEquals**(*headerName*, *expectedValue*)
  - Asserts that the provided header *headerName* does not match the provided *expectedValue*
  - Example:

```
response.assertHeaderNotEquals("Content-Type","text/html")
```

  - Equivalent to these:

```
assertHeaderNotEquals("Content-Type","text/html", response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Type", "text/html", "notEquals", response)
```

```
assert response.header('Content-Type').notEquals("text/html"): "The␣
↪actual value '"+
   request.header('Content-Type')?.value+"' matched the expected value
↪'text/html'
   for 'Content-Type' in response"
```

```
assert notEquals("text/html", response.header('Content-Type').value):
↪"The actual value '"+
   request.header('Content-Type')?.value+"' matched the expected value
↪'text/html'
   for 'Content-Type' in response"
```

- **assertHeaderGreaterThan**(*headerName*, *expectedValue*)
  - Asserts that the provided header *headerName* is greater than the provided *expectedValue*
  - Example:

```
response.assertHeaderGreaterThan("Content-Length", 0)
```

  - Equivalent to each of these:
    *
```
assertHeaderGreaterThan("Content-Length", 0, response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Length", 0, "greaterThan", response)
```

    *
```
assert response.header('Content-Length').greaterThan(0): "Expected 'Content-
↪Length' header
   to be greater than 0 but was "+ response.header('Content-Length')?.value␣
↪+" in response"
```

```
assert greaterThan(0, response.header('Content-Length').value): "Expected
↪'Content-Length' header
   to be greater than 0 but was "+ response.header('Content-Length')?.value␣
↪+" in response"
```

- **assertHeaderLessThan**(*headerName*, *expectedValue*)
  - Asserts that the provided header *headerName* is less than the provided *expectedValue*
  - Example:

```
response.assertHeaderLessThan("Content-Length", 100)
```

  - Equivalent to each of these:

```
assertHeaderLessThan("Content-Length", 100, response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Length", 100, "lessThan", response)
```

```
assert response.header('Content-Length').lessThan(100): "Expected 'Content-
↪Length' header to be
   less than 100 but was "+ response.header('Content-Length')?.value +" in␣
↪response"
```

```
assert lessThan(100, response.header('Content-Length').value): "Expected
↪'Content-Length' header to be
   less than 100 but was "+ response.header('Content-Length')?.value +" in␣
↪response"
```

- **assertHeaderIn**(*headerName*, *expectedValues*)
  - Asserts that the provided header *headerName* is one of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that 'Content-Type' header is 'application/fhir+json' or
↪'application/fhir+json;charset=UTF-8'
response.assertHeaderIn("Content-Type", "application/fhir+json,
↪application/fhir+json;charset=UTF-8")
```

  - Equivalent to each of these:

```
assertHeaderIn("Content-Type", "application/fhir+json,application/fhir+json;
↪charset=UTF-8", response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Type", "application/fhir+json,application/fhir+json;
↪charset=UTF-8", "in", response)
```

```
* assert response.header("Content-Type").in("application/fhir+json,
↪application/fhir+json;charset=UTF-8"):
   "Expected one of the values in [application/fhir+json, application/
↪fhir+json;charset=UTF-8] for
   'Content-Type' header but encountered \""+ response.header('Content-Type
↪')?.value +"\" in response"
```

```
* assert isIn("application/fhir+json,application/fhir+json;charset=UTF-8",␣
↪response.header("Content-Type").value):
   "Expected one of the values in [application/fhir+json, application/
↪fhir+json;charset=UTF-8] for 'Content-Type'
      header but encountered \""+ response.header('Content-Type')?.value +"\
↪" in response"
```

- **assertHeaderNotIn**(*headerName*, *expectedValues*)
  - Asserts that the provided header *headerName* is none of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that 'Content-Type' header is neither 'text/plain' nor 'text/html'
response.assertHeaderNotIn("Content-Type", "text/plain,text/html")
```

  - Equivalent to each of these:

```
* assertHeaderNotIn("Content-Type", "text/plain,text/html", response)
```

```
// header, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertHeader("Content-Type", "text/plain,text/html", "notIn", response)
```

```
* assert response.header("Content-Type").notIn("text/plain,text/html") :
↪"Expected
   none of the values in [text/plain, text/html] for 'Content-Type' header␣
↪but
   encountered \""+ response.header('Content-Type')?.value +"\" with␣
↪operator 'notIn'
   in response"
```

```
* assert isNotIn("text/plain,text/html", response.header("Content-Type").
↪value) : "Expected
   none of the values in [text/plain, text/html] for 'Content-Type' header␣
↪but
   encountered \""+ response.header('Content-Type')?.value +"\" with␣
↪operator 'notIn'
   in response"
```

9.8.4.5 Minimum

Minimum assertions are efficient in that it relieves the author from performing many individual path evaluations against the payload and comparing those results to a set of expected values. The author would define one fixture that contains all of the expected elements in the request or response payload and would then reference this fixture as minimumId.

The assertion can be performed on `request` and `response` variables offered by the Touchstone Rules-Engine.

- **assertMinimum**(*mimimumId*)
  - Asserts that the payload contains all the element/content in another fixture pointed to by the provided minimumId. This can be a statically defined fixture or one that is dynamically set via `responseId` during TestScript execution.
  - Example:

```
response.assertMinimum("patient-create-PeterChalmers-min")
```

  - Equivalent to these:

```
assertMinimum("patient-create-PeterChalmers-min", response)
```

Note that in both of the examples above, the fixture "patient-create-PeterChalmers-min" is expected to have been defined and uploaded to Touchstone. See the test script connectathon-18-patient-base-client-id-json for an example on how minimumId assertion is performed in TestScript and how the minimumId fixture is defined:

- Minimum assertion definition in TestScript:

```
<action>
    <assert>
        <description value="Confirm that the returned resource contains the expected retained elements and values.
        <direction value="response"/>
        <minimumId value="patient-create-PeterChalmers-min"/>
        <warningOnly value="true"/>
    </assert>
</action>
```

- Minimum fixture definition in TestScript:

```
<fixture id="patient-create-PeterChalmers-min">
    <autocreate value="false"/>
    <autodelete value="false"/>
    <resource>
        <reference value="../_reference/resources/patient-create-PeterChalmers-min.json"/>
    </resource>
</fixture>
```

- Fixture contents for `patient-create-PeterChalmers-min.json` can be found here

9.8.4.6 Path

Path assertions involve extracting element (or attribute) values and comparing them to an expected value.

9.8.4.6.1 FHIRPath

FHIRPath assertions can be evaluated against payloads of both `request` and `response` variables offered by the Touchstone Rules-Engine. The same FHIRPath expression will work against both XML and JSON content. However, FHIRPath assertions run significantly slower than XPath assertions and JsonPath assertions.

- **assertFhirPathBoolean**(*fhirpath*)
  - Asserts that the evaluated value of the provided *fhirpath* expression is true
  - Example:

```
// Asserts that the number of resource id elements in the response is 10
response.assertFhirPathBoolean("entry.resource.id.count() = 10")
```

  - Equivalent to each of these:

```
assertFhirPathBoolean("entry.resource.id.count() = 10", response)
```

```
assert response.getFhirPathBoolean("entry.resource.id.count() = 10") : "The
   expression \"entry.resource.id.count() != 10\" evaluated to false"
```

```
assert response.fhirPathBoolean("entry.resource.id.count() = 10") : "The
   expression \"entry.resource.id.count() != 10\" evaluated to false"
```

- **assertFhirPathContains**(*fhirpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *fhirpath* expression contains the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertFhirPathContains("entry[1].resource.name.family",
↪"Chalmers")
```

  - Equivalent to each of these:

```
assertFhirPathContains("entry[1].resource.name.family", "Chalmers",␣
↪response)
```

```
// fhirPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertFhirPath("entry[1].resource.name.family", "Chalmers", "contains",␣
↪response)
```

```
def family = response.getFhirPathValue("entry[1].resource.name.family")

assert family.contains("Chalmers"): "The actual value \""+family+"\" did␣
↪not contain the expected value
   \"Chalmers\" for \"entry[1].resource.name.family\" in response."
```

```
def family = response.fhirPathValue("entry[1].resource.name.family")
```

(continues on next page)

(continued from previous page)

```
assert containsIgnoreCase("chalmers", family): "The actual value \""+family+
↪"\" did not contain the expected value
  \"Chalmers\"  for \"entry[1].resource.name.family\" in response."
```

```
* def family = response.fhirPath("entry[1].resource.name.family").value

  assert contains("Chalmers", family): "The actual value \""+family+"\" did␣
  ↪not contain the expected value
    \"Chalmers\"  for \"entry[1].resource.name.family\" in response."
```

Notice how the value of a FHIRPath evaluation can be stored in a variable. This is useful when you want to develop more complicated rule scripts where the assertions involve multiple comparisons.

- **assertFhirPathNotContains**(*fhirpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *fhirpath* expression does not contain the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertFhirPathNotContains("entry[1].resource.name.family",
↪"Chalmers")
```

  - Equivalent to each of these:

```
* assertFhirPathNotContains("entry[1].resource.name.family", "Chalmers",␣
  ↪response)
```

```
* // fhirPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertFhirPath("entry[1].resource.name.family", "Chalmers", "notContains",␣
  ↪response)
```

```
* def family = response.getFhirPathValue("entry[1].resource.name.family")

  assert !family.contains("Chalmers"): "The actual value \""+family+"\"␣
  ↪contained the expected value
    \"Chalmers\" for \"entry[1].resource.name.family\" with operator
  ↪'notContains' in response."
```

```
* def family = response.fhirPathValue("entry[1].resource.name.family")

  assert notContainsIgnoreCase("chalmers", family): "The actual value \"
  ↪"+family+"\" contained the expected value
    \"Chalmers\"  for \"entry[1].resource.name.family\" with operator
  ↪'notContains' in response."
```

```
* def family = response.fhirPath("entry[1].resource.name.family").value
```

(continues on next page)

(continued from previous page)

```
assert notContains("Chalmers", family): "The actual value \""+family+"\"␣
↪contained the expected value
  \"Chalmers\"  for \"entry[1].resource.name.family\" with operator
↪'notContains' in response."
```

- **assertFhirPathEmpty**(*fhirpath*)
  - Asserts that the evaluated value of the provided *fhirpath* expression is absent or empty.
  - Example:

```
response.assertFhirPathEmpty("entry[0].resource.photo.title")
```

  - Equivalent to these:

```
assertFhirPathEmpty("entry[0].resource.photo.title", response)
```

```
response.fhirPath("entry[0].resource.photo.title").empty();
```

```
def title = response.getFhirPathValue("entry[0].resource.photo.title")

assert !title: "Expected title to be absent but was present in response"
```

```
def title = response.fhirPathValue("entry[0].resource.photo.title")

assert title==null: "Expected title to be absent but was present in␣
↪response"
```

```
def title = response.fhirPath("entry[0].resource.photo.title").value

assert empty(title): "Expected title to be absent but was present in␣
↪response"
```

- **assertFhirPathNotEmpty**(*fhirpath*)
  - Asserts that the evaluated value of the provided *fhirpath* expression is present and not empty.
  - Example:

```
response.assertFhirPathNotEmpty("entry[0].resource.birthDate")
```

  - Equivalent to these:

```
assertFhirPathNotEmpty("entry[0].resource.birthDate", response)
```

```
response.fhirPath("entry[0].resource.birthDate").notEmpty();
```

```
def title = response.getFhirPathValue("entry[0].resource.birthDate")

assert title: "Expected birthDate to be absent but was present in␣
↪response"
```

```
def title = response.fhirPathValue("entry[0].resource.birthDate")

assert title!=null: "Expected birthDate to be absent but was present in␣
↪response"
```

```
def title = response.fhirPath("entry[0].resource.birthDate").value

assert notEmpty(title): "Expected birthDate to be absent but was present␣
↪in response"
```

- **assertFhirPathEquals(***fhirpath*, *expectedValue***)**
  - Asserts that the evaluated value of the provided *fhirpath* expression matches the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the first entry is
↪'Chalmers'
response.assertFhirPathEquals("entry[0].resource.name.family", "Chalmers
↪")
```

  - Equivalent to each of these:
    *
```
assertFhirPathEquals("entry[0].resource.name.family", "Chalmers", response)
```

    *
```
// fhirPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertFhirPath("entry[0].resource.name.family", "Chalmers", "equals",␣
↪response)
```

    *
```
def family = response.getFhirPathValue("entry[0].resource.name.family")

assert family.equals("Chalmers"): "The actual value \""+family+"\" did not␣
↪match the expected value
\"Chalmers\" for \"entry[0].resource.name.family\" in response."
```

    *
```
def family = response.fhirPathValue("entry[0].resource.name.family")

assert equalsIgnoreCase("chalmers", family): "The actual value \""+family+"\
↪" did not match the expected value
   \"Chalmers\"  for \"entry[0].resource.name.family\" in response."
```

    *
```
def family = response.fhirPath("entry[0].resource.name.family").getValue()

assert equals("Chalmers", family): "The actual value \""+family+"\" did not␣
↪match the expected value
   \"Chalmers\"  for \"entry[0].resource.name.family\" in response."
```

    *
```
def family = response.fhirPath("entry[0].resource.name.family").value

assert family == "Chalmers": "The actual value \""+family+"\" did not match␣
↪the expected value
   \"Chalmers\" for \"entry[0].resource.name.family\" in response."
```

- **assertFhirPathNotEquals**(*fhirpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *fhirpath* expression does not match the provided *expectedValue*.
  - Example:

```
// Asserts that values of the family element of the first entry is not
↪'Chalmers'
response.assertFhirPathNotEquals("entry[0].resource.name.family",
↪"Chalmers")
```

  - Equivalent to each of these:

```
* assertFhirPathNotEquals("entry[0].resource.name.family", "Chalmers",␣
↪response)
```

```
* // fhirPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertFhirPath("entry[0].resource.name.family", "Chalmers", "notEquals",␣
↪response)
```

```
* def family = response.getFhirPathValue("entry[0].resource.name.family")

assert !family.equals("Chalmers"): "The actual value \""+family+"\" matched
   the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
↪with operator
      'notEquals' in response."
```

```
* def family = response.fhirPathValue("entry[0].resource.name.family")

assert notEqualsIgnoreCase("chalmers", family): "The actual value \"
↪"+family+"\" matched
   the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
↪with operator
      'notEquals' in response."
```

```
* def family = response.fhirPath("entry[0].resource.name.family").getValue()

assert family != "Chalmers": "The actual value \""+family+"\" matched
   the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
↪with operator
      'notEquals' in response."
```

```
* def family = response.fhirPath("entry[0].resource.name.family").value

assert notEquals("Chalmers", family): "The actual value \""+family+"\"␣
↪matched
   the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
↪with operator
      'notEquals' in response."
```

- **assertFhirPathGreaterThan**(*fhirpath*, *expectedValue*)

- Asserts that the evaluated value of the provided *fhirpath* expression is greater than the provided *expectedValue*.
- Example:

```
// Asserts that resource id of the second entry is greater than 1100 in␣
↪the response
response.assertFhirPathGreaterThan("entry[1].resource.id", 1100)
```

- Equivalent to each of these:

  * ```
    assertFhirPathGreaterThan("entry[1].resource.id", 1100, response)
    ```

  * ```
    // fhirPath, expectedValue, and operator can be passed as parameters from␣
    ↪test script.
    assertFhirPath("entry[1].resource.id", "1100", "greaterThan", response)
    ```

  * ```
    def id = response.getFhirPathValue("entry[1].resource.id")

    assert id.toInteger() > 1100: "Expected \"entry[1].resource.id\" to be␣
    ↪greater than 5000
        but was "+id+" in response"
    ```

  * ```
    def id = response.fhirPathValue("entry[1].resource.id")

    assert (id as Integer) > 1100: "Expected \"entry[1].resource.id\" to be␣
    ↪greater than 5000
       but was "+id+" in response"
    ```

  * ```
    def id = response.fhirPath("entry[1].resource.id").value

    assert greaterThan(1100, id): "Expected \"entry[1].resource.id\" to be␣
    ↪greater than 5000
       but was "+id+" in response"
    ```

- **assertFhirPathLessThan(***fhirpath*, *expectedValue***)**
  - Asserts that the evaluated value of the provided *fhirpath* expression is less than the provided *expectedValue*.
  - Example:

```
// Asserts that resource id of the second entry is less than 1100 in the␣
↪response
response.assertFhirPathLessThan("entry[1].resource.id", 1100)
```

  - Equivalent to each of these:

    * ```
      assertFhirPathLessThan("entry[1].resource.id", 1100, response)
      ```

    * ```
      // fhirPath, expectedValue, and operator can be passed as parameters from␣
      ↪test script.
      assertFhirPath("entry[1].resource.id", "1100", "lessThan", response)
      ```
      ```

```
def id = response.getFhirPathValue("entry[1].resource.id")

assert id.toInteger() < 1100: "Expected \"entry[1].resource.id\" to be less␣
↪than 5000
    but was "+id+" in response"
```

```
def id = response.fhirPath("entry[1].resource.id").getValue()

assert (id as Integer) < 1100: "Expected \"entry[1].resource.id\" to be␣
↪less than 5000
   but was "+id+" in response"
```

```
def id = response.fhirPath("entry[1].resource.id").value

assert lessThan(1100, id): "Expected \"entry[1].resource.id\" to be less␣
↪than 5000
   but was "+id+" in response"
```

- **assertFhirPathIn**(*fhirpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *fhirpath* expression is one of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertFhirPathIn("entry[1].resource.id", "1100,1101,1102")
```

  - Equivalent to each of these:

```
assertFhirPathIn("entry[1].resource.id", "1100,1101,1102", response)
```

```
// fhirPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertFhirPath("entry[1].resource.id", "1100,1101,1102", "in", response)
```

```
def id = response.getFhirPathValue("entry[1].resource.id")

assert id in ["1100", "1101", "1102"]: "Expected one of the values in [1100,
↪ 1101, 1102]
   for \"entry[1].resource.id\" but encountered \""+id+"\" in response."
```

```
def id = response.fhirPath("entry[1].resource.id").value

assert isIn("1100,1101,1102", id): "Expected one of the values in [1100,␣
↪1101, 1102] for
   \"entry[1].resource.id\" but encountered \""+id+"\" in response."
```

- **assertFhirPathNotIn**(*fhirpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *fhirpath* expression is none of the provided *expectedValues* where each value is separated by a comma.

- Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertFhirPathNotIn("entry[1].resource.id", "1100,1101,1102")
```

- Equivalent to each of these:

  *
```
assertFhirPathNotIn("entry[1].resource.id", "1100,1101,1102", response)
```

  *
```
// fhirPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertFhirPath("entry[1].resource.id", "1100,1101,1102", "notIn", response)
```

  *
```
def id = response.getFhirPathValue("entry[1].resource.id")

assert !(id in ["1100", "1101", "1102"]): "Expected none of the values in␣
↪[1100, 1101, 1102]
   for \"entry[1].resource.id\" but encountered \""+id+"\" with operator
↪'notIn' in response."
```

  *
```
def id = response.fhirPath("entry[1].resource.id").value

assert isNotIn("1100,1101,1102", id): "Expected none of the values in [1100,
↪ 1101, 1102] for
   \"entry[1].resource.id\" but encountered \""+id+"\" with operator 'notIn
↪' in response."
```

9.8.4.6.2 JSONPath

JSONPath assertions can be evaluated against payloads of both `request` and `response` variables offered by the Touchstone Rules-Engine. Both XPath assertions and JSONPath assertions run significantly faster than FHIRPath assertions. Separate XPath and JSONPath expressions are needed though for XML and JSON content while only one FHIRPath expression is needed for both XML and JSON content.

- **assertJsonPathContains**(*jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *jsonpath* expression contains the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertJsonPathContains("entry[1].resource.name.family",
↪"Chalmers")
```

  - Equivalent to each of these:

  *
```
assertJsonPathContains("entry[1].resource.name.family", "Chalmers",␣
↪response)
```

```
// jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertJsonPath("entry[1].resource.name.family", "Chalmers", "contains",␣
↪response)
```

```
def family = response.getJsonPathValue("entry[1].resource.name.family")

assert family.contains("Chalmers"): "The actual value \""+family+"\" did␣
↪not contain the expected value
   \"Chalmers\" for \"entry[1].resource.name.family\" in response."
```

```
def family = response.jsonPath("entry[1].resource.name.family").value

assert contains("Chalmers", family): "The actual value \""+family+"\" did␣
↪not contain the expected value
   \"Chalmers\"  for \"entry[1].resource.name.family\" in response."
```

Notice how the value of JsonPath evaluation can be stored in a variable. This is useful when you want to develop more complicated rule scripts where the assertions involve multiple comparisons.

- **assertJsonPathNotContains**(*jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *jsonpath* expression does not contain the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertJsonPathNotContains("entry[1].resource.name.family",
↪"Chalmers")
```

  - Equivalent to each of these:

```
assertJsonPathNotContains("entry[1].resource.name.family", "Chalmers",␣
↪response)
```

```
// jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertJsonPath("entry[1].resource.name.family", "Chalmers", "notContains",␣
↪response)
```

```
def family = response.getJsonPathValue("entry[1].resource.name.family")

assert !family.contains("Chalmers"): "The actual value \""+family+"\"␣
↪contained the expected value
   \"Chalmers\" for \"entry[1].resource.name.family\" with operator
↪'notContains' in response."
```

```
def family = response.jsonPath("entry[1].resource.name.family").value
```

(continues on next page)

(continued from previous page)

```
assert notContains("Chalmers", family): "The actual value \""+family+"\"␣
↪contained the expected value
   \"Chalmers\"  for \"entry[1].resource.name.family\" with operator
↪'notContains' in response."
```

- **assertJsonPathEmpty**(*jsonpath*)
  - Asserts that the evaluated value of the provided *jsonpath* expression is absent or empty.
  - Example:

```
response.assertJsonPathEmpty("entry[0].resource.photo.title")
```

  - Equivalent to these:

```
assertJsonPathEmpty("entry[0].resource.photo.title", response)
```

```
response.jsonPath("entry[0].resource.photo.title").empty();
```

```
def title = response.getJsonPathValue("entry[0].resource.photo.title")

assert !title: "Expected title to be absent but was present in response"
```

```
def title = response.jsonPath("entry[0].resource.photo.title").value

assert empty(title): "Expected title to be absent but was present in␣
↪response"
```

- **assertJsonPathNotEmpty**(*jsonpath*)
  - Asserts that the evaluated value of the provided *jsonpath* expression is present and not empty.
  - Example:

```
response.assertJsonPathNotEmpty("entry[0].resource.birthDate")
```

  - Equivalent to these:

```
assertJsonPathNotEmpty("entry[0].resource.birthDate", response)
```

```
response.jsonPath("entry[0].resource.birthDate").notEmpty();
```

```
def title = response.getJsonPathValue("entry[0].resource.birthDate")

assert title: "Expected birthDate to be absent but was present in␣
↪response"
```

```
def title = response.jsonPath("entry[0].resource.birthDate").value

assert notEmpty(title): "Expected birthDate to be absent but was present␣
↪in response"
```

- **assertJsonPathEquals**(*jsonpath*, *expectedValue*)

- Asserts that the evaluated value of the provided *jsonpath* expression matches the provided *expectedValue*.
- Example:

```
// Asserts that the value of the family element of the first entry is
↪'Chalmers'
response.assertJsonPathEquals("entry[1]/resource/Patient/name/family",
↪"Chalmers")
```

- Equivalent to each of these:

  *
```
assertJsonPathEquals("entry[1]/resource/Patient/name/family", "Chalmers",␣
↪response)
```

  *
```
// xPath, expectedValue, and operator can be passed as parameters from test␣
↪script.
assertJsonPath("entry[1]/resource/Patient/name/family", "Chalmers", "equals
↪", response)
```

  *
```
def family = response.getJsonPathValue("entry[1]/resource/Patient/name/
↪family")

assert family.equals("Chalmers"): "The actual value \""+family+"\" did not␣
↪match the expected value
\"Chalmers\" for \"entry[1]/resource/Patient/name/family\" in response."
```

  *
```
def family = response.jsonPath("entry[1]/resource/Patient/name/family").
↪getValue()

assert family == "Chalmers": "The actual value \""+family+"\" did not match␣
↪the expected value
   \"Chalmers\" for \"entry[1]/resource/Patient/name/family\" in response."
```

  *
```
def family = response.xPath("entry[1]/resource/Patient/name/family").value

assert equals("Chalmers", family): "The actual value \""+family+"\" did not␣
↪match the expected value
   \"Chalmers\"  for \"entry[1]/resource/Patient/name/family\" in response."
```

- **assertJsonPathEquals(***jsonpath*, *expectedValue***)**
  - Asserts that the evaluated value of the provided *jsonpath* expression matches the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the first entry is
↪'Chalmers'
response.assertJsonPathEquals("entry[0].resource.name.family", "Chalmers
↪")
```

  - Equivalent to each of these:

  *
```
assertJsonPathEquals("entry[0].resource.name.family", "Chalmers", response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertJsonPath("entry[0].resource.name.family", "Chalmers", "equals",␣
  ↪response)
```

```
* def family = response.getJsonPathValue("entry[0].resource.name.family")

  assert family.equals("Chalmers"): "The actual value \""+family+"\" did not␣
  ↪match the expected value
  \"Chalmers\" for \"entry[0].resource.name.family\" in response."
```

```
* def family = response.jsonPath("entry[0].resource.name.family").getValue()

  assert family == "Chalmers": "The actual value \""+family+"\" did not match␣
  ↪the expected value
     \"Chalmers\" for \"entry[0].resource.name.family\" in response."
```

```
* def family = response.jsonPath("entry[0].resource.name.family").value

  assert equals("Chalmers", family): "The actual value \""+family+"\" did not␣
  ↪match the expected value
     \"Chalmers\"  for \"entry[0].resource.name.family\" in response."
```

- **assertJsonPathNotEquals(***jsonpath*, *expectedValue***)**
  - Asserts that the evaluated value of the provided *jsonpath* expression does not match the provided *expectedValue*.
  - Example:

```
// Asserts that values of the family element of the first entry is not
↪'Chalmers'
response.assertJsonPathNotEquals("entry[0].resource.name.family",
↪"Chalmers")
```

  - Equivalent to each of these:

```
* assertJsonPathNotEquals("entry[0].resource.name.family", "Chalmers",␣
  ↪response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertJsonPath("entry[0].resource.name.family", "Chalmers", "notEquals",␣
  ↪response)
```

```
* def family = response.getJsonPathValue("entry[0].resource.name.family")

  assert !family.equals("Chalmers"): "The actual value \""+family+"\" matched
     the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
  ↪with operator
        'notEquals' in response."
```

```
* def family = response.jsonPath("entry[0].resource.name.family").getValue()

  assert family != "Chalmers": "The actual value \""+family+"\" matched
     the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
  ↪with operator
        'notEquals' in response."
```

```
* def family = response.jsonPath("entry[0].resource.name.family").value

  assert notEquals("Chalmers", family): "The actual value \""+family+"\"␣
  ↪matched
     the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
  ↪with operator
        'notEquals' in response."
```

- **assertJsonPathGreaterThan**(*jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *jsonpath* expression is greater than the provided *expectedValue*.
  - Example:

```
// Asserts that resource id of the second entry is greater than 1100 in␣
↪the response
response.assertJsonPathGreaterThan("entry[1].resource.id", 1100)
```

  - Equivalent to each of these:

```
* assertJsonPathGreaterThan("entry[1].resource.id", 1100, response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertJsonPath("entry[1].resource.id", "1100", "greaterThan", response)
```

```
* def id = response.getJsonPathValue("entry[1].resource.id")

  assert id.toInteger() > 1100: "Expected \"entry[1].resource.id\" to be␣
  ↪greater than 5000
      but was "+id+" in response"
```

```
* def id = response.jsonPath("entry[1].resource.id").getValue()

  assert (id as Integer) > 1100: "Expected \"entry[1].resource.id\" to be␣
  ↪greater than 5000
     but was "+id+" in response"
```

```
* def id = response.jsonPath("entry[1].resource.id").value

  assert greaterThan(1100, id): "Expected \"entry[1].resource.id\" to be␣
  ↪greater than 5000
     but was "+id+" in response"
```

- **assertJsonPathLessThan**(*jsonpath*, *expectedValue*)

- Asserts that the evaluated value of the provided *jsonpath* expression is less than the provided *expectedValue*.
- Example:

```
// Asserts that resource id of the first entry is less than 1100 in the␣
↪response
response.assertJsonPathLessThan("entry[0].resource.id", 1100)
```

- Equivalent to each of these:

  *
```
assertJsonPathLessThan("entry[0].resource.id", 1100, response)
```

  *
```
// jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertJsonPath("entry[0].resource.id", "1100", "lessThan", response)
```

  *
```
def id = response.getJsonPathValue("entry[0].resource.id")

assert id.toInteger() < 1100: "Expected \"entry[0].resource.id\" to be less␣
↪than 5000
    but was "+id+" in response"
```

  *
```
def id = response.jsonPath("entry[0].resource.id").getValue()

assert (id as Integer) < 1100: "Expected \"entry[0].resource.id\" to be␣
↪less than 5000
   but was "+id+" in response"
```

  *
```
def id = response.jsonPath("entry[0].resource.id").value

assert lessThan(1100, id): "Expected \"entry[0].resource.id\" to be less␣
↪than 5000
   but was "+id+" in response"
```

- **assertJsonPathIn**(*jsonpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *jsonpath* expression is one of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertJsonPathIn("entry[1].resource.id", "1100,1101,1102")
```

  - Equivalent to each of these:

  *
```
assertJsonPathIn("entry[1].resource.id", "1100,1101,1102", response)
```

  *
```
// jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertJsonPath("entry[1].resource.id", "1100,1101,1102", "in", response)
```

```
* def id = response.getJsonPathValue("entry[1].resource.id")

  assert id in ["1100", "1101", "1102"]: "Expected one of the values in [1100,
  ↪ 1101, 1102]
    for \"entry[1].resource.id\" but encountered \""+id+"\" in response."
```

```
* def id = response.jsonPath("entry[1].resource.id").value

  assert isIn("1100,1101,1102", id): "Expected one of the values in [1100,␣
  ↪1101, 1102] for
    \"entry[1].resource.id\" but encountered \""+id+"\" in response."
```

- **assertJsonPathNotIn**(*jsonpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *jsonpath* expression is none of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertJsonPathNotIn("entry[1].resource.id", "1100,1101,1102")
```

  - Equivalent to each of these:

```
* assertJsonPathNotIn("entry[1].resource.id", "1100,1101,1102", response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertJsonPath("entry[1].resource.id", "1100,1101,1102", "notIn", response)
```

```
* def id = response.getJsonPathValue("entry[1].resource.id")

  assert !(id in ["1100", "1101", "1102"]): "Expected none of the values in␣
  ↪[1100, 1101, 1102]
    for \"entry[1].resource.id\" but encountered \""+id+"\" with operator
  ↪'notIn' in response."
```

```
* def id = response.jsonPath("entry[1].resource.id").value

  assert isNotIn("1100,1101,1102", id): "Expected none of the values in [1100,
  ↪ 1101, 1102] for
    \"entry[1].resource.id\" but encountered \""+id+"\" with operator 'notIn
  ↪' in response."
```

9.8.4.6.3 NonFHIRPath

NonFhirPath assertions (comprising of XPath and JSONPath) can be evaluated against payloads of both `request` and `response` variables offered by the Touchstone Rules-Engine. Both XPath assertions and JsonPath assertions run significantly faster than FHIRPath assertions.

Both XPath and JSONPath expressions are needed for nonFhirPath assertions. Touchstone will use the XPath expression if the payload is XML and will use the JSONPath expression if it's JSON. This relieves the rule author from checking the content type of the payload which needs to be done for XPath assertion and JsonPath assertion.

- **assertNonFhirPathContains**(*xpath*, *jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression contains the provided *expectedValue*. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertNonFhirPathContains("entry[2]/resource/Patient/name/family
↪", "entry[1].resource.name.family", "Chalmers")
```

  - Equivalent to each of these:

```
assertNonFhirPathContains("entry[2]/resource/Patient/name/family",
↪"entry[1].resource.name.family", "Chalmers", response)
```

```
// jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertNonFhirPath("entry[2]/resource/Patient/name/family", "entry[1].
↪resource.name.family", "Chalmers", "contains", response)
```

```
def family = response.getNonFhirPathValue("entry[2]/resource/Patient/name/
↪family", "entry[1].resource.name.family")

assert family.contains("Chalmers"): "The actual value \""+family+"\" did␣
↪not contain the expected value
  \"Chalmers\" for \"entry[1].resource.name.family\" in response."
```

```
def family = response.nonFhirPath("entry[2]/resource/Patient/name/family",
↪"entry[1].resource.name.family").value

assert contains("Chalmers", family): "The actual value \""+family+"\" did␣
↪not contain the expected value
  \"Chalmers\"  for \"entry[1].resource.name.family\" in response."
```

    Notice how the value of NonFhirPath evaluation can be stored in a variable. This is useful when you want to develop more complicated rule scripts where the assertions involve multiple comparisons.

- **assertNonFhirPathNotContains**(*xpath*, *jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression does not contain the provided *expectedValue*. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertNonFhirPathNotContains("entry[2]/resource/Patient/name/
↪family", "entry[1].resource.name.family", "Chalmers")
```

- Equivalent to each of these:

  *
```
assertNonFhirPathNotContains("entry[2]/resource/Patient/name/family",
↪"entry[1].resource.name.family", "Chalmers", response)
```

  *
```
// jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertNonFhirPath("entry[2]/resource/Patient/name/family", "entry[1].
↪resource.name.family", "Chalmers", "notContains", response)
```

  *
```
def family = response.getNonFhirPathValue("entry[2]/resource/Patient/name/
↪family", "entry[1].resource.name.family")

assert !family.contains("Chalmers"): "The actual value \""+family+"\"␣
↪contained the expected value
  \"Chalmers\" for \"entry[1].resource.name.family\" with operator
↪'notContains' in response."
```

  *
```
def family = response.nonFhirPath("entry[2]/resource/Patient/name/family",
↪"entry[1].resource.name.family").value

assert notContains("Chalmers", family): "The actual value \""+family+"\"␣
↪contained the expected value
  \"Chalmers\"  for \"entry[1].resource.name.family\" with operator
↪'notContains' in response."
```

- **assertNonFhirPathEmpty**(*jsonpath*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression is absent or empty.
  - Example:

```
response.assertNonFhirPathEmpty("entry[1]/resource/Patient/photo/title",
↪"entry[0].resource.photo.title")
```

  - Equivalent to these:

```
assertNonFhirPathEmpty("entry[1]/resource/Patient/photo/title",
↪"entry[0].resource.photo.title", response)
```

```
response.jsonPath("entry[1]/resource/Patient/photo/title", "entry[0].
↪resource.photo.title").empty();
```

```
def title = response.getNonFhirPathValue("entry[1]/resource/Patient/
↪photo/title", "entry[0].resource.photo.title")

assert !title: "Expected title to be absent but was present in response"
```

```
def title = response.jsonPath("entry[1]/resource/Patient/photo/title",
↪"entry[0].resource.photo.title").value

assert empty(title): "Expected title to be absent but was present in␣
↪response"
```

- **assertNonFhirPathNotEmpty**(*jsonpath*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression is present and not empty. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
response.assertNonFhirPathNotEmpty("entry[1]/resource/Patient/birthDate",
↪ "entry[0].resource.birthDate")
```

  - Equivalent to these:

```
assertNonFhirPathNotEmpty("entry[1]/resource/Patient/birthDate",
↪"entry[0].resource.birthDate", response)
```

```
response.nonFhirPath("entry[1]/resource/Patient/birthDate", "entry[0].
↪resource.birthDate").notEmpty();
```

```
def title = response.getNonFhirPathValue("entry[1]/resource/Patient/
↪birthDate", "entry[0].resource.birthDate")

assert title: "Expected birthDate to be absent but was present in␣
↪response"
```

```
def title = response.nonFhirPath("entry[1]/resource/Patient/birthDate",
↪"entry[0].resource.birthDate").value

assert notEmpty(title): "Expected birthDate to be absent but was present␣
↪in response"
```

- **assertNonFhirPathEquals**(*xpath*, *jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression matches the provided *expectedValue*. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that the value of the family element of the first entry is
↪'Chalmers'
response.assertNonFhirPathEquals("entry[1]/resource/Patient/name/family",
↪ "Chalmers")
```

  - Equivalent to each of these:

```
assertNonFhirPathEquals("entry[1]/resource/Patient/name/family", "Chalmers",
↪ response)
```

```
* // xPath, expectedValue, and operator can be passed as parameters from test␣
  ↪script.
  assertNonFhirPath("entry[1]/resource/Patient/name/family", "Chalmers",
  ↪"equals", response)
```

```
* def family = response.getNonFhirPathValue("entry[1]/resource/Patient/name/
  ↪family")

  assert family.equals("Chalmers"): "The actual value \""+family+"\" did not␣
  ↪match the expected value
  \"Chalmers\" for \"entry[1]/resource/Patient/name/family\" in response."
```

```
* def family = response.nonFhirPath("entry[1]/resource/Patient/name/family").
  ↪getValue()

  assert family == "Chalmers": "The actual value \""+family+"\" did not match␣
  ↪the expected value
    \"Chalmers\" for \"entry[1]/resource/Patient/name/family\" in response."
```

```
* def family = response.xPath("entry[1]/resource/Patient/name/family").value

  assert equals("Chalmers", family): "The actual value \""+family+"\" did not␣
  ↪match the expected value
    \"Chalmers\"  for \"entry[1]/resource/Patient/name/family\" in response."
```

- **assertNonFhirPathEquals**(*xpath*, *jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression matches the provided *expectedValue*. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that the value of the family element of the first entry is
↪'Chalmers'
response.assertNonFhirPathEquals("entry[1]/resource/Patient/name/family",
↪ "entry[0].resource.name.family", "Chalmers")
```

  - Equivalent to each of these:

```
* assertNonFhirPathEquals("entry[1]/resource/Patient/name/family", "entry[0].
  ↪resource.name.family", "Chalmers", response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertNonFhirPath("entry[1]/resource/Patient/name/family", "entry[0].
  ↪resource.name.family", "Chalmers", "equals", response)
```

```
* def family = response.getNonFhirPathValue("entry[1]/resource/Patient/name/
  ↪family", "entry[0].resource.name.family")

  assert family.equals("Chalmers"): "The actual value \""+family+"\" did not␣
  ↪match the expected value
  \"Chalmers\" for \"entry[0].resource.name.family\" in response."
```

```
* def family = response.nonFhirPath("entry[1]/resource/Patient/name/family",
  ↪"entry[0].resource.name.family").getValue()

  assert family == "Chalmers": "The actual value \""+family+"\" did not match␣
  ↪the expected value
    \"Chalmers\" for \"entry[0].resource.name.family\" in response."
```

```
* def family = response.nonFhirPath("entry[1]/resource/Patient/name/family",
  ↪"entry[0].resource.name.family").value

  assert equals("Chalmers", family): "The actual value \""+family+"\" did not␣
  ↪match the expected value
    \"Chalmers\"  for \"entry[0].resource.name.family\" in response."
```

- **assertNonFhirPathNotEquals**(*xpath*, *jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression does not match the provided *expectedValue*. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that values of the family element of the first entry is not
↪'Chalmers'
response.assertNonFhirPathNotEquals("entry[1]/resource/Patient/name/
↪family", "entry[0].resource.name.family", "Chalmers")
```

  - Equivalent to each of these:

```
* assertNonFhirPathNotEquals("entry[1]/resource/Patient/name/family",
  ↪"entry[0].resource.name.family", "Chalmers", response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertNonFhirPath("entry[1]/resource/Patient/name/family", "entry[0].
  ↪resource.name.family", "Chalmers", "notEquals", response)
```

```
* def family = response.getNonFhirPathValue("entry[1]/resource/Patient/name/
  ↪family", "entry[0].resource.name.family")

  assert !family.equals("Chalmers"): "The actual value \""+family+"\" matched
     the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
  ↪with operator
       'notEquals' in response."
```

```
* def family = response.nonFhirPath("entry[1]/resource/Patient/name/family",
  ↪"entry[0].resource.name.family").getValue()

  assert family != "Chalmers": "The actual value \""+family+"\" matched
     the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
  ↪with operator
       'notEquals' in response."
```

```
* def family = response.nonFhirPath("entry[1]/resource/Patient/name/family",
↪"entry[0].resource.name.family").value

assert notEquals("Chalmers", family): "The actual value \""+family+"\"␣
↪matched
  the expected value \"Chalmers\" for \"entry[0].resource.name.family\"␣
↪with operator
     'notEquals' in response."
```

- **assertNonFhirPathGreaterThan**(*xpath*, *jsonpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression is greater than the provided *expectedValue*. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that resource id of the second entry is greater than 1100 in␣
↪the response
response.assertNonFhirPathGreaterThan("entry[2]/resource/Patient/id",
↪"entry[1].resource.id", 1100)
```

  - Equivalent to each of these:

```
* assertNonFhirPathGreaterThan("entry[2]/resource/Patient/id", "entry[1].
↪resource.id", 1100, response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertNonFhirPath("entry[2]/resource/Patient/id", "entry[1].resource.id",
↪"1100", "greaterThan", response)
```

```
* def id = response.getNonFhirPathValue("entry[2]/resource/Patient/id",
↪"entry[1].resource.id")

assert id.toInteger() > 1100: "Expected \"entry[1].resource.id\" to be␣
↪greater than 5000
    but was "+id+" in response"
```

```
* def id = response.nonFhirPath("entry[2]/resource/Patient/id", "entry[1].
↪resource.id").getValue()

assert (id as Integer) > 1100: "Expected \"entry[1].resource.id\" to be␣
↪greater than 5000
   but was "+id+" in response"
```

```
* def id = response.nonFhirPath("entry[2]/resource/Patient/id", "entry[1].
↪resource.id").value

assert greaterThan(1100, id): "Expected \"entry[1].resource.id\" to be␣
↪greater than 5000
   but was "+id+" in response"
```

- **assertNonFhirPathLessThan**(*xpath*, *jsonpath*, *expectedValue*)

- Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression is less than the provided *expectedValue*. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
- Example:

```
// Asserts that resource id of the first entry is less than 1100 in the␣
↪response
response.assertNonFhirPathLessThan("entry[1]/resource/Patient/id",
↪"entry[0].resource.id", 1100)
```

- Equivalent to each of these:

```
assertNonFhirPathLessThan("entry[1]/resource/Patient/id", "entry[0].
↪resource.id", 1100, response)
```

```
// jsonPath, expectedValue, and operator can be passed as parameters from␣
↪test script.
assertNonFhirPath("entry[1]/resource/Patient/id", "entry[0].resource.id",
↪"1100", "lessThan", response)
```

```
def id = response.getNonFhirPathValue("entry[1]/resource/Patient/id",
↪"entry[0].resource.id")

assert id.toInteger() < 1100: "Expected \"entry[0].resource.id\" to be less␣
↪than 5000
    but was "+id+" in response"
```

```
def id = response.nonFhirPath("entry[1]/resource/Patient/id", "entry[0].
↪resource.id").getValue()

assert (id as Integer) < 1100: "Expected \"entry[0].resource.id\" to be␣
↪less than 5000
   but was "+id+" in response"
```

```
def id = response.nonFhirPath("entry[1]/resource/Patient/id", "entry[0].
↪resource.id").value

assert lessThan(1100, id): "Expected \"entry[0].resource.id\" to be less␣
↪than 5000
   but was "+id+" in response"
```

- **assertNonFhirPathIn**(*xpath*, *jsonpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression is one of the provided *expectedValues* where each value is separated by a comma. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertNonFhirPathIn("entry[2]/resource/Patient/id", "entry[1].
↪resource.id", "1100,1101,1102")
```

  - Equivalent to each of these:

```
* assertNonFhirPathIn("entry[2]/resource/Patient/id", "entry[1].resource.id",
  ↪"1100,1101,1102", response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertNonFhirPath("entry[2]/resource/Patient/id", "entry[1].resource.id",
  ↪"1100,1101,1102", "in", response)
```

```
* def id = response.getNonFhirPathValue("entry[2]/resource/Patient/id",
  ↪"entry[1].resource.id")

  assert id in ["1100", "1101", "1102"]: "Expected one of the values in [1100,
  ↪ 1101, 1102]
     for \"entry[1].resource.id\" but encountered \""+id+"\" in response."
```

```
* def id = response.nonFhirPath("entry[2]/resource/Patient/id", "entry[1].
  ↪resource.id").value

  assert isIn("1100,1101,1102", id): "Expected one of the values in [1100,␣
  ↪1101, 1102] for
     \"entry[1].resource.id\" but encountered \""+id+"\" in response."
```

- **assertNonFhirPathNotIn**(*xpath*, *jsonpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *xpath* or *jsonpath* expression is none of the provided *expectedValues* where each value is separated by a comma. Touchstone will use *xpath* if the payload is XML and *jsonpath* if it is JSON.
  - Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertNonFhirPathNotIn("entry[2]/resource/Patient/id",
↪"entry[1].resource.id", "1100,1101,1102")
```

  - Equivalent to each of these:

```
* assertNonFhirPathNotIn("entry[2]/resource/Patient/id", "entry[1].resource.id
  ↪", "1100,1101,1102", response)
```

```
* // jsonPath, expectedValue, and operator can be passed as parameters from␣
  ↪test script.
  assertNonFhirPath("entry[2]/resource/Patient/id", "entry[1].resource.id",
  ↪"1100,1101,1102", "notIn", response)
```

```
* def id = response.getNonFhirPathValue("entry[2]/resource/Patient/id",
  ↪"entry[1].resource.id")

  assert !(id in ["1100", "1101", "1102"]): "Expected none of the values in␣
  ↪[1100, 1101, 1102]
     for \"entry[1].resource.id\" but encountered \""+id+"\" with operator
  ↪'notIn' in response."
```

```
* def id = response.nonFhirPath("entry[2]/resource/Patient/id", "entry[1].
  ↪resource.id").value

  assert isNotIn("1100,1101,1102", id): "Expected none of the values in [1100,
  ↪ 1101, 1102] for
    \"entry[1].resource.id\" but encountered \""+id+"\" with operator 'notIn
  ↪' in response."
```

9.8.4.6.4 XPath

XPath assertions can be evaluated against payloads of both `request` and `response` variables offered by the Touchstone Rules-Engine. Both XPath assertions and JsonPath assertions run significantly faster than FHIRPath assertions. Separate XPath and JSONPath expressions are needed though for XML and JSON content while only one FHIRPath expression is needed for both XML and JSON content.

- **assertXPathContains**(*xpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* expression contains the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertXPathContains("entry[2]/resource/Patient/name/family",
↪"Chalmers")
```

  - Equivalent to each of these:

```
* assertXPathContains("entry[2]/resource/Patient/name/family", "Chalmers",␣
  ↪response)
```

```
* // xPath, expectedValue, and operator can be passed as parameters from test␣
  ↪script.
  assertXPath("entry[2]/resource/Patient/name/family", "Chalmers", "contains",
  ↪ response)
```

```
* def family = response.getXPathValue("entry[2]/resource/Patient/name/family")

  assert family.contains("Chalmers"): "The actual value \""+family+"\" did␣
  ↪not contain the expected value
    \"Chalmers\" for \"entry[2]/resource/Patient/name/family\" in response."
```

```
* def family = response.xPath("entry[2]/resource/Patient/name/family").value

  assert contains("Chalmers", family): "The actual value \""+family+"\" did␣
  ↪not contain the expected value
    \"Chalmers\"  for \"entry[2]/resource/Patient/name/family\" in response."
```

    Notice how the value of XPath evaluation can be stored in a variable. This is useful when you want to develop more complicated rule scripts where the assertions involve multiple comparisons.

- **assertXPathNotContains**(*xpath*, *expectedValue*)

- Asserts that the evaluated value of the provided *xpath* expression does not contain the provided *expectedValue*.
- Example:

```
// Asserts that the value of the family element of the second entry␣
↪contains 'Chalmers'
response.assertXPathNotContains("entry[2]/resource/Patient/name/family",
↪"Chalmers")
```

- Equivalent to each of these:

  *
```
assertXPathNotContains("entry[2]/resource/Patient/name/family", "Chalmers",␣
↪response)
```

  *
```
// xPath, expectedValue, and operator can be passed as parameters from test␣
↪script.
assertXPath("entry[2]/resource/Patient/name/family", "Chalmers",
↪"notContains", response)
```

  *
```
def family = response.getXPathValue("entry[2]/resource/Patient/name/family")

assert !family.contains("Chalmers"): "The actual value \""+family+"\"␣
↪contained the expected value
   \"Chalmers\" for \"entry[2]/resource/Patient/name/family\" with operator
↪'notContains' in response."
```

  *
```
def family = response.xPath("entry[2]/resource/Patient/name/family").value

assert notContains("Chalmers", family): "The actual value \""+family+"\"␣
↪contained the expected value
   \"Chalmers\"  for \"entry[2]/resource/Patient/name/family\" with␣
↪operator 'notContains' in response."
```

- **assertXPathEmpty**(*xpath*)
  - Asserts that the evaluated value of the provided *xpath* expression is absent or empty.
  - Example:

```
response.assertXPathEmpty("entry[1]/resource/Patient/photo/title")
```

  - Equivalent to these:

```
assertXPathEmpty("entry[1]/resource/Patient/photo/title", response)
```

```
response.fhirPath("entry[1]/resource/Patient/photo/title").empty();
```

```
def title = response.getXPathValue("entry[1]/resource/Patient/photo/title
↪")

assert !title: "Expected title to be absent but was present in response"
```

```
def title = response.xPath("entry[1]/resource/Patient/photo/title").value

assert empty(title): "Expected title to be absent but was present in␣
↪response"
```

- **assertXPathNotEmpty**(*xpath*)
  - Asserts that the evaluated value of the provided *xpath* expression is present and not empty.
  - Example:

```
response.assertXPathNotEmpty("entry[1]/resource/Patient/birthDate")
```

  - Equivalent to these:

```
assertXPathNotEmpty("entry[1]/resource/Patient/birthDate", response)
```

```
response.xPath("entry[1]/resource/Patient/birthDate").notEmpty();
```

```
def title = response.getXPathValue("entry[1]/resource/Patient/birthDate")

assert title: "Expected birthDate to be absent but was present in␣
↪response"
```

```
def title = response.xPath("entry[1]/resource/Patient/birthDate").value

assert notEmpty(title): "Expected birthDate to be absent but was present␣
↪in response"
```

- **assertXPathEquals**(*xpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* expression matches the provided *expectedValue*.
  - Example:

```
// Asserts that the value of the family element of the first entry is
↪'Chalmers'
response.assertXPathEquals("entry[1]/resource/Patient/name/family",
↪"Chalmers")
```

  - Equivalent to each of these:
    *

```
assertXPathEquals("entry[1]/resource/Patient/name/family", "Chalmers",␣
↪response)
```

    *

```
// xPath, expectedValue, and operator can be passed as parameters from test␣
↪script.
assertXPath("entry[1]/resource/Patient/name/family", "Chalmers", "equals",␣
↪response)
```

    *

```
def family = response.getXPathValue("entry[1]/resource/Patient/name/family")

assert family.equals("Chalmers"): "The actual value \""+family+"\" did not␣
↪match the expected value
\"Chalmers\" for \"entry[1]/resource/Patient/name/family\" in response."
```

```
* def family = response.xPath("entry[1]/resource/Patient/name/family").
  ↪getValue()

  assert family == "Chalmers": "The actual value \""+family+"\" did not match␣
  ↪the expected value
    \"Chalmers\" for \"entry[1]/resource/Patient/name/family\" in response."
```

```
* def family = response.xPath("entry[1]/resource/Patient/name/family").value

  assert equals("Chalmers", family): "The actual value \""+family+"\" did not␣
  ↪match the expected value
    \"Chalmers\"  for \"entry[1]/resource/Patient/name/family\" in response."
```

- **assertXPathNotEquals**(*xpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* expression does not match the provided *expectedValue*.
  - Example:

```
// Asserts that values of the family element of the first entry is not
↪'Chalmers'
response.assertXPathNotEquals("entry[1]/resource/Patient/name/family",
↪"Chalmers")
```

  - Equivalent to each of these:

```
* assertXPathNotEquals("entry[1]/resource/Patient/name/family", "Chalmers",␣
  ↪response)
```

```
* // xPath, expectedValue, and operator can be passed as parameters from test␣
  ↪script.
  assertXPath("entry[1]/resource/Patient/name/family", "Chalmers", "notEquals
  ↪", response)
```

```
* def family = response.getXPathValue("entry[1]/resource/Patient/name/family")

  assert !family.equals("Chalmers"): "The actual value \""+family+"\" matched
     the expected value \"Chalmers\" for \"entry[1]/resource/Patient/name/
  ↪family\" with operator
        'notEquals' in response."
```

```
* def family = response.xPath("entry[1]/resource/Patient/name/family").
  ↪getValue()

  assert family != "Chalmers": "The actual value \""+family+"\" matched
     the expected value \"Chalmers\" for \"entry[1]/resource/Patient/name/
  ↪family\" with operator
        'notEquals' in response."
```

```
* def family = response.xPath("entry[1]/resource/Patient/name/family").value

  assert notEquals("Chalmers", family): "The actual value \""+family+"\"␣
  ↪matched
```

(continues on next page)

(continued from previous page)

```
  the expected value \"Chalmers\" for \"entry[1]/resource/Patient/name/
↪family\" with operator
     'notEquals' in response."
```

- **assertXPathGreaterThan**(*xpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* expression is greater than the provided *expectedValue*.
  - Example:

```
// Asserts that resource id of the second entry is greater than 1100 in␣
↪the response
response.assertXPathGreaterThan("entry[2]/resource/Patient/id", 1100)
```

  - Equivalent to each of these:
    *
```
assertXPathGreaterThan("entry[2]/resource/Patient/id", 1100, response)
```

    *
```
// xPath, expectedValue, and operator can be passed as parameters from test␣
↪script.
assertXPath("entry[2]/resource/Patient/id", "1100", "greaterThan", response)
```

    *
```
def id = response.getXPathValue("entry[2]/resource/Patient/id")

assert id.toInteger() > 1100: "Expected \"entry[2]/resource/Patient/id\" to␣
↪be greater than 5000
    but was "+id+" in response"
```

    *
```
def id = response.xPath("entry[2]/resource/Patient/id").getValue()

assert (id as Integer) > 1100: "Expected \"entry[2]/resource/Patient/id\"␣
↪to be greater than 5000
   but was "+id+" in response"
```

    *
```
def id = response.xPath("entry[2]/resource/Patient/id").value

assert greaterThan(1100, id): "Expected \"entry[2]/resource/Patient/id\" to␣
↪be greater than 5000
   but was "+id+" in response"
```

- **assertXPathLessThan**(*xpath*, *expectedValue*)
  - Asserts that the evaluated value of the provided *xpath* expression is less than the provided *expectedValue*.
  - Example:

```
// Asserts that resource id of the first entry is less than 1100 in the␣
↪response
response.assertXPathLessThan("entry[1]/resource/Patient/id", 1100)
```

  - Equivalent to each of these:
    *
```
assertXPathLessThan("entry[1]/resource/Patient/id", 1100, response)
```

```
* // xPath, expectedValue, and operator can be passed as parameters from test␣
  ↪script.
  assertXPath("entry[1]/resource/Patient/id", "1100", "lessThan", response)
```

```
* def id = response.getXPathValue("entry[1]/resource/Patient/id")

  assert id.toInteger() < 1100: "Expected \"entry[1]/resource/Patient/id\" to␣
  ↪be less than 5000
      but was "+id+" in response"
```

```
* def id = response.xPath("entry[1]/resource/Patient/id").getValue()

  assert (id as Integer) < 1100: "Expected \"entry[1]/resource/Patient/id\"␣
  ↪to be less than 5000
     but was "+id+" in response"
```

```
* def id = response.xPath("entry[1]/resource/Patient/id").value

  assert lessThan(1100, id): "Expected \"entry[1]/resource/Patient/id\" to be␣
  ↪less than 5000
     but was "+id+" in response"
```

- **assertXPathIn**(*xpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *xpath* expression is one of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertXPathIn("entry[2]/resource/Patient/id", "1100,1101,1102")
```

  - Equivalent to each of these:

```
* assertXPathIn("entry[2]/resource/Patient/id", "1100,1101,1102", response)
```

```
* // xPath, expectedValue, and operator can be passed as parameters from test␣
  ↪script.
  assertXPath("entry[2]/resource/Patient/id", "1100,1101,1102", "in",␣
  ↪response)
```

```
* def id = response.getXPathValue("entry[2]/resource/Patient/id")

  assert id in ["1100", "1101", "1102"]: "Expected one of the values in [1100,
  ↪ 1101, 1102]
     for \"entry[2]/resource/Patient/id\" but encountered \""+id+"\" in␣
  ↪response."
```

```
* def id = response.xPath("entry[2]/resource/Patient/id").value
```

(continues on next page)

(continued from previous page)

```
assert isIn("1100,1101,1102", id): "Expected one of the values in [1100,␣
↪1101, 1102] for
   \"entry[2]/resource/Patient/id\" but encountered \""+id+"\" in response."
```

- **assertXPathNotIn**(*xpath*, *expectedValues*)
  - Asserts that the evaluated value of the provided *xpath* expression is none of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that resource id of the second entry is either 1100 or 1101␣
↪or 1102
response.assertXPathNotIn("entry[2]/resource/Patient/id", "1100,1101,1102
↪")
```

  - Equivalent to each of these:

```
assertXPathNotIn("entry[2]/resource/Patient/id", "1100,1101,1102", response)
```

```
// xPath, expectedValue, and operator can be passed as parameters from test␣
↪script.
assertXPath("entry[2]/resource/Patient/id", "1100,1101,1102", "notIn",␣
↪response)
```

```
def id = response.getXPathValue("entry[2]/resource/Patient/id")

assert !(id in ["1100", "1101", "1102"]): "Expected none of the values in␣
↪[1100, 1101, 1102]
   for \"entry[2]/resource/Patient/id\" but encountered \""+id+"\" with␣
↪operator 'notIn' in response."
```

```
def id = response.xPath("entry[2]/resource/Patient/id").value

assert isNotIn("1100,1101,1102", id): "Expected none of the values in [1100,
↪ 1101, 1102] for
   \"entry[2]/resource/Patient/id\" but encountered \""+id+"\" with␣
↪operator 'notIn' in response."
```

9.8.4.7 Profile

The assertion below can be performed on `request` and `response` variables offered by the Touchstone Rules-Engine. This is similar to what the TestScript does via the validateProfileId assertion.

See the test script Patient-server-id-json for an example on how validateProfileId assertion is performed in TestScript and how the profile is defined:

- ValidateProfile assertion definition in TestScript:

```
<action>
    <assert>
        <description value="Validate that the returned resource conforms to the corresponding FHIR bundle profile."/>
        <direction value="response"/>
        <validateProfileId value="bundle-profile"/>
        <warningOnly value="false"/>
    </assert>
</action>
```

- Profile definition in TestScript:

```
<profile id="bundle-profile">
    <reference value="http://hl7.org/fhir/StructureDefinition/Bundle"/>
</profile>
```

- **assertValidWithProfileId**(*validateProfileId*)
  - Asserts that the response is valid according to the Profile specified by validateProfileId.
  - Example:

```
response.assertValidWithProfileId("bundle-profile")
```

  - Equivalent to these:

```
response.validateWithProfileId("bundle-profile")
```

```
assertValidWithProfileId("bundle-profile", response)
```

```
validateWithProfileId("bundle-profile", response)
```

- **assertValidWithProfile**(*validateProfileReference*)
  - Rather than specifying a profile id that's defined in the TestScript, we can validate using the profile URI directly. This call asserts that the request or response is valid according to the Profile URI specified by *validateProfileReference*.
  - Example:

```
response.assertValidWithProfile("http://hl7.org/fhir/StructureDefinition/
↪Bundle")
```

  - Equivalent to these:

```
response.validateWithProfile("http://hl7.org/fhir/StructureDefinition/
↪Bundle")
```

```
assertValidWithProfile("http://hl7.org/fhir/StructureDefinition/Bundle",␣
↪response)
```

```
validateWithProfile("http://hl7.org/fhir/StructureDefinition/Bundle",␣
↪response)
```

- **assertValidXmlExtractWithProfileId**(*xpath*, *validateProfileId*)
  - Asserts that the XML content extracted from the request or response using the specified *xpath* expression is valid according to the Profile specified by validateProfileId. Please refer to XPath for more information on XPath expressions.
  - Example:

```
response.assertValidXmlExtractWithProfileId("Bundle/entry[1]/resource/
↪Patient", "bundle-profile")
```

- Equivalent to these:

```
response.extractXmlAndValidateWithProfileId("Bundle/entry[1]/resource/
↪Patient", "bundle-profile")
```

```
assertValidXmlExtractWithProfileId("Bundle/entry[1]/resource/Patient",
↪"bundle-profile", response)
```

```
extractXmlAndValidateWithProfileId("Bundle/entry[1]/resource/Patient",
↪"bundle-profile", response)
```

- **assertValidXmlExtractWithProfile**(*xpath*, *validateProfileReference*)
  - Asserts that the XML content extracted from the request or response using the specified *xpath* expression is valid according to the Profile URI specified by *validateProfileReference*. Please refer to XPath for more information on XPath expressions.
  - Example:

```
response.assertValidXmlExtractWithProfile("Bundle/entry[1]/resource/
↪Patient",
        "http://hl7.org/fhir/StructureDefinition/Bundle")
```

  - Equivalent to these:

```
response.extractXmlAndValidateWithProfile("Bundle/entry[1]/resource/
↪Patient",
        "http://hl7.org/fhir/StructureDefinition/Bundle")
```

```
assertValidXmlExtractWithProfile("Bundle/entry[1]/resource/Patient",
        "http://hl7.org/fhir/StructureDefinition/Bundle", response)
```

```
extractXmlAndValidateWithProfile("Bundle/entry[1]/resource/Patient",
        "http://hl7.org/fhir/StructureDefinition/Bundle", response)
```

- **assertValidJsonExtractWithProfileId**(*jsonpath*, *validateProfileId*)
  - Asserts that the JSON content extracted from the request or response using the specified *jsonpath* expression is valid according to the Profile specified by validateProfileId. Please refer to JSONPath for more information on JSONPath expressions.
  - Example:

```
response.assertValidJsonExtractWithProfileId("context.orders[0]", "proc-
↪request-profile")
```

  - Equivalent to these:

```
response.extractJsonAndValidateWithProfileId("context.orders[0]", "proc-
↪request-profile")
```

```
assertValidJsonExtractWithProfileId("context.orders[0]", "proc-request-
↪profile", response)
```

```
extractJsonAndValidateWithProfileId("context.orders[0]", "proc-request-
↪profile", response)
```

- **assertValidJsonExtractWithProfile**(*jsonpath*, *validateProfileReference*)
  - Asserts that the JSON content extracted from the request or response using the specified *jsonpath* expression is valid according to the Profile URI specified by *validateProfileReference*. Please refer to JSONPath for more information on JSONPath expressions.
  - Example:

```
response.assertValidJsonExtractWithProfile("context.orders[0]",
         "http://hl7.org/fhir/StructureDefinition/ProcedureRequest")
```

  - Equivalent to these:

```
response.extractJsonAndValidateWithProfile("context.orders[0]",
         "http://hl7.org/fhir/StructureDefinition/ProcedureRequest")
```

```
assertValidJsonExtractWithProfile("context.orders[0]",
         "http://hl7.org/fhir/StructureDefinition/ProcedureRequest",␣
↪response)
```

```
extractJsonAndValidateWithProfile("context.orders[0]",
         "http://hl7.org/fhir/StructureDefinition/ProcedureRequest",␣
↪response)
```

9.8.4.8 Request Method

The following assertions can be performed on `request` variable offered by the Touchstone Rules-Engine.

They would validate the request method. These assertions would make sense in Peer-to-Peer testing.

- **assertRequestMethodEquals**(*expectedValue*)
  - Asserts that the method of the request matches the provided *expectedValue*
  - Example:

```
request.assertRequestMethodEquals("GET")
```

  - Equivalent to these:

```
assertRequestMethodEquals("GET", request)
```

```
// expected method and operator can be passed as parameters from test␣
↪script.
assertRequestMethod("GET", "equals", request)
```

```
assert request.getMethod()=="GET": "The actual value \""+request.
↪getMethod()+"\" did not
   match the expected value \"GET\" for HTTP method in request"
```

```
assert equals("GET", request.getMethod()): "The actual value \""+request.
↪getMethod()+"\"
   did not match the expected value \"GET\" for HTTP method in request"
```

- **assertRequestMethodNotEquals**(*expectedValue*)
  - Asserts that the method of the request does not match the provided *expectedValue*
  - Example:

```
request.assertRequestMethodNotEquals("PUT")
```

  - Equivalent to these:

```
assertRequestMethodNotEquals("PUT", request)
```

```
// expected method and operator can be passed as parameters from test␣
↪script.
assertRequestMethod("PUT", "notEquals", request)
```

```
assert request.getMethod()!="PUT": "The actual value \""+request.
↪getMethod()+"\" matched
   the expected value \"PUT\" for HTTP method with operator 'notEquals'␣
↪in request"
```

```
assert notEquals("PUT", request.getMethod()): "The actual value \"
↪"+request.getMethod()
   +"\" matched the expected value \"PUT\" for HTTP method with operator
↪'notEquals'
      in request"
```

- **assertRequestMethodIn**(*expectedValues*)
  - Asserts that the method of the request is one of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
request.assertRequestMethodIn("POST, PUT")
```

  - Equivalent to these:

```
assertRequestMethodIn("POST, PUT", request)
```

```
// expected method and operator can be passed as parameters from test␣
↪script.
assertRequestMethod("POST, PUT", "in", request)
```

```
assert request.getMethod() in ["GET, PUT"]: "Expected one of the values␣
↪in [GET, PUT] for
   HTTP method but encountered \""+request.getMethod()+"\" in request"
```

```
assert isIn("GET, PUT", request.getMethod()): "Expected one of the␣
↪values in [GET, PUT] for
   HTTP method but encountered \""+request.getMethod()+"\" in request"
```

- **assertRequestMethodNotIn**(*expectedValues*)
  - Asserts that the method of the request is none of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
request.assertRequestMethodNotIn("POST, PUT")
```

  - Equivalent to these:

```
assertRequestMethodNotIn("POST, PUT", request)
```

```
// expected method and operator can be passed as parameters from test␣
↪script.
assertRequestMethod("POST, PUT", "notIn", request)
```

```
assert !(request.getMethod() in ["POST", "PUT"]): "Expected none of the␣
↪values in [POST, PUT] for
   HTTP method but encountered \""+request.getMethod()+"\" with operator
↪'notIn' in request"
```

```
assert isNotIn("POST, PUT", request.getMethod()): "Expected none of the␣
↪values in [POST, PUT] for
   HTTP method but encountered \""+request.getMethod()+"\" with operator
↪'notIn' in request"
```

9.8.4.9 Request URL

The following assertions can be performed on `request` variable offered by the Touchstone Rules-Engine.

They would validate the request URL. These assertions would make sense in Peer-to-Peer testing.

- **assertRequestURLEquals**(*expectedValue*)
  - Asserts that the URL of the request matches the provided *expectedValue*
  - Example:

```
request.assertRequestURLEquals("http://touchstone.aegis.net:57084/fhir3-
↪0-1/Patient?name=Connectathon15")
```

  - Equivalent to these:

```
assertRequestURLEquals("http://touchstone.aegis.net:57084/fhir3-0-1/
↪Patient?name=Connectathon15", request)
```

```
// expected URL and operator can be passed as parameters from test␣
↪script.
assertRequestURL("http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15",
      "equals", request)
```

```
assert request.getURL()=="http://touchstone.aegis.net:57084/fhir3-0-1/
↪Patient?name=Connectathon15": "The
   actual value \""+request.getURL()+"\" did not match the expected value
      \"http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15\" for URL in request"
```

```
assert equals("http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15",
   request.getURL()): "The actual value \""+request.getURL()+"\" did not␣
↪match the expected value
      \"http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15\" for URL in request"
```

- **assertRequestURLNotEquals(***expectedValue***)**
  - Asserts that the URL of the request does not match the provided *expectedValue*
  - Example:

```
request.assertRequestURLNotEquals(
            "http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15")
```

  - Equivalent to these:

```
assertRequestURLNotEquals("http://touchstone.aegis.net:57084/fhir3-0-1/
↪Patient?name=Connectathon15",
   request)
```

```
// expected URL and operator can be passed as parameters from test␣
↪script.
assertRequestURL("http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15",
                  "notEquals", request)
```

```
assert request.getURL()=="http://touchstone.aegis.net:57084/fhir3-0-1/
↪Patient?name=Connectathon15":
   "The actual value \""+request.getURL()+"\" matched the expected value
   \"http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15\" for URL with
      operator 'notEquals' in request"
```

```
assert notEquals("http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15",
   request.getURL()): "The actual value \""+request.getURL()+"\" matched␣
↪the expected value
      \"http://touchstone.aegis.net:57084/fhir3-0-1/Patient?
↪name=Connectathon15\" for URL with
```

(continues on next page)

(continued from previous page)

```
        operator 'notEquals' in request"
```

- **assertRequestURLContains**(*expectedValue*)
  - Asserts that the URL of the request contains the provided *expectedValue*
  - Example:

```
request.assertRequestURLContains("fhir3-0-1/Patient?name=Connectathon15")
```

  - Equivalent to these:

```
assertRequestURLContains("fhir3-0-1/Patient?name=Connectathon15",␣
↪request)
```

```
// expected URL and operator can be passed as parameters from test␣
↪script.
assertRequestURL("fhir3-0-1/Patient?name=Connectathon15", "contains",␣
↪request)
```

```
assert request.getURL().contains("fhir3-0-1/Patient?name=Connectathon15
↪"): "The actual value
   \""+request.getURL()+"\" did not contain the expected value
      \"fhir3-0-1/Patient?name=Connectathon15\" for URL in request"
```

```
assert contains("fhir3-0-1/Patient?name=Connectathon15", request.
↪getURL()): "The actual value
   \""+request.getURL()+"\" did not contain the expected value
   \"fhir3-0-1/Patient?name=Connectathon15\" for URL in request"
```

- **assertRequestURLNotContains**(*expectedValue*)
  - Asserts that the URL of the request does not contain the provided *expectedValue*
  - Example:

```
request.assertRequestURLNotContains("fhir3-0-1/Patient?
↪name=Connectathon15")
```

  - Equivalent to these:

```
assertRequestURLNotContains("fhir3-0-1/Patient?name=Connectathon15",␣
↪request)
```

```
// expected URL and operator can be passed as parameters from test␣
↪script.
assertRequestURL("fhir3-0-1/Patient?name=Connectathon15", "notContains",␣
↪request)
```

```
assert !request.getURL().contains("fhir3-0-1/Patient?name=Connectathon15
↪"): "The actual value
   \""+request.getURL()+"\" contained the expected value \"fhir3-0-1/
↪Patient?name=Connectathon15\"
      for URL with operator 'notContains' in request"
```

```
assert notContains("fhir3-0-1/Patient?name=Connectathon15", request.
↪getURL()): "The actual value
  \""+request.getURL()+"\" contained the expected value \"fhir3-0-1/
↪Patient?name=Connectathon15\"
     for URL with operator 'notContains' in request"
```

9.8.4.9.1 Request URL Prefixes

The *expectedValue* in requestURL can be prefixed for fine-grained assertion of individual parts of the URL.

Prefix	Description	Operators	Example
schemeHostPortPath:	Validates the scheme, host, port, and path.	equals, notEquals	"schemeHostPortPath: http://hostname:11111/fhirSys1"
		contains, notContains	"schemeHostPortPath: http://hostname:11111/fhir"
schemeHostPort:	Validates the scheme, host, and port.	equals, notEquals	"schemeHostPort: http://hostname:11111"
		contains, notContains	"schemeHostPort: http://hostname:111"
schemeHost:	Validates the scheme and host.	equals, notEquals	"schemeHost: http://hostname"
		contains, notContains	"schemeHost: http://hos"
scheme:	Validates the scheme.	equals, notEquals	"scheme: https"
		contains, notContains	"scheme: http"
hostPortPath:	Validates the host, port, and path.	equals, notEquals	"hostPortPath: hostname:11111/fhir"
		contains, notContains	"hostPortPath: hostname:111"
hostPort:	Validates the host and port.	equals, notEquals	"hostPort: hostname:11111"
		contains, notContains	"hostPort: hostname:1"
host:	Validates the host.	equals, notEquals	"host: hostname"
		contains, notContains	"host: hostna"

9.8.4.10 Resource

The following assertions can be performed on `request` and `response` variables offered by the Touchstone Rules-Engine.

They would validate the resource type within the payload.

- **assertResourceEquals(***expectedValue***)**
  - Asserts that the resource type of the payload matches the provided *expectedValue*
  - Example:

```
response.assertResourceEquals("Bundle")
```

  - Equivalent to these:

```
assertResourceEquals("Bundle", response)
```

```
// expected resource type and operator can be passed as parameters from␣
↪test script.
assertResource("Bundle", "equals", response)
```

```
assert response.resource=="Bundle": "The actual value \""+response.
↪resource
   +"\" did not match the expected value \"Bundle\" for resource type in␣
↪response"
```

```
assert equals("Bundle", response.resource): "The actual value \"
↪"+response.resource
   +"\" did not match the expected value \"Bundle\" for resource type in␣
↪response"
```

- **assertResourceNotEquals(***expectedValue***)**
  - Asserts that the resource type of the payload does not match the provided *expectedValue*
  - Example:

```
response.assertResourceNotEquals("Patient")
```

  - Equivalent to these:

```
assertResourceNotEquals("Patient", response)
```

```
// expected resource type and operator can be passed as parameters from␣
↪test script.
assertResource("Patient", "notEquals", response)
```

```
assert response.resource!="Patient": "The actual value \""+response.
↪resource +"\" matched
   the expected value \"Patient\" for resource type with operator
↪'notEquals' in response"
```

```
assert notEquals("Patient", response.resource): "The actual value \"
↪"+response.resource
   +"\" matched the expected value \"Patient\" for resource type with␣
↪operator 'notEquals' in response"
```

- **assertResourceIn**(*expectedValues*)
  - Asserts that the resource type of the payload is one of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
response.assertResourceIn("Bundle, Patient")
```

  - Equivalent to these:

```
assertResourceIn("Bundle, Patient", response)
```

```
// expected resource type and operator can be passed as parameters from␣
↪test script.
assertResource("Bundle, Patient", "in", response)
```

```
assert response.resource in ["Bundle", "Patient"]: "Expected one of the␣
↪values in
   [Bundle, Patient] for resource type but encountered \""+response.
↪resource+"\" in response"
```

```
assert isIn("Bundle, Patient", response.resource): "Expected one of the␣
↪values in
   [Bundle, Patient] for resource type but encountered \""+response.
↪resource+"\"
      in response"
```

- **assertResourceNotIn**(*expectedValues*)
  - Asserts that the resource type of the payload is none of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
response.assertResourceNotIn("Bundle, Patient")
```

  - Equivalent to these:

```
assertResourceNotIn("Bundle, Patient", response)
```

```
// expected resource type and operator can be passed as parameters from␣
↪test script.
assertResource("Bundle, Patient", "notIn", response)
```

```
assert !(response.resource in ["Bundle", "Patient"]): "Expected none of␣
↪the values in
   [Bundle, Patient] for resource type but encountered \""+response.
↪resource+"\" with
   operator 'notIn' in response"
```

```
assert isNotIn("Bundle, Patient", response.resource): "Expected none of␣
↪the values in
   [Bundle, Patient] for resource type but encountered \""+response.
↪resource+"\"
      with operator 'notIn' in response"
```

9.8.4.11 Response Code

The following assertions can be performed on `response` variable offered by the Touchstone Rules-Engine.

They would validate the response status code. See Status Code Definitions and List of HTTP status codes.

- **assertResponseCodeEquals**(*expectedValue*)
  - Asserts that the status code of the response matches the provided *expectedValue*
  - Example:

```
response.assertResponseCodeEquals(200)
```

  - Equivalent to these:

```
assertResponseCodeEquals(200, response)
```

```
// expected status code and operator can be passed as parameters from␣
↪test script.
assertResponseCode(200, "equals", response)
```

```
assert response.responseCodeInt==200: "The actual value \""+response.
↪responseCodeInt+"\"
   did not match the expected value \"200\" for response code in response
↪"
```

```
assert equals(200, response.responseCode): "The actual value \"
↪"+response.responseCode+"\"
   did not match the expected value \"200\" for response code in response
↪"
```

```
assert equals(200, responseCode): "The actual value \""+responseCode+"\"
   did not match the expected value \"200\" for response code in response
↪"
```

Note that in the last example above, we're using the **responseCode** binding directly. See Bindings.

- **assertResponseCodeNotEquals**(*expectedValue*)
  - Asserts that the status code of the response does not match the provided *expectedValue*
  - Example:

```
response.assertResponseCodeNotEquals(200)
```

  - Equivalent to these:

```
assertResponseCodeNotEquals(200, response)
```

```
// expected status code and operator can be passed as parameters from␣
↪test script.
assertResponseCode(200, "notEquals", response)
```

```
assert response.responseCodeInt==200: "The actual value \""+response.
↪responseCodeInt+"\"
   matched the expected value \"200\" for response code with operator
↪'notEquals' in response"
```

```
assert notEquals(200, response.responseCode): "The actual value \"
↪"+response.responseCode+"\"
   matched the expected value \"200\" for response code with operator
↪'notEquals' in response"
```

```
assert notEquals(200, responseCode): "The actual value \""+response.
↪responseCode+"\"
   matched the expected value \"200\" for response code with operator
↪'notEquals' in response"
```

- **assertResponseCodeGreaterThan**(*expectedValue*)
  - Asserts that the status code of the response is greater than the provided *expectedValue*
  - Example:

    ```
    response.assertResponseCodeGreaterThan(399)
    ```

  - Equivalent to each of these:
    * ```
      assertResponseCodeGreaterThan(399, response)
      ```

      ```
      // expected status code and operator can be passed as parameters from test␣
      ↪script.
      assertResponseCode(399, "greaterThan", response)
      ```

    * ```
      assert response.responseCodeInt > 399: "Expected response code to be␣
      ↪greater than 399 but was
            "+response.responseCodeInt+" in response"
      ```

    * ```
      assert greaterThan(399, response.responseCode) : "Expected response code to␣
      ↪be greater than 399
         but was "+response.responseCode+" in response"
      ```

- **assertResponseCodeLessThan**(*expectedValue*)
  - Asserts that the status code of the response is less than the provided *expectedValue*
  - Example:

    ```
    response.assertResponseCodeLessThan(300)
    ```

  - Equivalent to each of these:
    ```

```
* assertResponseCodeLessThan(300, response)
```

```
// expected status code and operator can be passed as parameters from test␣
↪script.
assertResponseCode(300, "lessThan", response)
```

```
* assert response.responseCodeInt < 300: "Expected response code to be less␣
↪than 300 but was
      "+response.responseCodeInt+" in response"
```

```
* assert lessThan(300, response.responseCode) : "Expected response code to be␣
↪less than 300
   but was "+response.responseCode+" in response"
```

- **assertResponseCodeIn**(*expectedValues*)
  - Asserts that the status code of the response is one of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that the response code is either 200 or 201
response.assertResponseCodeIn("200,201")
```

  - Equivalent to each of these:

```
* assertResponseCodeIn("200,201", response)
```

```
// expected status code and operator can be passed as parameters from test␣
↪script.
assertResponseCode("200,201", "in", response)
```

```
* assert isIn("200,201", response.responseCode): "Expected one of the values␣
↪in [200, 201]
   for response code but encountered \""+response.responseCode+"\" in␣
↪response"
```

- **assertResponseCodeNotIn**(*expectedValues*)
  - Asserts that the status code of the response is none of the provided *expectedValues* where each value is separated by a comma.
  - Example:

```
// Asserts that the response code is neither 200 nor 201
response.assertResponseCodeNotIn("200,201")
```

  - Equivalent to each of these:

```
* assertResponseCodeNotIn("200,201", response)
```

```
// expected status code and operator can be passed as parameters from test␣
↪script.
assertResponseCode("200,201", "notIn", response)
```

```
* assert isNotIn("200,201", response.responseCode): "Expected none of the␣
↪values in [200, 201]
   for response code but encountered \""+response.responseCode+"\" with␣
↪operator 'notIn' in response"
```

9.8.5 Short-circuiting

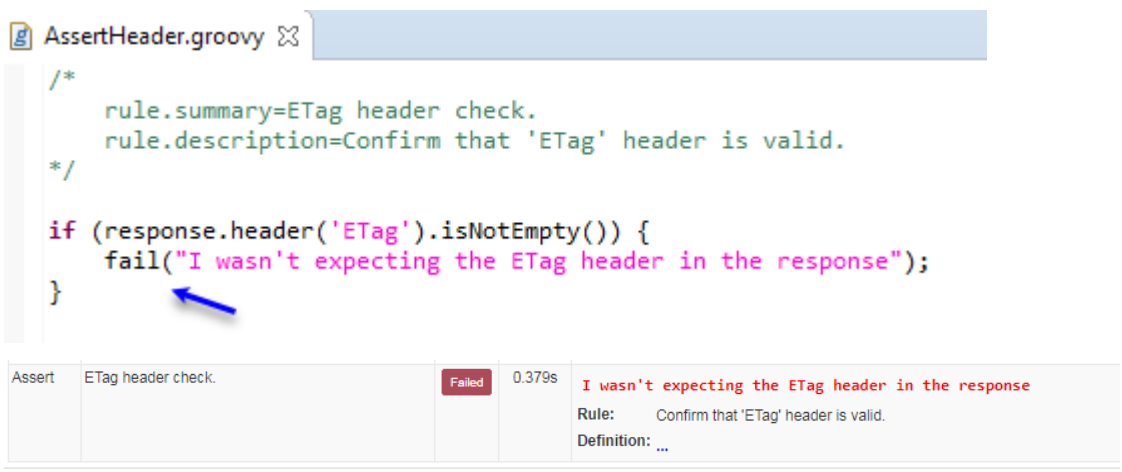
Rather than relying on the Rule API assertions to raise errors and skips with pre-defined messages, you can cause the rule execution to stop at any point based on certain conditions in your custom rule definitions. The message you construct will be displayed to end-user on the UI.

9.8.5.1 Failing

You can cause an assertion to fail with an arbitrary failure message using the following call:

- **fail**(*arbitrary message*)

Rule execution will stop at the point where this call is placed. The value provided in *arbitrary message* will be shown on the UI as the assertion failure message.



9.8.5.2 Warning

You can cause an assertion to stop with an arbitrary warning message using the following call:

- **warn**(*arbitrary message*)

Rule execution will stop at the point where this call is placed. The value provided in *arbitrary message* will be shown on the UI as the assertion warning message. Note that the test script execution will be marked as `Passed with warning(s)` in this case (provided no failures took place).

AssertHeader.groovy

```
/*
    rule.summary=ETag header check.
    rule.description=Confirm that 'ETag' header is valid.
*/

if (response.header('ETag').isNotEmpty()) {
    warn("I wasn't expecting the ETag header in the response but I'm okay with a warning");
}
```

Test Script Execution - /FHIRSandbox/Initech/Patient/Server Assigned Id/Patient-server-id-json

Exec Id:	20180804045058836	**Description:**	Example test script to demonstrate rule authoring.
Start Time:	08/04/2018 04:50:58PM	**Test Setup:**	FHIRSandbox-Initech-Patient--All
End Time:	08/04/2018 04:50:59PM	**Executed By:**	Peter Gibbons
Status:	Passed W	**Organization:**	Initech
Duration:	0.804s	**Origin:**	Touchstone
Version:	6	**Destination:**	AEGIS.net, Inc. - QA WildFHIR FHIR-3-3-0 C http://qafhir4.dev.aegis.net:8
Specification:	FHIR 3.3.0 - R4 Ballot 1	**Test Script:**	/FHIRSandbox/Initech/Patient/Server Assigned Id/Patient-server-id-json

1 tests	1 passes	0 failures	0 skipped	0 running	0 waiting	0 not started	100% success

Setup [show] **Duration**: 0.300s **Status**: Passed

Tests

Test Name	Description
Test: RegisterNewPatient	Create a new patient, no extensions where the client assigns the resource id using JSON. The expected response is 201 (format. An OperationOutcome or empty response is also allowed.

Action	Description	Status	Duration	Details
Operation	**create** - Patient **Origin:** Touchstone **Destination:** AEGIS.net, Inc. - QA WildFHIR FHIR-3-3-0 http://qafhir4.dev.aegis.net:8080/fhir3-3-0	201 Created	0.095s	...
Assert	ETag header check.	Warning	0.333s	I wasn't expecting the ETag header in the response but I'm okay with a warning **Rule:** Confirm that 'ETag' header is valid. **Definition:** ...

9.8.5.3 Skipping

You can cause an assertion to skip with an arbitrary message using the following call:

- **skip**(*arbitrary message*)

Rule execution will stop at the point where this call is placed. The value provided in *arbitrary message* will be shown on the UI as the assertion skipped message. Note that the test script execution will be marked as `Skipped` in this case (provided no failures took place).

AssertHeader.groovy

```
/*
    rule.summary=ETag header check.
    rule.description=Confirm that 'ETag' header is valid.
*/

if (response.header('ETag').isNotEmpty()) {
    skip("I want to skip the rest of the test");
}
```

Test Script Execution - /FHIRSandbox/Initech/Patient/Server Assigned Id/Patient-se

Exec Id:	20180804045615527	**Description:**	Example test script to demonstrate rule a
Start Time:	08/04/2018 04:56:15PM	**Test Setup:**	FHIRSandbox-Initech-Patient--All
End Time:	08/04/2018 04:56:16PM	**Executed By:**	Peter Gibbons
Status:	Skipped	**Organization:**	Initech
Duration:	0.967s	**Origin:**	Touchstone
Version:	6	**Destination:**	AEGIS.net, Inc. - QA WildFHIR FHIR-3-3-
Specification:	FHIR 3.3.0 - R4 Ballot 1	**Test Script:**	/FHIRSandbox/Initech/Patient/Server As

1 tests	0 passes	0 failures	1 skipped	0 running	0 waiting

Setup [show] **Duration**: 0.417s **Status**: Passed

Tests

Test Name	Description
Test: RegisterNewPatient	Create a new patient, no extensions where the client assigns the resource id using JSON format. An OperationOutcome or empty response is also allowed.

Action	Description	Status	Duration	Details
Operation	**create** - Patient **Origin:** Touchstone **Destination:** AEGIS.net, Inc. - QA WildFHIR FHIR-3-3-0 http://qafhir4.dev.aegis.net:8080/fhir3-3-0	201 Created	0.093s	...
Assert	ETag header check.	Skipped	0.362s	I want to skip the rest of the test **Rule:** Confirm that 'ETag' header is valid. **Definition:** ...

9.8.6 Multiple Assertions

This section covers how assertions can be performed on multiple requests and responses and reporting all of the assertion failures and warnings.

9.8.6.1 Short-circuiting when a request or response assertion fails

This is the behavior that has been covered extensively in Rule API.

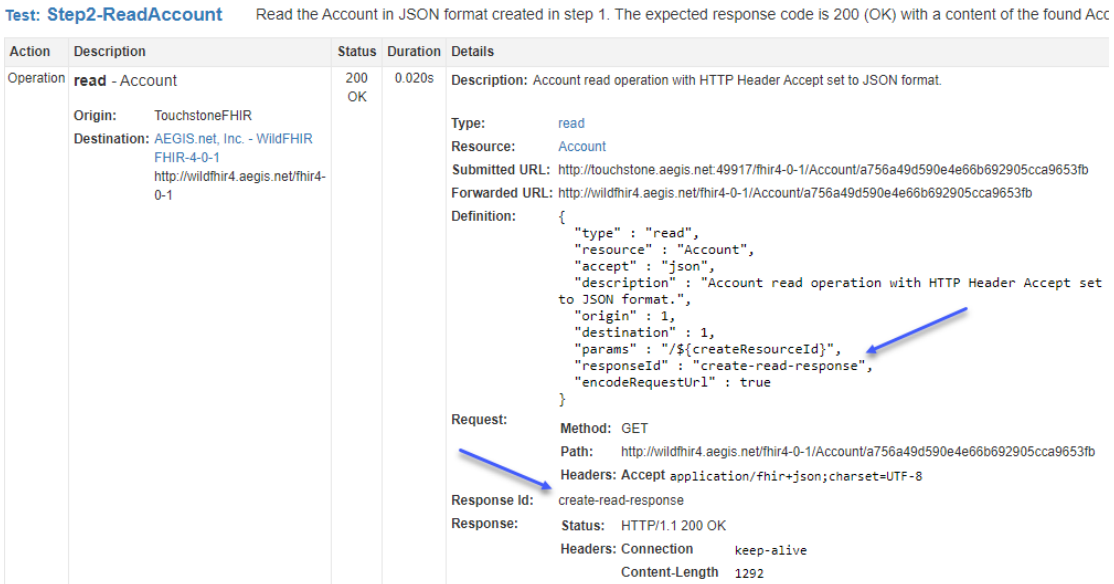
To grab the last response in TestScript Execution and perform an assertion on the response code, for example, we can use the following construct:

```
response.assertResponseCodeEquals(200)
```

To perform assertion on a response in TestScript Execution other than last response (e.g. one that was received upstream in another test), we could first grab the response from responses as such:

```
def createResponse = responses.get('create-read-response');
```

The responseId '**create-read-response**' in the example above would have been specified by the TestScript author in the TestScript **operation** definition and would be shown on the TestScript Execution screen:



A series of assertions could then be performed using the Rule API:

```
createResponse.assertResourceEquals("Bundle")
assert createResponse.header("Content-Type").contains("json") || createResponse.header(
↪"Content-Type").contains("xml"): "The actual value \""+response.header('Content-Type')?
↪.value +"\" did not contain the expected value \"json\" or \"xml\" for 'Content-Type'␣
↪header in response."
createResponse.assertResponseCodeEquals(200)
createResponse.assertBodyNotEmpty()
createResponse.assertValidWithProfile("http://hl7.org/fhir/StructureDefinition/Bundle")
```

The rule execution will fail on the first assertion failure. This is good in that we wouldn't want **assertValidWithProfile** to be performed if **assertBodyNotEmpty** failed, for example.

If we wanted to perform assertions on multiple requests and responses and wanted the assertion failures to be reported on all of the requests and responses, then the above approach would not work as rule-execution would stop on the first request or response assertion failure.

9.8.6.2 Continuing rule-execution when a request or response assertion fails

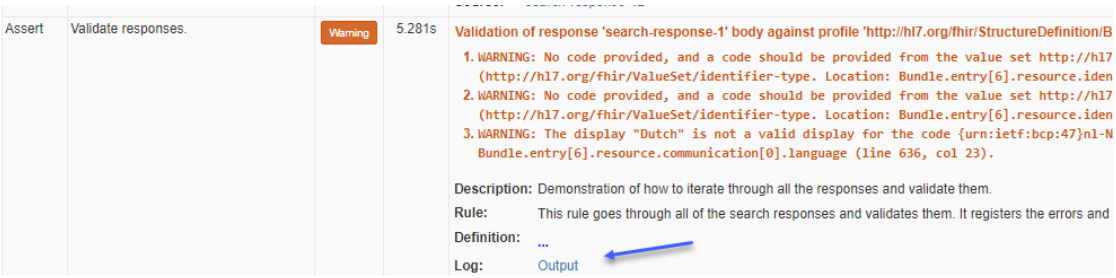
If multiple assertions are performed in a rule (e.g. on all search responses) and one of the assertions fail, then subsequent assertions are not performed. To perform all the assertions, the rule author must catch the assertion failure and register the error with the rules-engine before proceeding with the other assertions. Touchstone will fail the overall rule assertion if one or more assertions fail and will consolidate all the assertion failure messages into one message.

Here's an example that demonstrates how to perform assertions on multple responses without short-circuiting on any given response assertion failure:

```
responses.each() { responseEntry ->
   def responseId = responseEntry.key
   if (responseId.startsWith("search-response")) { // The prefix can be passed as a rule␣
↪parameter from the test script
      try {
         def response = responseEntry.value

         logger.info("Validating Resource in "+responseId)
         response.assertResourceEquals("Bundle")

         logger.info("Validating Content-Type in "+responseId)
         assert response.header("Content-Type").contains("json") || response.header(
↪"Content-Type").contains("xml"): "The actual value \""+response.header('Content-Type')?
↪.value +"\" did not contain the expected value \"json\" or \"xml\" for 'Content-Type'␣
↪header in response."

         logger.info("Validating Response Code in "+responseId)
         response.assertResponseCodeEquals(200)

         logger.info("Validating Body in in "+responseId)
         response.assertBodyNotEmpty()

         logger.info("Validating Resource in "+responseId+"\n")
         response.assertValidWithProfile("http://hl7.org/fhir/StructureDefinition/Bundle
↪")
      } catch (Throwable e) {
         errorsAndWarnings.registerAndContinue(e);
      }
   }
}
```

Notice how we're iterating through the responses and registering any assertion errors with the Rules Engine using **errorsAndWarnings.registerAndContinue(e);**, and then proceeding with assertions on the next response.

The log messages will be accessible on TestScript Execution screen within the assertion Log **Output** link:



Note: When writing rules that act on requests and responses other than the last one in the TestScript Execution, it's best to set sourceId to 'none' in the assertion rule definition. The sourceId can be left out if the rule were operating on the last request or response.

```
<assert>
    <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-assert-rule">
        <extension url="ruleId">
            <valueId value="rule-validateAllResponses"/>
        </extension>
    </extension>
    <description value="This rule iterates through all the search responses and validates them."/>
    <sourceId value="none"/>
    <warningOnly value="false"/>
</assert>
```

9.8.7 Rule Outputs

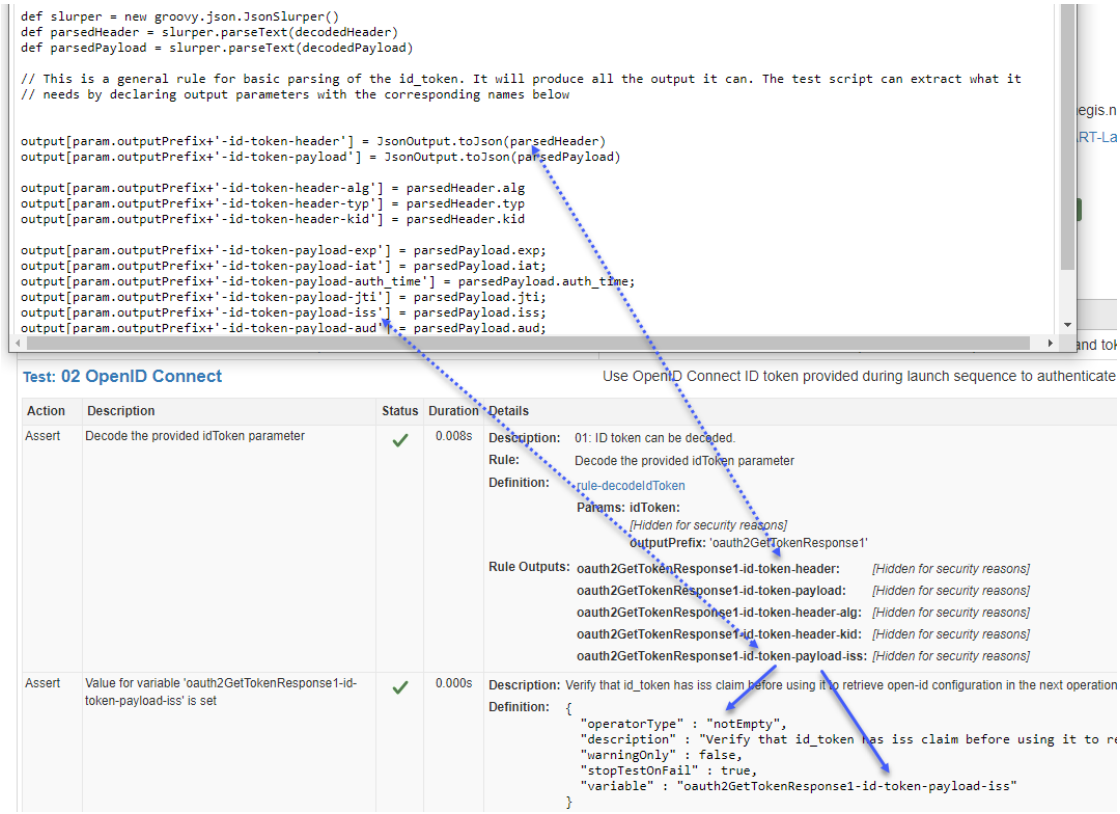
Groovy rules can produce documents or text, and place them in the running execution for somesequent assertions and operations to act upon them.

Here is an example of a testscript rule that produces rule outputs:

```
<action>
    <assert>
        <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-assert-rule">
            <extension url="ruleId">
                <valueId value="rule-decodeIdToken" />
            </extension>
            <extension url="param">
                <extension url="name">
                    <valueString value="idToken" />
                </extension>
                <extension url="value">
                    <valueString value="${oauth2GetTokenResponse1IdToken}" />
                </extension>
            </extension>
            <extension url="param">
                <extension url="name">
                    <valueString value="outputPrefix" />
                </extension>
                <extension url="value">
                    <valueString value="oauth2GetTokenResponse1" />
                </extension>
            </extension>
            <extension url="output">
                <extension url="name">
                    <valueString value="oauth2GetTokenResponse1-id-token-payload" />
                </extension>
                <extension url="type">
                    <valueString value="document" />
                </extension>
                <extension url="contentType">
                    <valueString value="json" />
                </extension>
            </extension>
            <extension url="output">
                <extension url="name">
                    <valueString value="oauth2GetTokenResponse1-id-token-header" />
                </extension>
                <extension url="type">
                    <valueString value="document" />
                </extension>
                <extension url="contentType">
                    <valueString value="json" />
                </extension>
            </extension>
            <extension url="output">
                <extension url="name">
                    <valueString value="oauth2GetTokenResponse1-id-token-payload-iss" />
                </extension>
            </extension>
```

In the example above, the rule-ouput **oauth2GetTokenResponse1-id-token-header** is defined to produce a document of type JSON while the rule **oauth2GetTokenResponse1-id-token-payload-iss** produces the default text/string type.

Within the Groovy rule (below), a JSON document is constructed and placed in the **output** binding under the **oauth2GetTokenResponse1-id-token-header** key (or **Rule Output Id**). This id can then be used as the value of sourceId in downstream operations and assertions. The output **oauth2GetTokenResponse1-id-token-payload-iss**, on the other hand, is populated as a text/string value and can be acted upon using string operators like **notEmpty** and **equals**.



9.8.7.1 Assertions on one Rule Output

As mentioned in the earlier section, rule outputs can become the sourceId or variable of common testscript assertions. Alternatively, a rule output can be grabbed by its rule output id:

```
def idTokenHeaderDoc = ruleOutputs.get('oauth2GetTokenResponse1-id-token-header')

def idTokenHeaderIssValue = ruleOutputs.get('oauth2GetTokenResponse1-id-token-payload-iss
↪')
```

A series of assertions could then be performed using the Rule API:

```
idTokenHeaderDoc.assertBodyNotEmpty()
idTokenHeaderDoc.assertJsonPathContains("iss", "theValue")

assert !idTokenHeaderIssValue.is(null): "Expected oauth2GetTokenResponse1-id-token-
↪payload-iss to be set"
```

If we wanted to perform assertions on multiple rule outputs and wanted the assertion failures to be reported on all of the rule outputs, then the above approach would not work as rule-execution would stop on the first rule outputs assertion failure.

9.8.7.2 Assertions on multiple Rule Outputs

If multiple assertions are performed in a rule (e.g. on all rule outputs) and one of the assertions fail, then subsequent assertions are not performed. To perform all the assertions, the rule author must catch the assertion failure and register the error with the rules-engine before proceeding with the other assertions. Touchstone will fail the overall rule assertion if one or more assertions fail and will consolidate all the assertion failure messages into one message.

Here's an example that demonstrates how to perform assertions on multple rule outputs without short-circuiting on any given rule output assertion failure:

```
ruleOutputs.each() { ruleOutputEntry ->
   def ruleOutputId = ruleOutputEntry.key
   if (ruleOutputId.startsWith("oauth2GetToken")) { // The prefix can be passed as a␣
↪rule parameter from the test script
      try {
         def idTokenHeaderDoc = ruleOutputEntry.value

         logger.info("Validating "+ruleOutputId)
         idTokenHeaderDoc.assertBodyNotEmpty()
         idTokenHeaderDoc.assertJsonPathContains("iss", "theValue")
      } catch (Throwable e) {
         errorsAndWarnings.registerAndContinue(e);
      }
   }
}
```

Notice how we're iterating through the ruleOutputs and registering any assertion errors with the Rules Engine using **errorsAndWarnings.registerAndContinue(e);**, and then proceeding with assertions on the next rule output.

9.8.7.3 Rule Output when outputting a Fixture

First, the Rule output when outputting a fixture and not a variable needs to set up the 'output' extension like so:

```
<action>
        <assert>
                <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-assert-rule">
                        <extension url="ruleId">
                                <valueId value="{rule Id}"/>
                        </extension>

                        <extension url="output">
                                <extension url="name">
                                        <valueString value="{name of the output fixture -
↪ Example - ruleOutputFile}"/>
                                </extension>
                                <extension url="type">
                                        <valueString value="document"/>
                                </extension>
                                <extension url="contentType">
```

(continues on next page)

(continued from previous page)

```
                                        <valueString value="{format of the fixture, i.e.␣
↪json or xml}"/>
                                </extension>
                        </extension>

                </extension>
                <description value="Full Description here" />
                <sourceId value="none"/>
                <warningOnly value="false" />
        </assert>
</action>
```

The output fixture can then be accessed in a assert by using assert.sourceId like so:

```
<action>
        <assert>
                <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-assert-stopTestOnFail">
                        <valueBoolean value="false" />
                </extension>
                <description value="Full Description here"/>
                <expression value="Bundle.type = 'batch'"/>
                <sourceId value="ruleOutputFile" />
                <warningOnly value="false"/>
        </assert>
</action>
```

Any Rule Output Fixture, once populated, can be referenced by the 'name' extension value like any other static/dynamic fixture id.

And can be used in a Operation like normal:

```
<action>
        <operation>
                <type>
                        <system value="http://touchstone.com/fhir/testscript-operation-
↪codes-extended"/>
                        <code value="process-message"/>
                </type>
                <description value="Full Description here"/>
                <accept value="json"/>
                <contentType value="json"/>
                <encodeRequestUrl value="true"/>
                <params value="/$validate"/>
                <sourceId value="ruleOutputFile"/>
        </operation>
</action>
```

Note: you may find this rule helpful.

9.8.8 XSLT and Schematron

Unless you plan on executing test scripts against a test system that only supports XML, it is highly recommended to write rules in Groovy as XSLT and Schematron rules can only be evaluated against requests and responses whose content is in XML while Groovy supports JSON as well.

9.8.8.1 Declarations

Below is the syntax for defining summary, description, and parameters in XSLT and Schematron. Please refer to Parameters for more information on how to supply parameters in test script.

Notice that the syntax is the same as that in Groovy. Only the comment characters are different. Instead of using `/*` and `*/`, we're using the XML comment characters `<!--` and `-->`

```
<!--
   rule.summary=Response '${param.header}' header cannot be empty
   rule.description=Validates the '${param.header}' header in the response
   rule.param.header.required=true
-->
```

9.8.8.2 Header assertions

Suppose you wanted to check the 'Content-Type' header in the response and compare it to an expected value passed as a parameter from the test script.

Below is the same logic coded in three formats for comparison:

9.8.8.2.1 Groovy

```
/*
   rule.summary=Response Content-Type header must be ${expectedContentType}.
   rule.description=Validates the content type of the response
   rule.param.expectedContentType.required=true
*/
assert !param.expectedContentType.is(null): "The parameter 'expectedContentType' was not␣
↪supplied"
assert !responseHeaders.header['Content-Type'].isEmpty(): "Could not find 'Content-Type'␣
↪header"
assert  responseHeaders.header['Content-Type'].containsIgnoreCase(param.
↪expectedContentType):
   "Expected Content-Type '"+ param.expectedContentType+"' did not match actual Content-
↪Type '"
   + responseHeaders.header['Content-Type']+"'"
```

9.8.8.2.2 XSLT

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsl:stylesheet xmlns:xhtml="http://www.w3.org/1999/xhtml" xmlns:xsl="http://www.w3.org/
↪1999/XSL/Transform"
   xmlns:saxon="http://saxon.sf.net/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:f="http://hl7.org/fhir" xmlns:t=
↪"http://touchstone.aegis.net/"
   version="2.0">
<!--
 rule.summary=Response Content-Type header must be ${expectedContentType}.
 rule.description=Validates the content type of the response
   rule.param.expectedContentType.required=true
-->
   <xsl:output method="xml" omit-xml-declaration="no" standalone="yes" indent="yes" />
   <xsl:param name="param" />
   <xsl:param name="headers" />
   <xsl:template match="/">
      <t:validationresult>
         <xsl:choose>
            <xsl:when test="$param">
               <xsl:choose>
                  <xsl:when test="$headers"> <!-- You can also use 'responseHeaders' or
↪'requestHeaders'.
                                    The right one will be supplied in place of headers␣
↪depending on direction -->
                     <xsl:choose>
                        <xsl:when test="$param/expectedContentType and $param/
↪expectedContentType/text()">
                           <xsl:choose>
                              <xsl:when test="$headers/Content-Type">
                                 <xsl:choose>
                                    <xsl:when test="contains(lower-case($headers/Content-
↪Type/text()),
                                       lower-case($param/expectedContentType))" />
                                    <xsl:otherwise>
                                       <t:error>Expected Content-Type '<xsl:value-of
                                          select="$param/expectedContentType" />'
                                          did not match actual Content-Type '<xsl:value-
↪of
                                          select="$headers/Content-Type" />'.</t:error>
                                    </xsl:otherwise>
                                 </xsl:choose>
                              </xsl:when>
                              <xsl:otherwise>
                                 <t:error>Could not find 'Content-Type' header.</t:error>
                              </xsl:otherwise>
                           </xsl:choose>
                        </xsl:when>
                        <xsl:otherwise>
                           <t:error>The parameter 'expectedContentType' was not supplied.
↪</t:error>
```

(continues on next page)

(continued from previous page)

```
                      </xsl:otherwise>
                   </xsl:choose>
                </xsl:when>
                <xsl:otherwise>
                   <t:error>The headers were not supplied.</t:error>
                </xsl:otherwise>
             </xsl:choose>
          </xsl:when>
          <xsl:otherwise>
             <t:error>The parameters were not supplied.</t:error>
          </xsl:otherwise>
       </xsl:choose>
    </t:validationresult>
  </xsl:template>
</xsl:stylesheet>
```

9.8.8.2.3 Schematron

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <sch:ns prefix="f" uri="http://hl7.org/fhir"/>
  <sch:ns prefix="h" uri="http://www.w3.org/1999/xhtml"/>

<!--
 rule.summary=Response Content-Type header must be ${expectedContentType}.
 rule.description=Validates the content type of the response
   rule.param.expectedContentType.required=true
-->
    <xsl:param name="param" />
    <xsl:param name="headers" />   <!-- You can also use 'responseHeaders' or
↪'requestHeaders'.
                      The right one will be supplied in place of headers depending on␣
↪direction -->
    <sch:pattern>
    <sch:title>RuleContentType</sch:title>
    <sch:rule context="/">
      <sch:assert test="$param">The parameters were not supplied.</sch:assert>
      <sch:assert test="$headers">The headers were not supplied.</sch:assert>
      <sch:assert test="$param/expectedContentType">The parameter 'expectedContentType'␣
↪was not supplied.
                                                        </sch:assert>
      <sch:assert test="$param/expectedContentType/text()">The parameter
↪'expectedContentType' was not
                                          supplied.</sch:assert>
      <sch:assert test="$headers/Content-Type">Could not find 'Content-Type' header.</
↪sch:assert>
      <sch:assert test="contains(lower-case($headers/Content-Type/text()), lower-case(
↪$param/expectedContentType))">
             Expected Content-Type '<xsl:value-of select="$param/expectedContentType"/>'␣
↪did not match actual
```

(continues on next page)

(continued from previous page)

```
            Content-Type '<xsl:value-of select="$headers/Content-Type"/>'.</sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

9.8.8.3 Payload assertions

Suppose you wanted to determine the FHIR resource within the response and compare it to an expected value passed as a parameter from the test script.

Below is the same logic coded in three formats for comparison:

9.8.8.3.1 Groovy

```
/*
 rule.summary=Response resource must be ${expectedResource}.
 rule.description=Validates the type of the resource in the response.
 rule.param.expectedResource.required=true
 */
assert response.resource==${param.expectedResource}: "Expected resource '"
   +param.expectedResource +"' did not match actual resource '"+response.resource+"'"
```

9.8.8.3.2 XSLT

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsl:stylesheet xmlns:xhtml="http://www.w3.org/1999/xhtml" xmlns:xsl="http://www.w3.org/
↪1999/XSL/Transform"
      xmlns:saxon="http://saxon.sf.net/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:f="http://hl7.org/fhir"
      xmlns:t="http://touchstone.aegis.net/" version="2.0">

<!--
 rule.summary=Response resource must be ${expectedResource}.
 rule.description=Validates the type of the resource in the response.
 rule.param.expectedResource.required=true
-->
   <xsl:output method="xml" omit-xml-declaration="no" standalone="yes" indent="yes" />
   <xsl:param name="param" />
      <xsl:template match="/">
      <t:validationresult>
         <xsl:choose>
            <xsl:when test="$param">
               <xsl:choose>
                  <xsl:when test="$param/expectedResource and $param/expectedResource/
↪text()">
                     <xsl:choose>
                        <xsl:when test="*[1]/name()=$param/expectedResource" />
                        <xsl:otherwise>
```

(continues on next page)

(continued from previous page)

```
                            <t:error>Expected resource '<xsl:value-of select="$param/
↪expectedResource"/>'
                                did not match actual resource '<xsl:value-of select="*[1]/
↪name()"/>'.</t:error>
                        </xsl:otherwise>
                     </xsl:choose>
                  </xsl:when>
                  <xsl:otherwise>
                     <t:error>The parameter 'expectedResource' was not supplied.</
↪t:error>
                  </xsl:otherwise>
               </xsl:choose>
            </xsl:when>
            <xsl:otherwise>
               <t:error>The parameters were not supplied.</t:error>
            </xsl:otherwise>
         </xsl:choose>
      </t:validationresult>
   </xsl:template>
</xsl:stylesheet>
```

9.8.8.3.3 Schematron

```
<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <sch:ns prefix="f" uri="http://hl7.org/fhir"/>
  <sch:ns prefix="h" uri="http://www.w3.org/1999/xhtml"/>

<!--
 rule.summary=Response resource must be ${expectedResource}.
 rule.description=Validates the type of the resource in the response.
 rule.param.expectedResource.required=true
-->

   <xsl:param name="param" />
    <sch:pattern>
    <sch:title>RuleResource</sch:title>
    <sch:rule context="/">
      <sch:assert test="$param">The parameters were not supplied.</sch:assert>
      <sch:assert test="$param/expectedResource">The parameter 'expectedBundleResource'
               was not supplied.</sch:assert>
      <sch:assert test="*[1]/name()=$param/expectedResource">Expected resource
         '<xsl:value-of select="$param/expectedResource"/>' did not match actual resource
         '<xsl:value-of select="*[1]/name()"/>'.</sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

9.9 OAuth2 Capabilities

Touchstone has the ability for a user to create and use a Test System that is OAuth2 enabled, and with that comes of few features that are important to take note of when it comes to Test Executions performed against a Test System that has an OAuth2 Authorization service connected to it.

9.9.1 Fixture ID's

With the addition of a more robust OAuth2 environment into Touchstone, there are a few fixtures, and values under those fixtures, that can be reached for test script authors.

1. One fixture is the *dest1SmartConfig* (*dest2SmartConfig*, *dest3SmartConfig*, etc.). The fixture allows the test script author to access different variables coming from the JSON document that is at the *.well-known/smart-configuration.json* endpoint. It acts the same as retrieving variables from the capabilities statement. For example, to use *dest1SmartConfig* to retrieve the Client ID, you would use the `sourceId` and `path` in the testscript as `sourceId` = *dest1SmartConfig*, `path` = *$.clientId*.

2. Test Systems have special Oauth2 values that can be be retrieved by test script authors. The fixtures have a naming convention of *dest1SystemConfig*, *dest2SystemConfig*, *origin1SystemConfig*, *origin2SystemConfig*, etc. These fixtures have the following attributes:

Fixture ID	Description
name	The Test System name
fullName	The Organization name + the Test System name
baseUrl	The Base URL of the Test System
supportsSmartOnFhir	Set to `true` if the Test System supports SMART on FHIR, `false` if it does not
oauth2GrantType	The Grant Type of the OAuth2 enables Test System, either Authorization Code or Client Credentials
clientId	The Client ID of the Test System that is registered with the OAuth2 server.
clientSecret	The Client Secret of the Test System that is registered with the OAuth2 server.
authEndpoint	The Authorization Endpoint for the Test System.
tokenEndpoint	The Token Endpoint for the Test System.
registerEndpoint	The Registration Endpoint for the Test System.
introspectEndpoint	The Introspection Endpoint for the Test System.
revocationEndpoint	The Revocation Endpoint for the Test System.
scopesSupported	The scopes that allowed for the server access to certain user scopes

3. An example of using these fixtures in a test script is below:

```
<variable>
    <name value="tokenEndpoint"/>
    <path value=".token_endpoint"/>
    <sourceId value="dest1SmartConfig"/>
</variable>
```

4. Another set of fixtures that can be used by the user in the test scripts are *oauth2AuthzRequest* and *oauth2AuthzRedirect*. These two fixtures are used to access parts the the last OAuth2 Authroization Request sent to the OAuth2 server and the last OAuth2 Authorization response returned from the server.

9.9.2 Base64Encoding

Touchstone is configured to use a **Base64Encoding** on the *operation.requestHeader.value* when it includes `Basic` + `A Single Space " "` ahead of the value and when the *operation.requestHeader.name* is equal to `Authorization` in your Testscript executions. This is done for security and is the explanation as to why an `Authorization` value in the header will be different from the one that was originally coded.

9.9.3 PKCE Functionality

1. Executing SMART testcases using PKCE needs Code Verifier (which is a randomly generated string meeting a certain requirement) and a Code Challenge (which is a string generated by hashing the Code Verifier and a Code Challenge Method).
2. The groovy rule PKCECodeExchange.groovy under FHIRCommon can be used for generating the Code Verifier, Code Challenge and the Code Challenge Method.
3. Steps to include PKCE information in a SMART Testcase,

   a. Define an extension to include the rule PKCECodeExchange.groovy

   Example:

```
<extension url="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript-rule">
        <extension url="ruleId">
                <valueId value="rule-fetchCodeExchange" />
        </extension>
        <extension url="path">
                <valueString value="/FHIRCommon/_reference/rule/
↪PKCECodeExchange.groovy" />
        </extension>
</extension>
```

   b. Create an assertion and place it before any operation that requests the authorization code

   Example:

```
<action>
        <assert>
                <extension url="http://touchstone.aegis.net/touchstone/fhir/
↪testing/StructureDefinition/testscript-assert-rule">
                        <extension url="ruleId">
                                <valueId value="rule-fetchCodeExchange" />
                        </extension>
                        <extension url="output">
                                <extension url="name">
                                        <valueString value="codeChallenge" /
↪>
                                </extension>
                        </extension>
                        <extension url="output">
                                <extension url="name">
                                        <valueString value=
↪"codeChallengeMethod" />
                                </extension>
```

(continues on next page)

(continued from previous page)

```
                        </extension>
                        <extension url="output">
                                <extension url="name">
                                        <valueString value="codeVerifier" />
                                </extension>
                        </extension>
                </extension>
                <extension url="http://touchstone.aegis.net/touchstone/fhir/
↪testing/StructureDefinition/testscript-assert-stopTestOnFail">
                <valueBoolean value="true" />
                </extension>
                <description value="Generates CodeChallenge and␣
↪CodeVerifier." />
                <warningOnly value="false" />
        </assert>
</action>
```

c. Include the Code Challenge and Code Challenge Method to the URL under the Authorization Request Operation

Example:

```
<action>
        <operation>
                <extension url="http://touchstone.aegis.net/touchstone/fhir/
↪testing/StructureDefinition/testscript-operation-oauth2AuthzRequestId">
                        <valueId value="oauth2AuthzRequest1" />
                </extension>
                <extension url="http://touchstone.aegis.net/touchstone/fhir/
↪testing/StructureDefinition/testscript-operation-oauth2AuthzRedirectId">
                        <valueId value="oauth2AuthzRedirect1" />
                </extension>
                <type>
                        <system value="http://touchstone.aegis.net/
↪touchstone/fhir/testing/CodeSystem/codesystem-testscript-operation-codes"␣
↪/>
                        <code value="oauth2-authorize" />
                </type>
                <description value="Redirect user to the authorize endpoint␣
↪for target test system specified in smart configuration" />
                <encodeRequestUrl value="true" />
                <url value="${authorizeEndpoint}?client_id=$
↪{dest1SystemConfig.clientId}&amp;scope=${oauth2RequiredScopes}&amp;aud=$
↪{dest1SystemConfig.baseUrl}&amp;code_challenge=${codeChallenge}&amp;code_
↪challenge_method=${codeChallengeMethod}" />
        </operation>
</action>
```

d. Add the Code Verifier to the Access Token Request. Note: If a Fixture is used for loading the token request parameters, update the fixture with this parameter

Example:

```
grant_type=authorization_code&code=${oauth2AuthzRedirect1AuthCode}&redirect_
↪uri=${oauth2AuthzRequest1RedirectUri}&code_verifier=${codeVerifier}
```

9.10 Bulk Data Capabilities

Touchstone has the ability for a user to create and use a Test System that supports the Bulk Data specification. Touchstone provides additional features that provide support for the evaluation and validation of the NDJSON returned files.

9.10.1 NDJSON File Evalutation and Validation

The Bulk Data specification defines a new FHIR extended operation $export which generates bulk data output files containing multiple FHIR resources using the NDJSON format for FHIR - application/fhir+ndjson.

Evaluation and validation of this NDJSON format for FHIR requires extended capabilities be implemented on the FHIR Test Engine and extended definitions within the FHIR TestScript resource. Touchstone has implemented these extended capabilities through the use of NDJSON Assertion Prefix syntax within the values of the following TestScript elements:

- TestScript.profile.reference
- TestScript.test.action.assert.expression
- TestScript.test.action.assert.path
- TestScript.test.action.assert.resource

9.10.2 NDJSON Assertion Prefix

The NDJSON Assertion-Prefix is specified within the curly braces and is composed of 3 parts (highlighted in color):

{ Evaluation-Operator | Filter-Index-Range | Filter-Path} Regular-Assert. All three are optional.

Note: The use of the NDJSON Assertion Prefix syntax within the *assert.resource* element prevents the containing testscript resource from validating on upload to Touchstone.

NDJSON assertions are applied in 4 stages:

1. Filter-Path evaluation. Example: .name[?(@.family=='Gracia')] i.e. all Patients whose family-name is 'Gracia'. Resource is included in evaluation if JSON-PATH evaluation results to a Truthy value (exists and is not false for our purposes). If no Filter-Path is specified then all resources filter through.
2. Filter-Index-Range. Example: 1 - 10 i.e. Include resources 1 through 10 in the assertion evaluation. If Filter-Index-Range is not specified then all resources filter through. If Filter-Path was specified then Filter-Index-Range operates on the resources that made it through Filter-Path evaluation and not the original resources.
3. Regular-Assert :- The Profile/Expression/Path/Resource evaluation on resources that made it past 1 and 2. Example: Patient (if assertion is resource).
4. Evaluation-Operator (any/all). If 'any' operator is used then overall assertion passes if single resource passes. Default is 'all'.

For examples of the NDJSON Assertion Prefix syntax, visit the Touchstone Testing Implementation Guide.

9.11 TestScript Extensions

The list of AEGIS Touchstone Testing Extensions is shown below. They can be found in our Touchstone IG.

- AEGIS Touchstone Testing TestScript Dynamic Fixture Extension
- AEGIS Touchstone Testing TestScript Rule Extension
- AEGIS Touchstone Testing TestScript Ruleset Extension
- AEGIS Touchstone Testing TestScript Assert Rule Extension
- AEGIS Touchstone Testing TestScript Assert Ruleset Extension
- AEGIS Touchstone Testing TestScript Assert Stop Test On Error Extension
- AEGIS Touchstone Testing TestScript Assert Variable Extension
- AEGIS Touchstone Testing TestScript Operation AuthorizeInNewTab Extension
- AEGIS Touchstone Testing TestScript Operation Oauth2AuthzRedirectId Extension
- AEGIS Touchstone Testing TestScript Operation Oauth2AuthzRequestId Extension
- AEGIS Touchstone Testing TestScript Operation pagesNext Extension
- AEGIS Touchstone Testing TestScript Operation SmartLaunchRequestId Extension
- AEGIS Touchstone Testing TestScript Test Manual Completion Extension
- AEGIS Touchstone Testing TestScript Variable ParamField Extension

9.12 FAQ

1. I've used a variable for the Accept header but Touchstone is rejecting my TestScript on upload. Touchstone only supports variables in these TestScript.operation elements: *encodeRequestUrl, params, requestHeader.value, and url*. If you need a dynamic header like Accept or ContentType you can define it manually using variables in the Operation.requestHeader element:

```
<requestHeader>
        <field value="Accept"/>
        <value value="application/fhir+${accept}"/>
</requestHeader>
```

2. I want to use the validateProfileId asserts but I don't know what IGs are supported. If your organization has a custom validator in Touchstone refer to Updating Validators to see how to view and add validation packages to your validator. Otherwise the list of IGs supported in the base Validators is avaliable here
3. My private authorization tokens are visible to unauthorized users. As authors of test scripts that involve authorization, please test access to testscript execution data and confirm that OAuth2-sensitive data (like access tokens) is hidden from unauthorized users. To hide OAuth2-related fixtures and rule parameters within operation and assertion executions on Test Script Execution screen from other users, reference the fixtures (sourceId/responseId in operation) and rule parameters with names that begin with "oauth2", "openId", "jwks", "idToken", or "accessToken" as appropriate. Touchstone will hide the data from everyone except the executor of the test execution when the prefixes (listed above) are used.
4. I am receiving a "Nothing to assert" error when I run test/assert extensions. Be sure to declare support for the touchstone IG using the following declaration:

```
<meta>
        <profile value="http://touchstone.aegis.net/touchstone/fhir/testing/
↪StructureDefinition/testscript"/>
</meta>
```

5. I am receiving "Stack Trace:" exception errors or unexpected false evaluations when using assert FHIRPath expressions containing literal or variables containing date, dateTime or time typed values. Please refer to the FHIRPath (Literals) specification regarding the use of literal values in FHIRPath expressions; i.e. Date, DateTime and Time.

   The FHIR TestScript defines support for FHIRPath expressions in the following elements: TestScript.variable.expression TestScript.setup.action.assert.compareToSourceExpression TestScript.setup.action.assert.expression.

   Examples of FHIRPath expressions with literal or variables containing date, dateTime or time values within TestScript expression elements:

```
<assert>
        <description value="Evaluate Patient.birthDate equivalent against literal date␣
↪value"/>
        <expression value="Patient.birthDate ~ @1995-06-11"/>
        <warningOnly value="false"/>
</assert>
```

```
<assert>
        <description value="Evaluate Patient.birthDate equivalent against variable 'd-
↪birthDate' date value"/>
        <expression value="Patient.birthDate ~ @${d-birthDate}"/>
        <warningOnly value="false"/>
</assert>
```

```
<assert>
        <description value="Evaluate Patient.birthDate extension dateTime equivalent␣
↪against variable 'dt-birthTime' dateTime value"/>
        <expression value="Patient.birthDate.extension.where(url = 'http://hl7.org/fhir/
↪StructureDefinition/patient-birthTime').value ~ @${dt-birthTime}"/>
        <warningOnly value="false"/>
</assert>
```

```
<assert>
        <description value="Evaluate Patient.birthDate extension dateTime convert to␣
↪Time only equivalent against variable 't-birthTime' time value (note use of 'T' in␣
↪prefix)"/>
        <expression value="Patient.birthDate.extension.where(url = 'http://hl7.org/fhir/
↪StructureDefinition/patient-birthTime').value.toTime() ~ @T${t-birthTime}"/>
        <warningOnly value="false"/>
</assert>
```

6. I am having issues with fixtures in my HL7-V2 TestScripts at runtime saying that the formats are incorrect. The HL7-V2 specification requires that an HL7-V2 message consists of one or more segments. Each segment is displayed on a different line of text. A carriage return character (r, which is 0D in hexadecimal) separates one segment from another. The Minimal Lower Layer Protocol (MLLP) defines the leading and trailing delimiters for an HL7 message. The MLLP Protocol sends and receives a vertical tab to start, and a carriage return and file separator to end. Make sure that the content format of your HL7-V2 fixtures is not being altered before upload

to Touchstone. In particular, if your TestScripts are being pulled from a repository before you are uploading to Touchstone, make sure that the content format is not being altered.

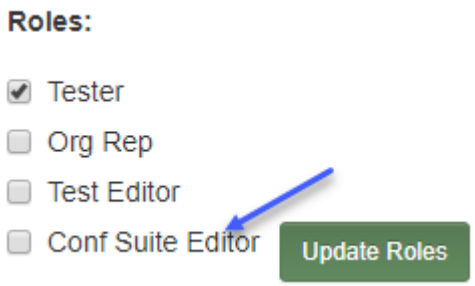
CHAPTER

TEN

CONFORMANCE SUITE AUTHORING

Organizations (with the appropriate subscription level) can now create their own Conformance Suites with custom test groups and test scripts.

The user must be assigned the **Conf Suite Editor** role before being able to create and edit Conformance Suites. Only users with this role can assign this role to other users within the organization:



10.1 Server Suites

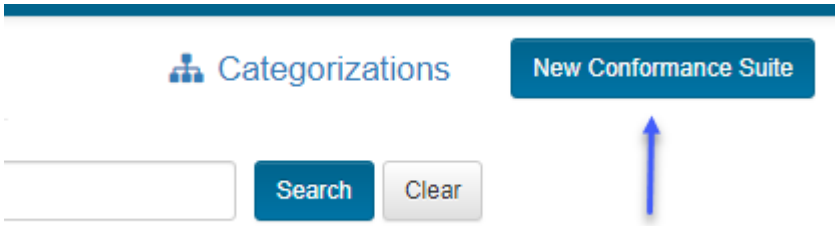
Note: This section assumes that you are already familiar with Conformance Testing. Please review the Conformance Testing documentation for additional information.

Test systems can be **Server** or **Client** test systems in Touchstone. This attribute is selected by the user on the Test System screen.

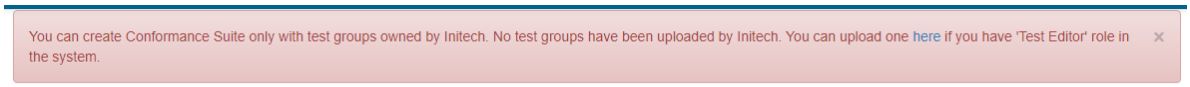
The following steps describe how to create Conformance Suites that are geared towards testing **Server** test systems:

1. Go to Conformance Suites page.

2. Click on **New Conformance Suite** button:



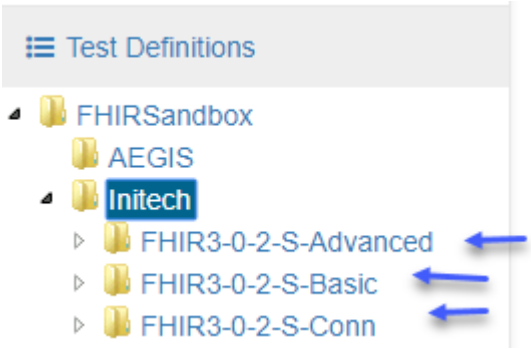
3. You will encounter the following error if you haven't uploaded test scripts to Touchstone yet:



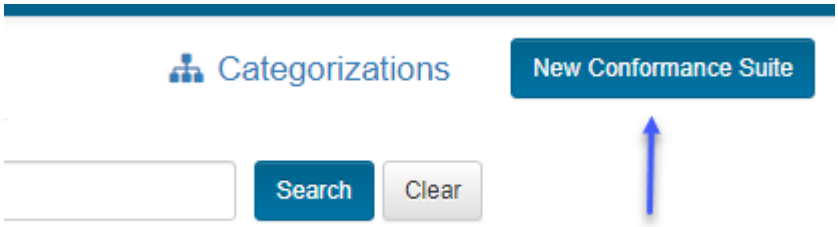
Note: Conformance Suites can be created only with test groups that your organization owns. Otherwise, the test group owner could inadvertently impact the Conformance results of many users in your program when a test group is modified.

To learn how to upload test groups to Touchstone, you can refer to the TestScript Authoring guide.

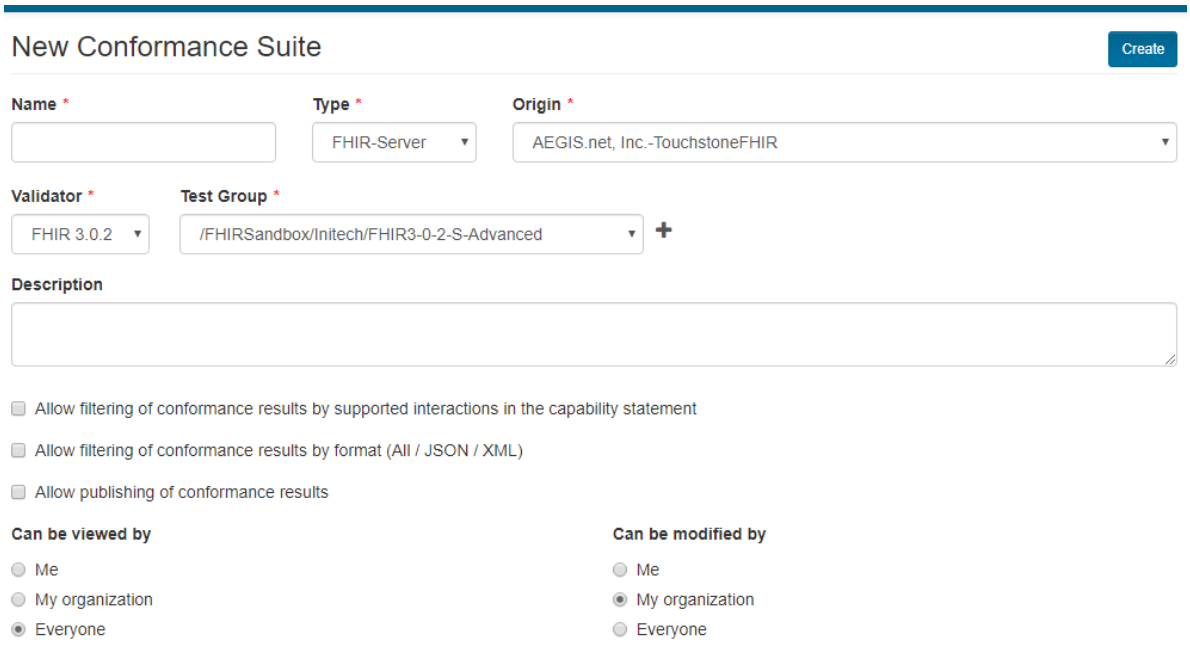
The user then uploads a few test groups:



4. Click on **New Conformance Suite** button again on the Conformance Suites page:



5. Below is the New Conformance Suite page:

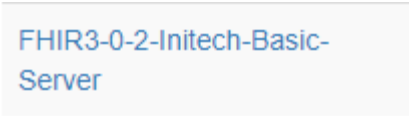


We will describe the meaning of each field as we populate them in this screen.

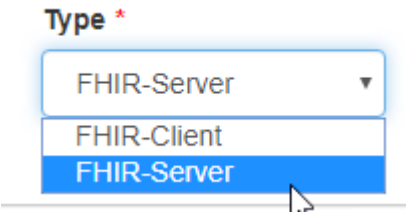
6. Enter the Name of the suite:

Note: Touchstone will prepend the specification name/version to the suite name. This is done to maintain uniformity and consistent naming convention across suites so it's easier for users to identify the specification that a suite is testing for. The type of suite (**Server** vs **Client**) is also appended by Touchstone to the name for the same reason.

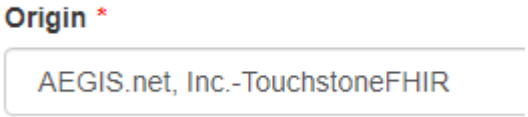
The final name of the suite in this instance would be the following:



7. Select the Type of the suite. For a suite that's geared towards testing **Server** test systems, the type would be **Server**:



8. Select the Anchor System of the suite. For a suite that's geared towards testing **Server** test systems (the SUT of the suite), the Anchor System would be a **Client/Origin** test system:



Conversely, if the suite is geared towards testing **Client** test systems (the SUT of the suite), the Anchor System would be a **Server/Destination** test system.

Anchor systems are needed to achieve comparable Conformance results among SUTs. If a suite uses different Anchor Systems, then the results would not be meaningful.

Anchor systems must be accessible by the organization that's creating the suite.

Note: It is highly recommended to use reliable anchor systems for your suites. This is to avoid unintended consequences on the program's conformance testing if the anchor system is deleted or its behavior is modified by the owning organization. Touchstone does not require the suite owner to also own the anchor system because it's not easy to make such systems available. It's rather common for organizations to rely on third parties for such systems. In this case, we're using **TouchstoneFHIR** as the **Client anchor system** from which requests will be sent to the **Server** test systems being tested in the suite.

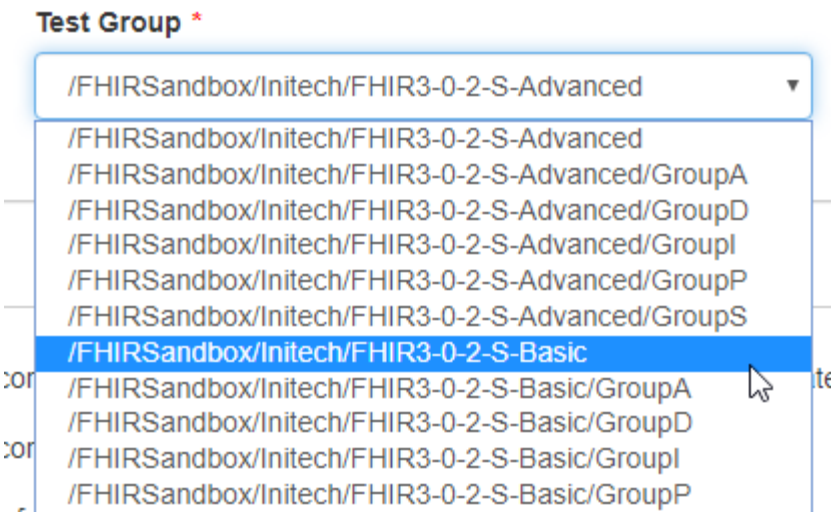
---

9. Select the Valdiator of the suite:

Validator is the AEGIS Validator to use for validating request and response payloads during test execution.

The list of Validators offered in the dropdown will be limited to the Validators used by the Test Groups owned by the suite author's organization.

9. Select the Test Group(s) of the suite:



Test Groups will contain the test scripts that the user will execute in conformance testing. The Test Group dropdown entries will be driven by the Validator selection. If you select **FHIR 3.0.2** as the Validator, for example, then the Test Group dropdown will be populated with FHIR 3.0.2 test groups that you have access to.

Additional test groups can be selected by clicking the **+** sign:

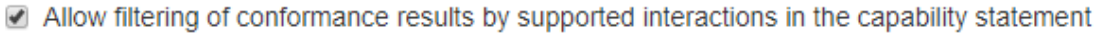
We'll cover multiple test group selection in the Categorization section of this guide.

Note: In order to be able to properly select test groups that were uploaded prior to Touchstone 5.0.0 release, you must re-upload them to Touchstone.

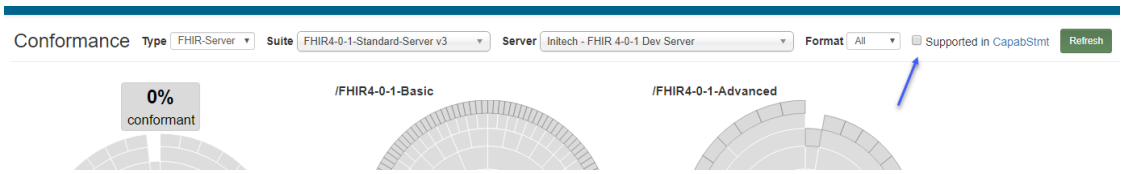
10. Enter a Description for the suite:

The system will generate a default description if none is entered.

11. Choose whether or not Supported checkbox is offered to users:

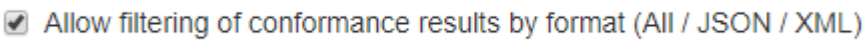


If checked, users will be offered the **Supported by CapabStmt** checkbox on conformance results screens. They would be able to filter results by supported interactions in the capability statement in addition to the default (All):

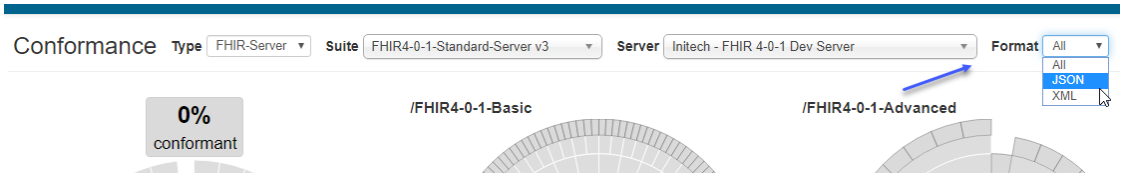


Please refer to *this section* to see how this checkbox selection affects conformance results.

12. Choose whether or not Formats dropdown is offered to users:

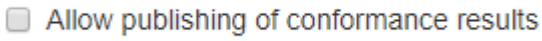


If checked, users will be offered the **Format** dropdown on conformance results screens. They would be able to filter results by individual formats (JSON / XML) in addition to the default (All):

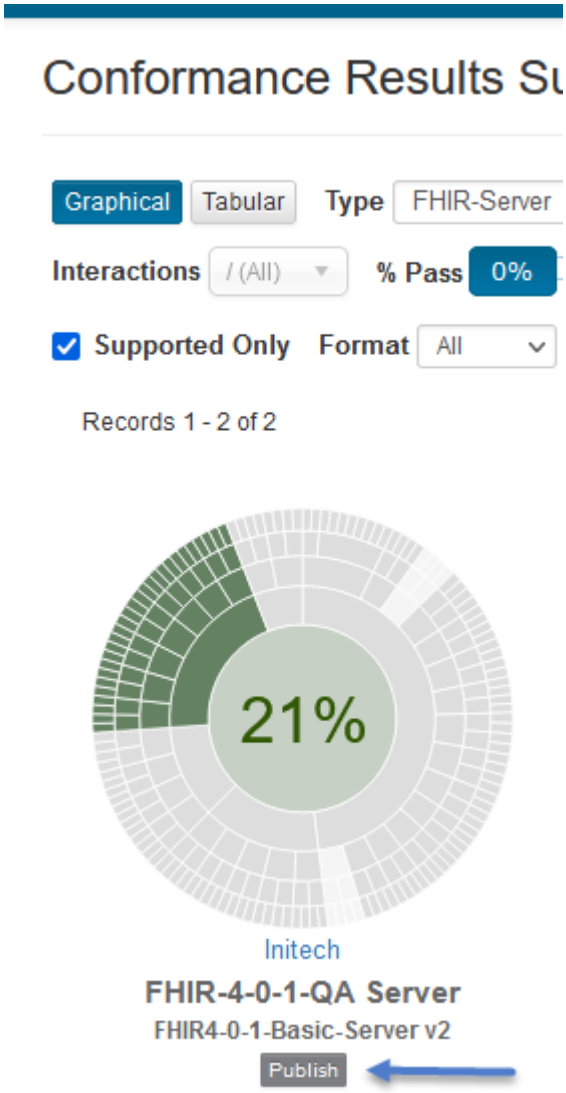


Please refer to *this section* to see how this checkbox selection affects conformance results.

13. Choose whether or not conformance results can be Published by users:



If checked, users will be offered the option of publishing conformance results. Those results along with the associated test executions would become publicly accessible if published:



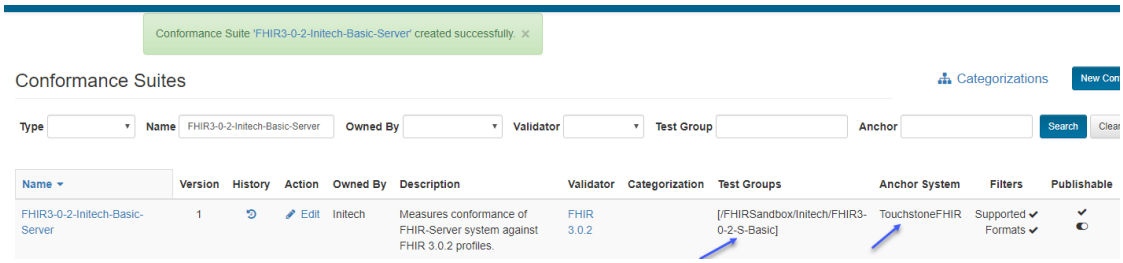
Please refer to Publishing to learn more about conformance results publishing.

14. Select the access attributes of the suite:

This works the same way as access attributes for test systems, test definitions, etc.

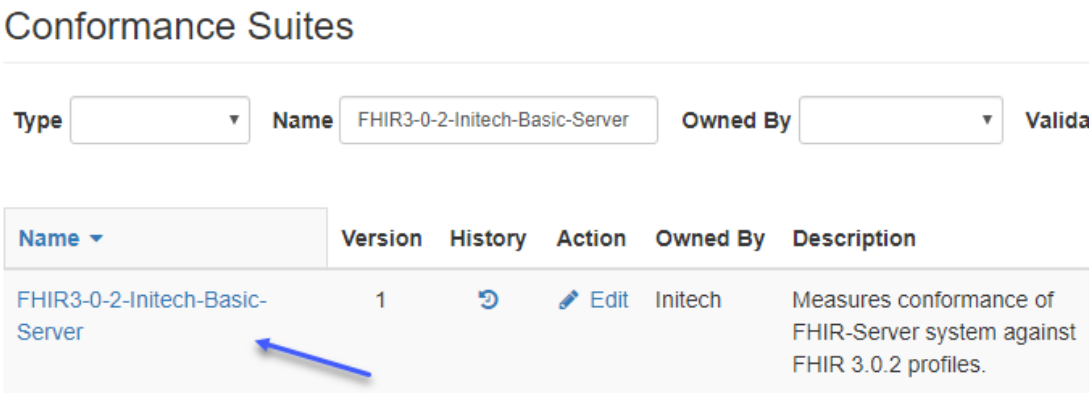
If you'd like only your organization to view and execute tests within the suite, then select `My Organization` for **Can be viewed by**.

15. Click on **Create**. Conformance Suite gets created:

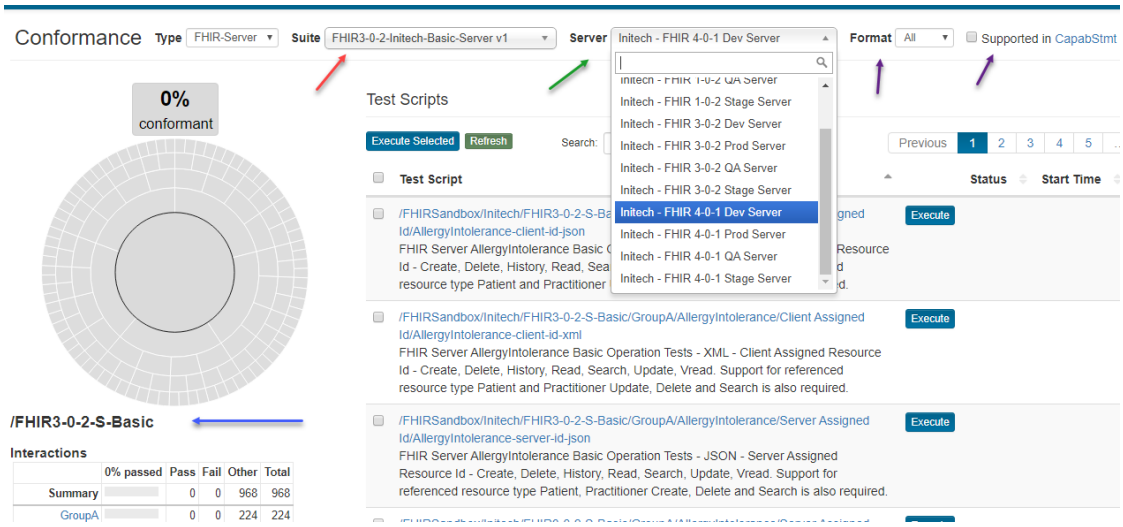


We selected **/FHIRSandbox/Initech/FHIR3-0-2-S-Basic** as the sole test group and **TouchstoneFHIR** as the anchor system for this suite.

16. Click on the name of the suite:



17. You'll be taken to the Current Conformance page where you can test the suite:



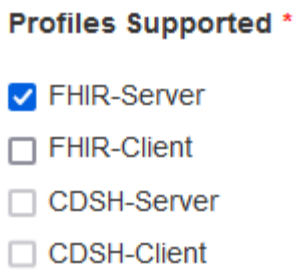
- The selected Conformance Suite (red arrow) is the one we clicked on in the list of suites in prior step.
- The suite is made up of only one test group (blue arrow) as that was the only test group we selected during creation.
- The test system dropdown (green arrow) lists the **Server** test systems that you have access to. Only **Server** test systems are populated because this is a **Server** Conformance Suite.
- The anchor system chosen for the suite (TouchstoneFHIR) is not listed in the test systems dropdown. The anchor system is the **Client** test system that will submit requests to the chosen **Server** test system in a **Server** conformance suite. As users select different **Server** test systems to run this Conformance Suite against, the same anchor system (**TouchstoneFHIR** in this case) will be used to submit requests to these **Server** test systems. This allows for uniform comparison of **Server**-system conformance results. **TouchstoneFHIR** is used to submit requests from because this was the anchor system chosen for this suite in step 8 above.
- The Supported checkbox and the Format dropdown (purple arrows) are offered to the user because we had checked these boxes during suite creation in steps 11 and 12 above.

10.2 Client Suites

Much of the content in this section is similar to the Server Suites section. It is repeated here for users that are primarily interested in creating conformance suites for testing **Client** test systems. Please refer to the Peer-to-Peer testing guide to learn more about such testing.

This section assumes that you are already familiar with Conformance Testing. Please go through the Conformance Testing guide if you're not.

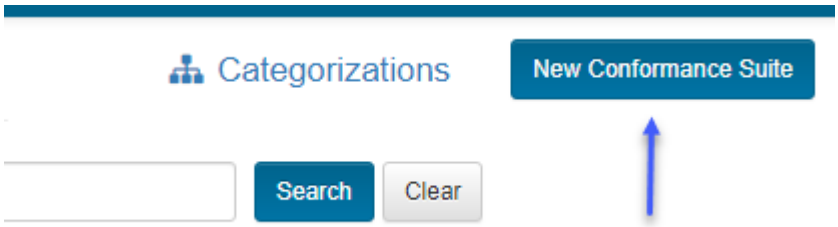
Test systems can be **Server** or **Client** test systems in Touchstone. This attribute is selected by the user on the Test System screen:



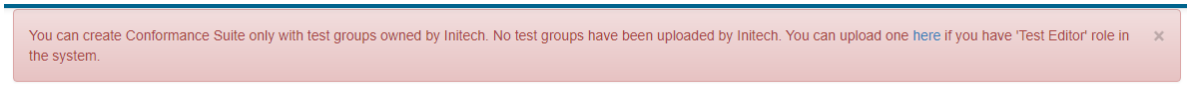
This section covers how to create Conformance Suites that are geared towards testing **Client** test systems. Here's how to create such a Conformance Suite:

1. Go to Conformance Suites page.

2. Click on **New Conformance Suite** button:



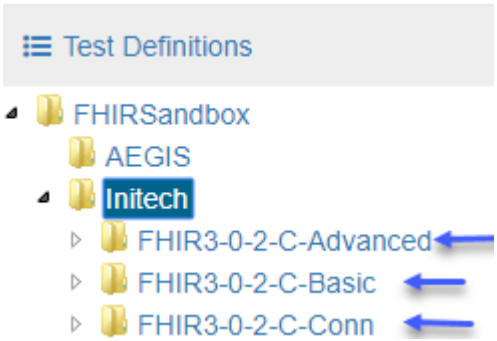
3. You will encounter the following error if you haven't uploaded test scripts to Touchstone yet:



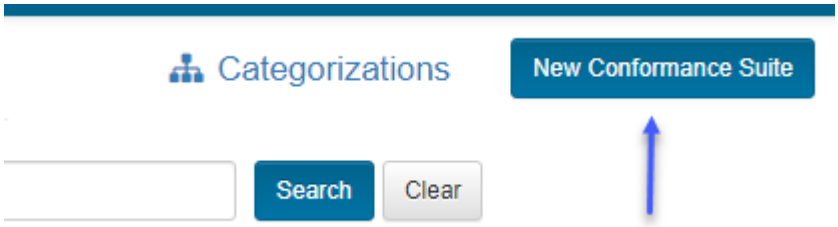
Note: Conformance Suites can be created only with test groups that your organization owns. Otherwise, the test group owner could inadvertently impact the Conformance results of many users in your program when a test group is modified.

To learn how to upload test groups to Touchstone, you can refer to the TestScript Authoring guide.

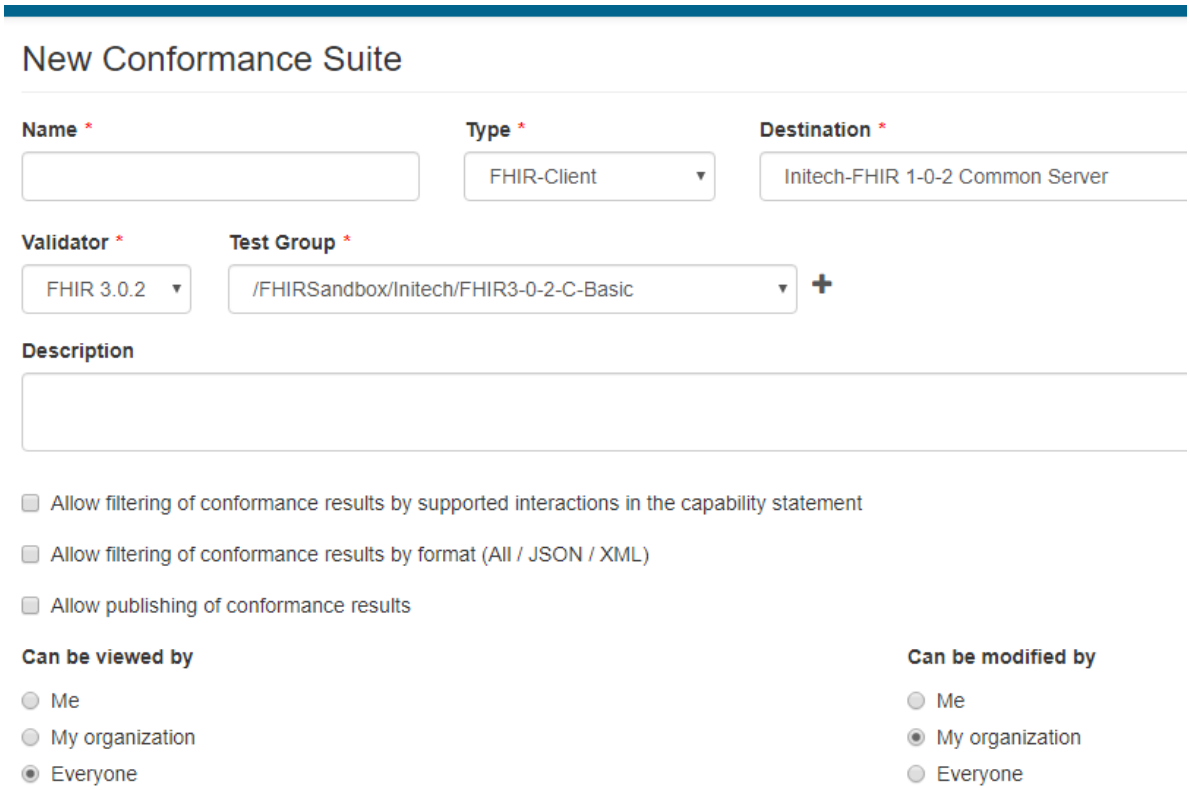
The user then uploads a few test groups:



4. Click on **New Conformance Suite** button again on the Conformance Suites page:



5. Below is the New Conformance Suite page:



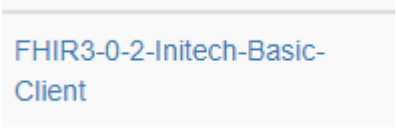
We will describe the meaning of each field as we populate them in this screen.

6. Enter the Name of the suite:

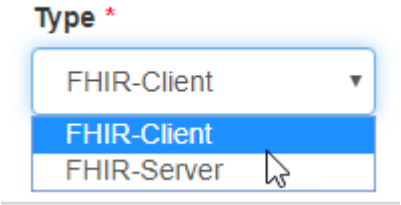
Note: Touchstone will prepend the specification name/version to the suite name. This is done to maintain uniformity and consistent naming convention across suites so it's easier for users to identify the specification that

a suite is testing for. The type of suite (**Server** vs **Client**) is also appended by Touchstone to the name for the same reason.

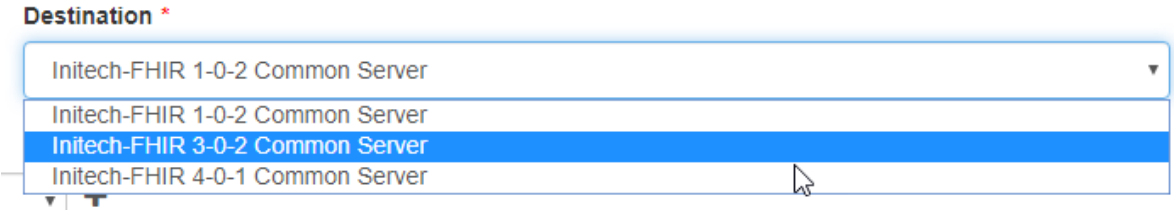
The final name of the suite in this instance would be the following:



7. Select the Type of the suite. For a suite that's geared towards testing **Client** test systems, the type would be **Client**:



8. Select the Anchor System of the suite. For a suite that's geared towards testing **Client** test systems (the SUT of the suite), the Anchor System would be a **Server/Destination** test system:



Conversely, if the suite is geared towards testing **Server** test systems (the SUT of the suite), the Anchor System would be a **Origin/Client** test system.

Anchor systems are needed to achieve comparable Conformance results among SUTs. If a suite uses different Anchor Systems, then the results would not be meaningful.

Anchor systems must be accessible by the organization that's creating the suite.

Note: It is highly recommended to use reliable anchor systems for your suites. This is to avoid unintended consequences on the program's conformance testing if the anchor system is deleted or its behavior is modified by the owning organization. Touchstone does not require the suite owner to also own the anchor system because it's not easy to make such systems available. It's rather common for organizations to rely on third parties for such systems. In this case, we're using **Initech-FHIR 3-0-2 Common Server** as the **Server anchor system** to which requests will be submitted from the **Client** test systems being tested in the suite.

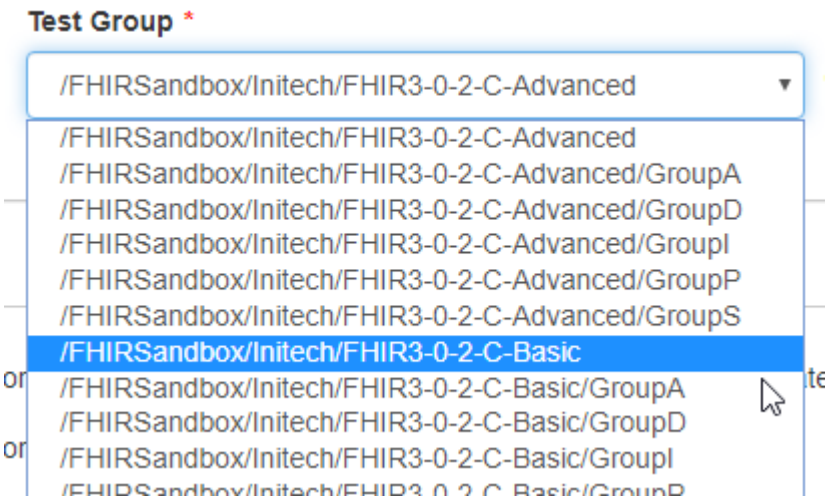
---

9. Select the Valdiator of the suite:

Validator is the AEGIS Validator to use for validating request and response payloads during test execution.

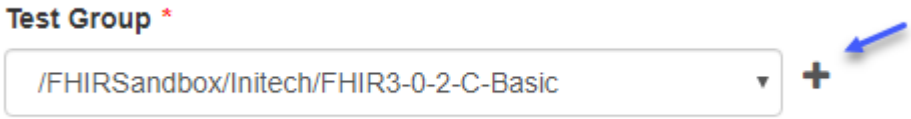
The list of Validators offered in the dropdown will be limited to the Validators used by the Test Groups owned by the suite author's organization.

9. Select the Test Group(s) of the suite:



Test Groups will contain the test scripts that the user will execute in conformance testing. The Test Group dropdown entries will be driven by the Validator selection. If you select **FHIR 3.0.2** as the Validator, for example, then the Test Group dropdown will be populated with FHIR 3.0.2 test groups that you have access to.

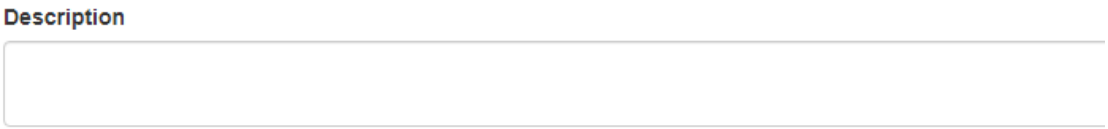
Additional test groups can be selected by clicking the **+** sign:



We'll cover multiple test group selection in the Categorization section of this guide.

Note: In order to be able to properly select test groups that were uploaded prior to Touchstone 5.0.0 release, you must re-upload them to Touchstone.

10. Enter a Description for the suite:

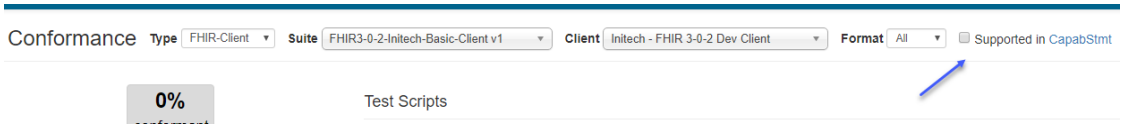


The system will generate a default description if none is entered.

11. Choose whether or not Supported checkbox is offered to users:

    ☑ Allow filtering of conformance results by supported interactions in the capability statement

    If checked, users will be offered the **Supported by CapabStmt** checkbox on conformance results screens. They would be able to filter results by supported interactions in the capability statement in addition to the default (All):

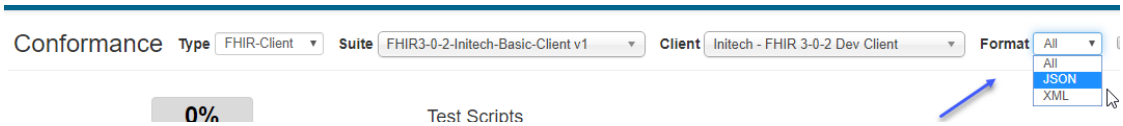
    

    Please refer to *this section* to see how this checkbox selection affects conformance results.

12. Choose whether or not Formats dropdown is offered to users:

    ☑ Allow filtering of conformance results by format (All / JSON / XML)

    If checked, users will be offered the **Format** dropdown on conformance results screens. They would be able to filter results by individual formats (JSON / XML) in addition to the default (All):

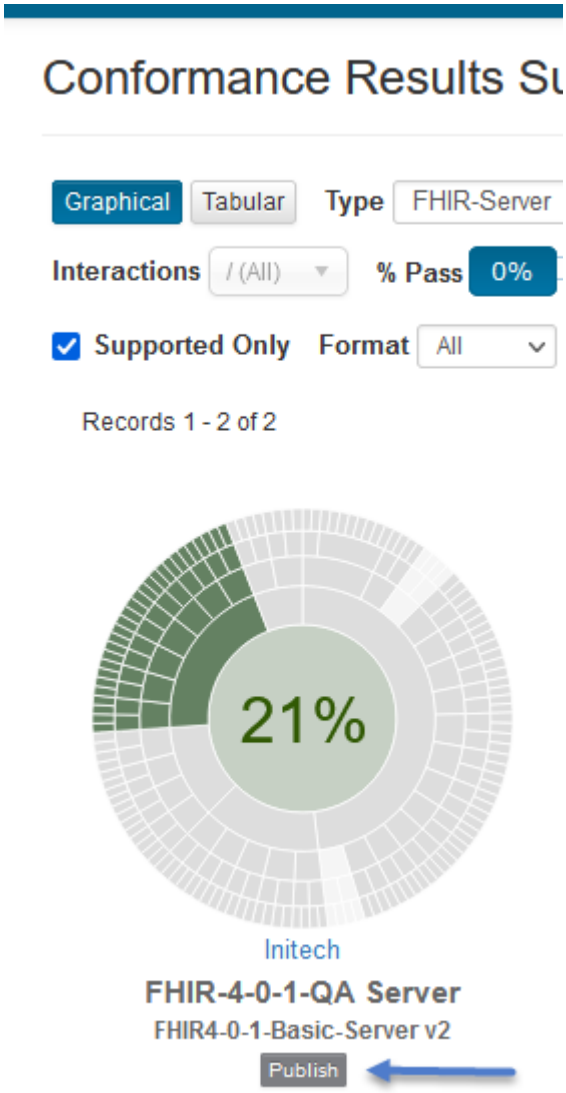
    

    Please refer to *this section* to see how this checkbox selection affects conformance results.

13. Choose whether or not conformance results can be Published by users:

    ☐ Allow publishing of conformance results

    If checked, users will be offered the option of publishing conformance results. Those results along with the associated test executions would become publicly accessible if published:



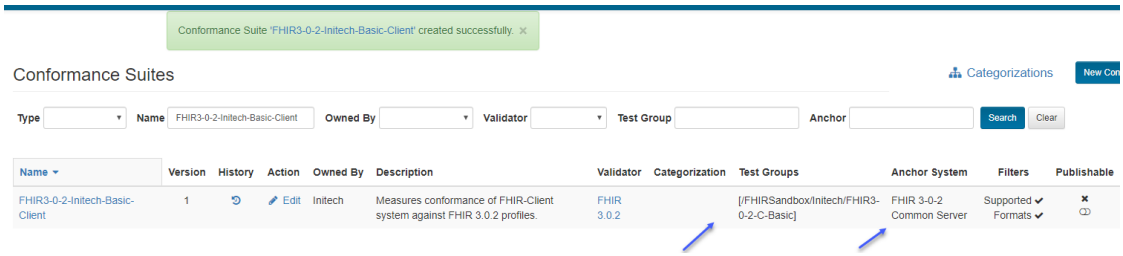
Please refer to Publishing to learn more about conformance results publishing.

14. Select the access attributes of the suite:

This works the same way as access attributes for test systems, test definitions, etc.

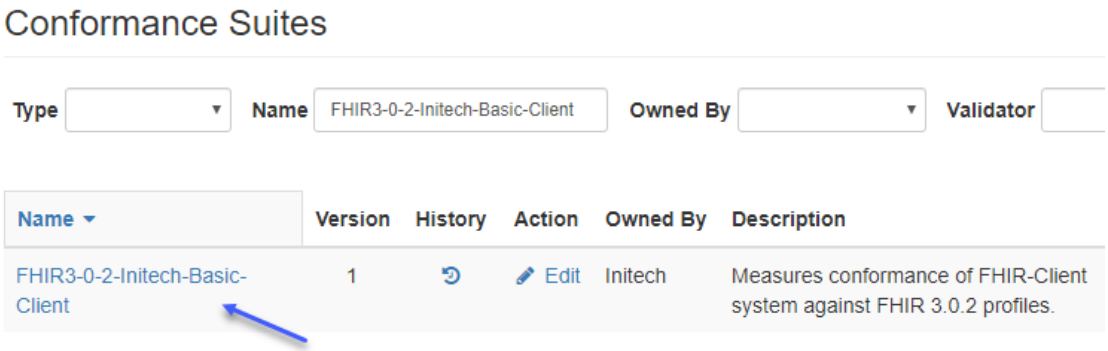
If you'd like only your organization to view and execute tests within the suite, then select `My Organization` for **Can be viewed by**.

15. Click on **Create**. Conformance Suite gets created:

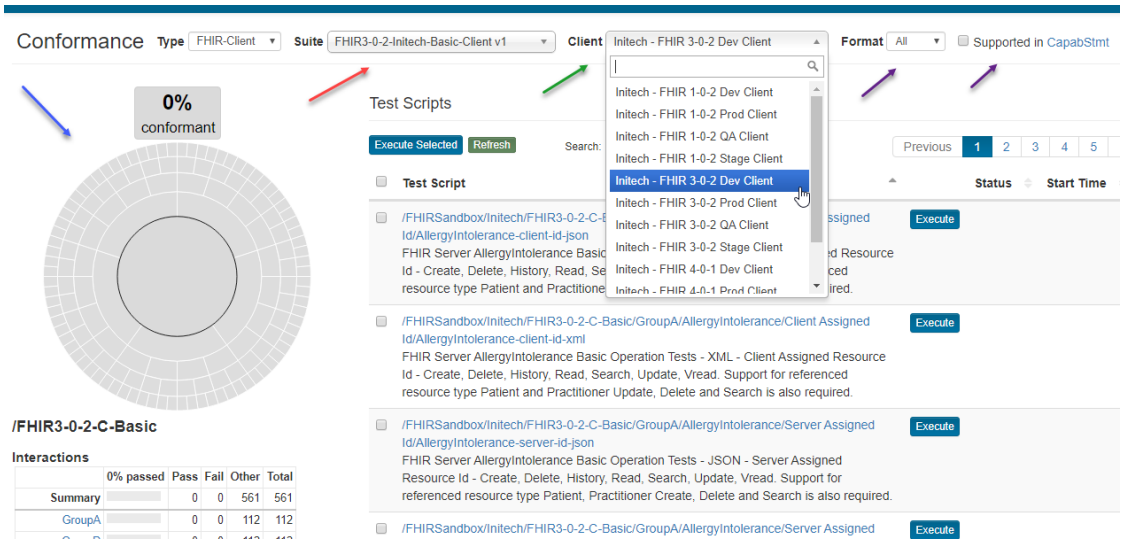


We selected **/FHIRSandbox/Initech/FHIR3-0-2-C-Basic** as the sole test group and **Initech-FHIR 3-0-2 Common Server** as the anchor system for this suite.

16. Click on the name of the suite:



17. You'll be taken to the Current Conformance page where you can test the suite:



- The selected Conformance Suite (red arrow) is the one we clicked on in the list of suites in prior step.
- The suite is made up of only one test group (blue arrow) as that was the only test group we selected during creation.
- The test system dropdown (green arrow) lists the **Client** test systems that you have access to. Only **Client** test systems are populated because this is a **Client** Conformance Suite.
- The anchor system chosen for the suite (Initech-FHIR 3-0-2 Common Server) is not listed in the test systems dropdown. The anchor system is the **Server** test system that requests will be submitted to by the chosen **Client** test system in a **Client** conformance suite. As users select different **Client** test systems to run this Conformance Suite against, the same anchor system (**Initech-FHIR 3-0-2 Common Server** in this case) will be used to submit requests to from these **Client** test systems. This allows for uniform comparison of **Client**-system conformance results. **Initech-FHIR 3-0-2 Common Server** is used to submit requests to because this was the anchor system chosen for this suite in step 8 above.
- The Supported checkbox and the Format dropdown (purple arrows) are offered to the user because we had checked these boxes during suite creation in steps 11 and 12 above.

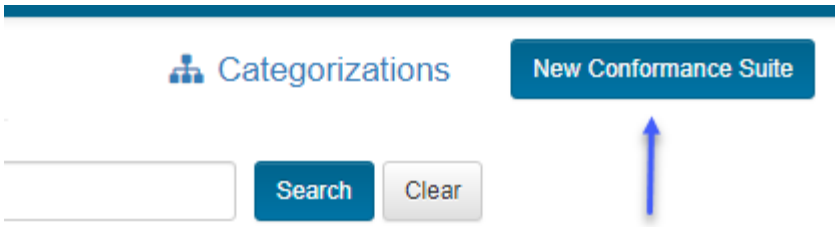
10.3 Categorization

10.3.1 Overview

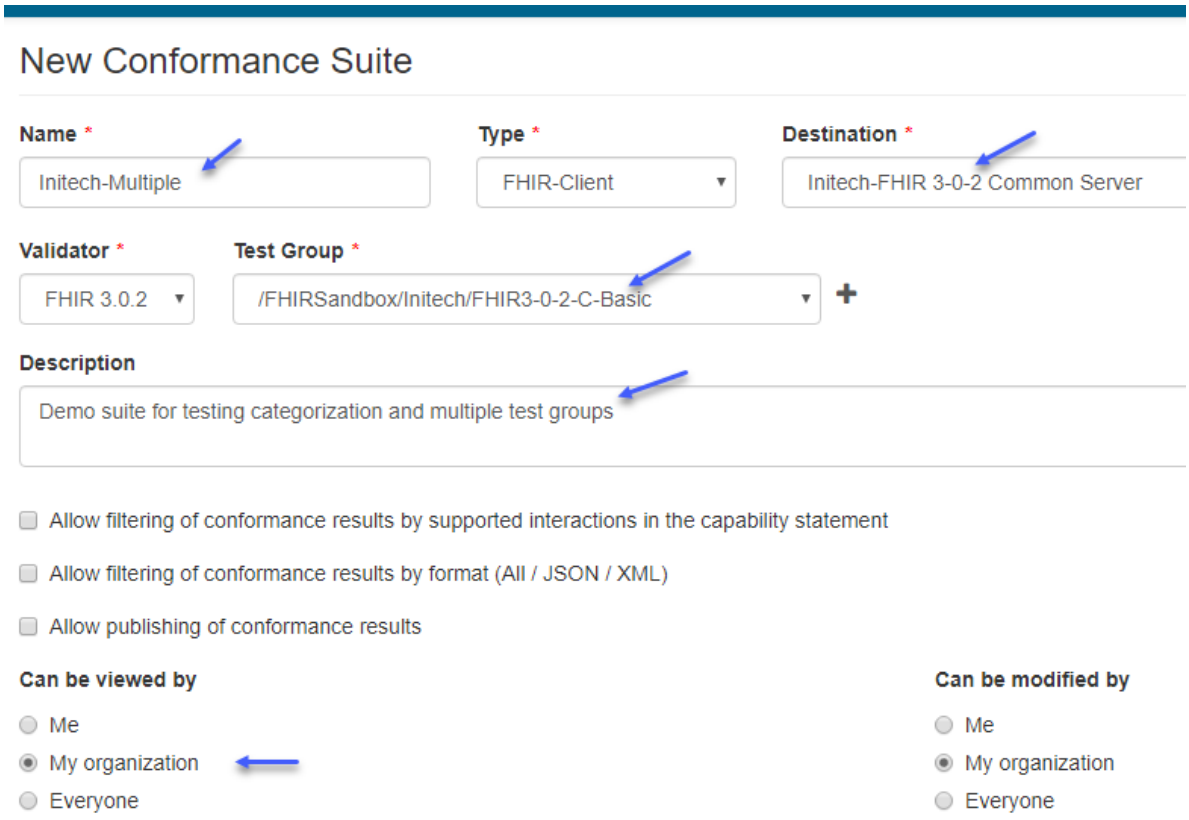
If a Conformance Suite is composed of more than one test group, then Categorization is required and interaction counts will be aggregated based on the Categorization definition.

We will create a suite with two test groups next:

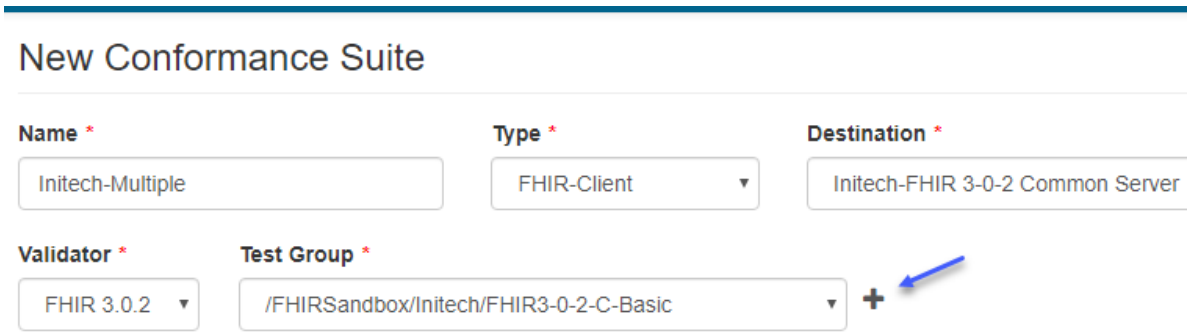
1. Click on "New Conformance Suite" on Conformance Suites page.



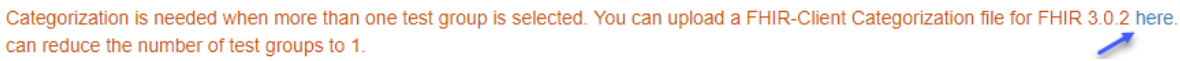
2. Enter the values for the fields as indicated by the blue arrows below:



3. Click the + sign:

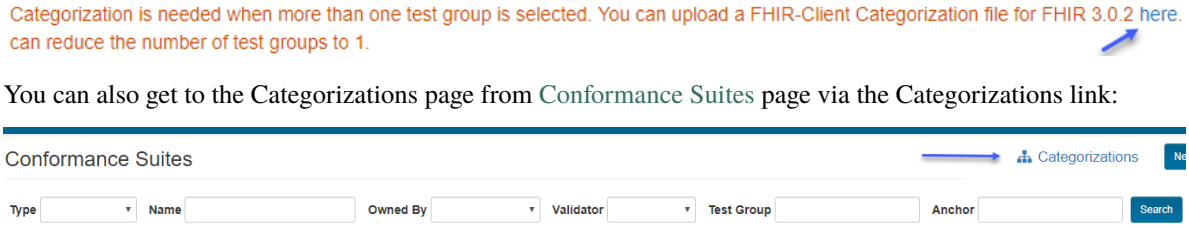


4. You'll get a warning indicating that Categorization is required when multiple test groups are selected:



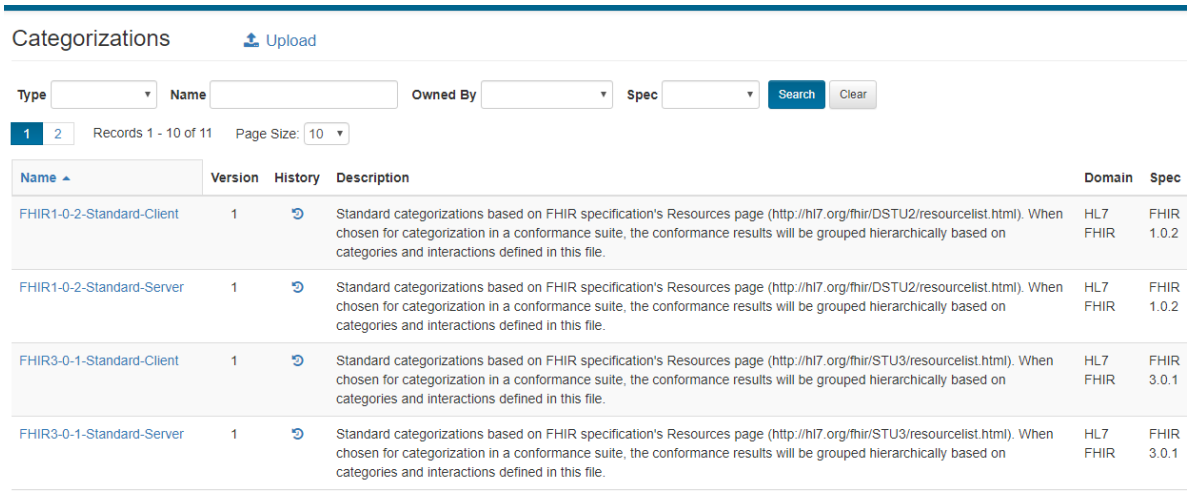
Touchstone uses the Categorization definition to channel the interactions of multiple test groups into one Result Summary chart on the Conformance Current page.

5. Open the link within the warning on a **new browser tab** to get to the Categorizations page:



You can also get to the Categorizations page from Conformance Suites page via the Categorizations link:

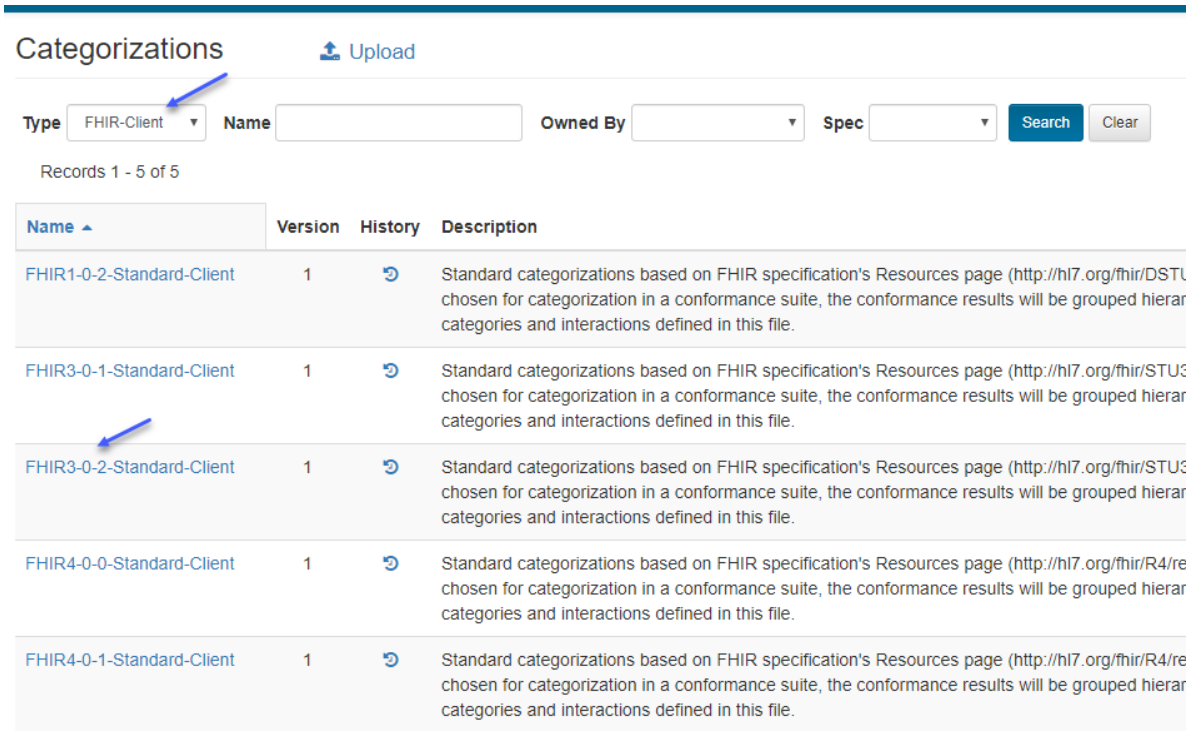
6. Here is the Categorizations page:



The page lists all the categorizations that you have view access to.

You can only use categorizations that your organization owns as part of your suites. This is to minimize the impact on your conformance program if other organizations modify the categorization. You can view other categorizations and copy them (assuming no Copyright infringement) into your own.

7. Filter for **FHIR-Client** categorizations (since the suite we're creating in this section is of **FHIR-Client** type) and click on **FHIR3-0-2-Standard-Client** categorization in the list:



8. Copy and paste the contents into a file in your XML editor:

Categorization

Name FHIR3-0-2-Standard-Client

Version 1

Description Standard categorizations based on FHIR specification's Resources page (http://hl7.org/fhir/STU3/resourcelist.html). When chosen for categorization in a conformance suite, the conformance results will be grouped hierarchically based on categories and interactions defined in this file.

Content

```
<?xml version="1.0" encoding="UTF-8"?>

<categ spec="FHIR 3.0.2" type="FHIR-Client" description="Standard categorizations based on FHIR specification's Resources page
(http://hl7.org/fhir/STU3/resourcelist.html). When chosen for categorization in a conformance suite, the conformance results will be
grouped hierarchically based on categories and interactions defined in this file.">
    <categ name="Resources">
        <categ name="Foundation">
            <categ name="Conformance">
                <categ name="CapabilityStatement" path="resource/CapabilityStatement"/>
                <categ name="StructureDefinition" path="resource/StructureDefinition"/>
                <categ name="ImplementationGuide" path="resource/ImplementationGuide"/>
                <categ name="SearchParameter" path="resource/SearchParameter"/>
                <categ name="MessageDefinition" path="resource/MessageDefinition"/>
                <categ name="OperationDefinition" path="resource/OperationDefinition"/>
                <categ name="CompartmentDefinition" path="resource/CompartmentDefinition"/>
                <categ name="StructureMap" path="resource/StructureMap"/>
                <categ name="GraphDefinition" path="resource/GraphDefinition"/>
                <categ name="DataElement" path="resource/DataElement"/>
            </categ>
            <categ name="Terminology">
                <categ name="CodeSystem" path="resource/CodeSystem"/>
                <categ name="ValueSet" path="resource/ValueSet"/>
                <categ name="ConceptMap" path="resource/ConceptMap"/>
                <categ name="ExpansionProfile" path="resource/ExpansionProfile"/>
                <categ name="NamingSystem" path="resource/NamingSystem"/>
            </categ>
            <categ name="Security">
                <categ name="Provenance" path="resource/Provenance"/>
                <categ name="AuditEvent" path="resource/AuditEvent"/>
                <categ name="Consent" path="resource/Consent"/>
            </categ>
```

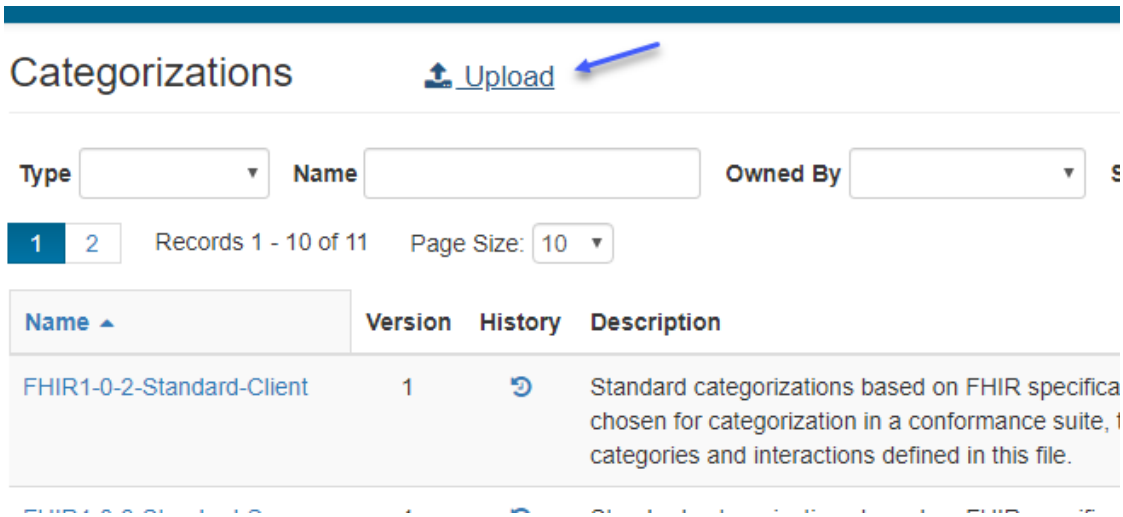
We've saved the contents into a file called "**InitechCert.xml**":

9. Change the description appropriately:

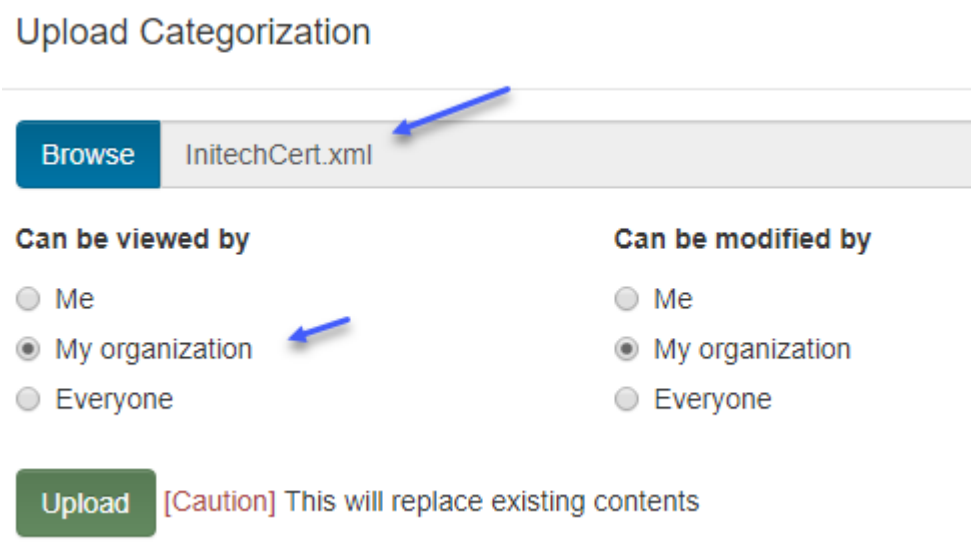
InitechCert.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<categ spec="FHIR 3.0.2" type="FHIR-Client"
    description="Initech Cert Categorization for FHIR 3.0.2 Client">
    <categ name="Resources">
        <categ name="Foundation">
            <categ name="Conformance">
                <categ name="CapabilityStatement" path="resource/CapabilityStatement" />
                <categ name="StructureDefinition" path="resource/StructureDefinition" />
                <categ name="ImplementationGuide" path="resource/ImplementationGuide" />
                <categ name="SearchParameter" path="resource/SearchParameter" />
                <categ name="MessageDefinition" path="resource/MessageDefinition" />
                <categ name="OperationDefinition" path="resource/OperationDefinition" />
                <categ name="CompartmentDefinition" path="resource/CompartmentDefinition" /
                <categ name="StructureMap" path="resource/StructureMap" />
                <categ name="GraphDefinition" path="resource/GraphDefinition" />
                <categ name="DataElement" path="resource/DataElement" />
            </categ>
            <categ name="Terminology">
                <categ name="CodeSystem" path="resource/CodeSystem" />
                <categ name="ValueSet" path="resource/ValueSet" />
                <categ name="ConceptMap" path="resource/ConceptMap" />
                <categ name="ExpansionProfile" path="resource/ExpansionProfile" />
                <categ name="NamingSystem" path="resource/NamingSystem" />
            </categ>
            <categ name="Security">
                <categ name="Provenance" path="resource/Provenance" />
                <categ name="AuditEvent" path="resource/AuditEvent" />
                <categ name="Consent" path="resource/Consent" />
            </categ>
            <categ name="Documents">
                <categ name="Composition" path="resource/Composition" />
                <categ name="DocumentManifest" path="resource/DocumentManifest" />
                <categ name="DocumentReference" path="resource/DocumentReference" />
```

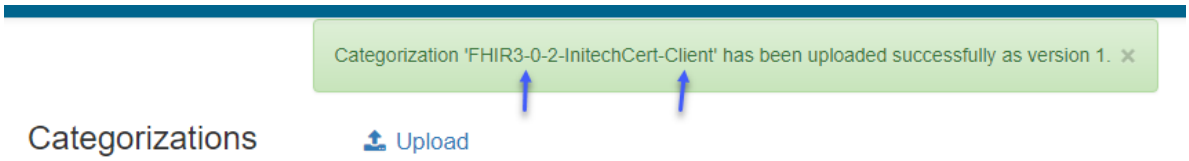
10. Upload the Categorization file on the Categorizations page:



11. Browse to the InitechCert.xml file on your machine and click on Upload button:



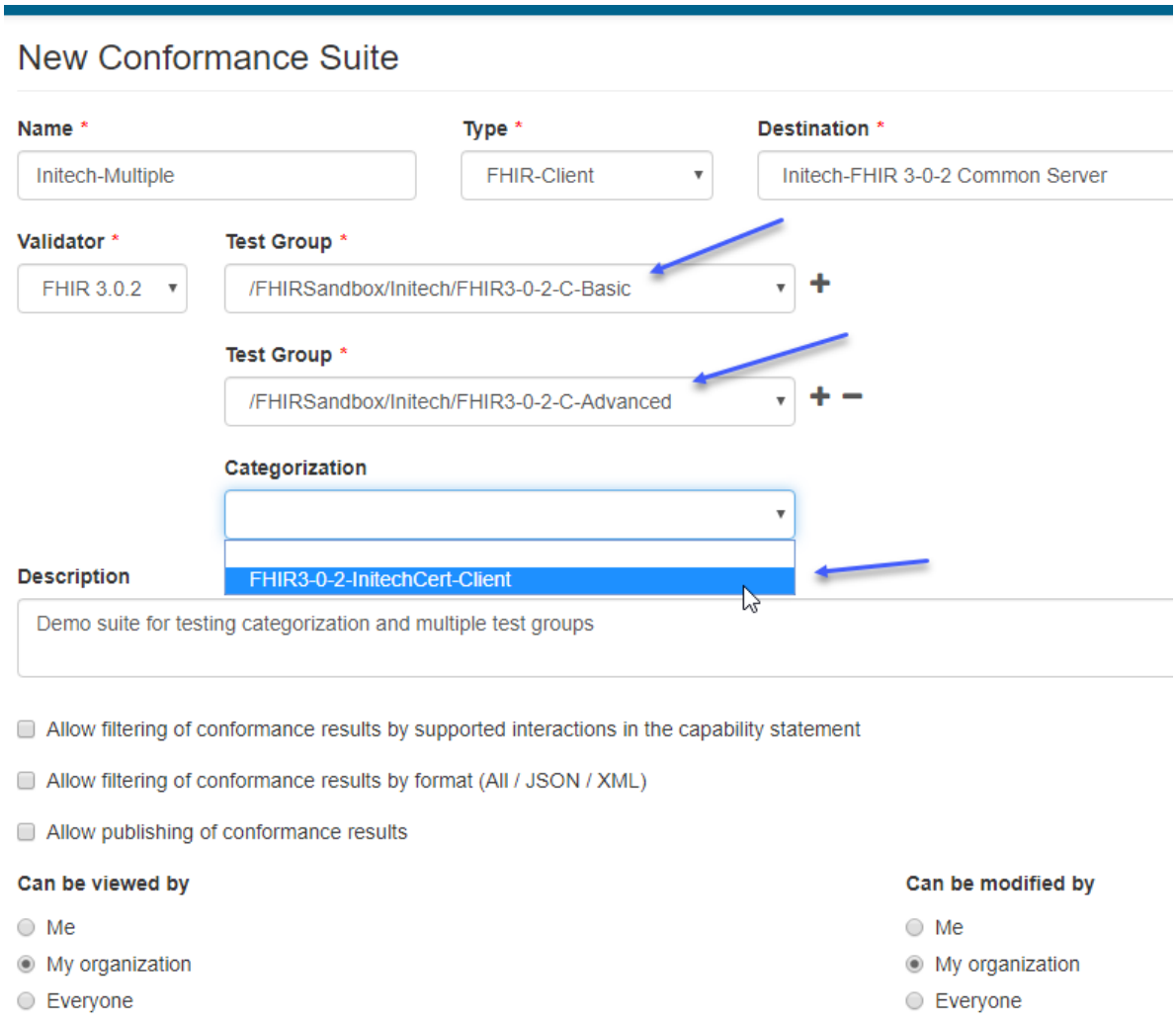
12. The Categorization will take on the name of the uploaded file. Touchstone will prepend the Specification name/version and will append the type to the name of the file to enforce uniform naming convention across categorizations so they're easier to identify by end-users:



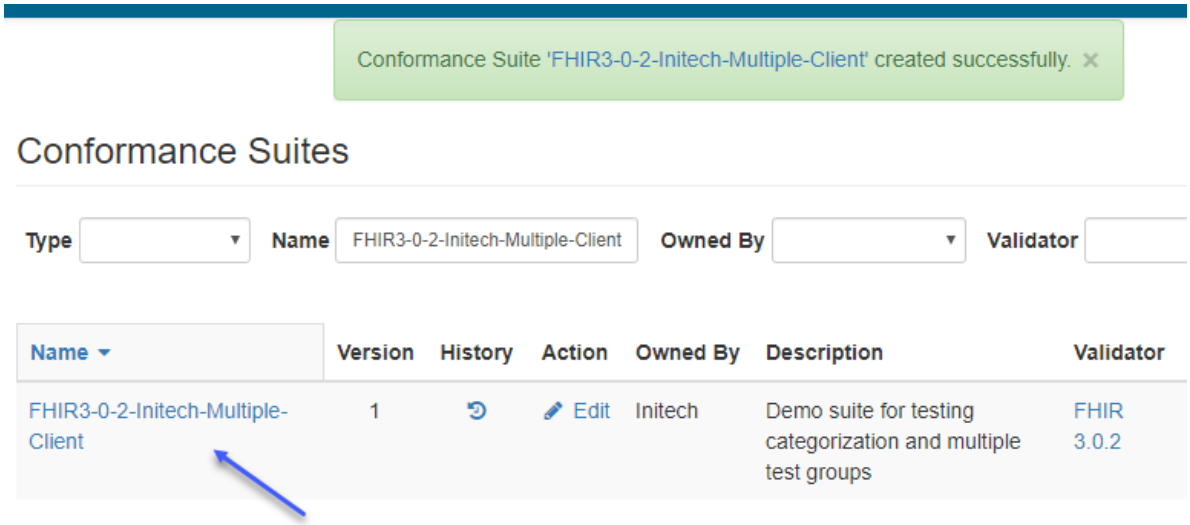
13. Resume where we left off on *New Conformance Suite* page from step 4.

    Re-enter the field values if you had to refresh the screen and click on the **+** sign again.

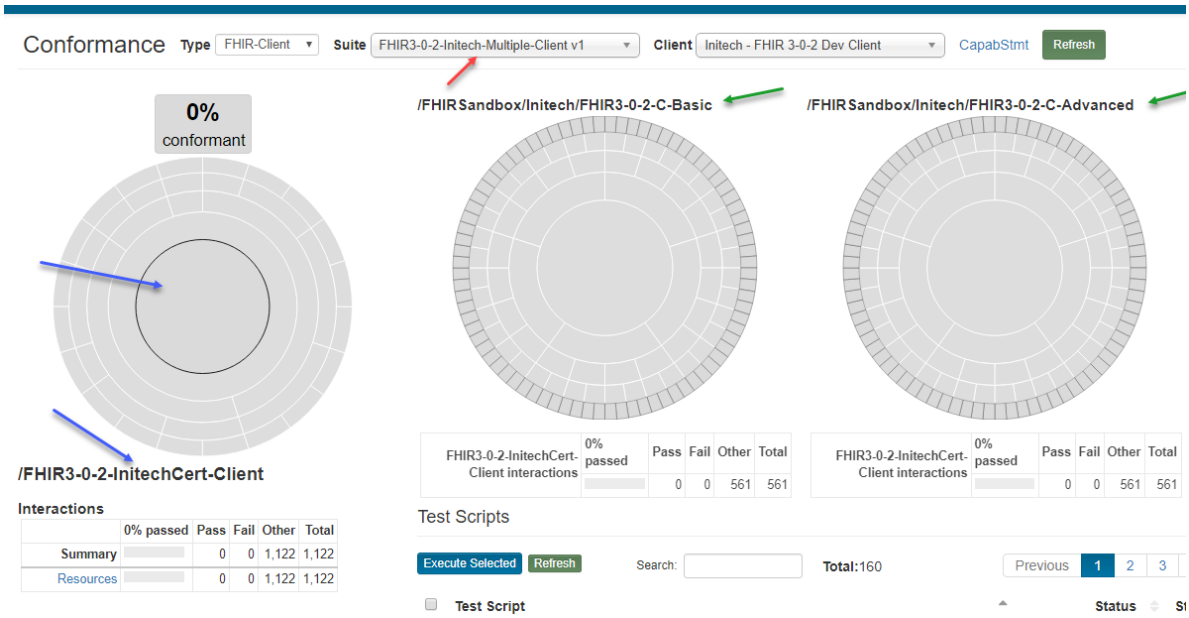
    This time the system will not generate a warning and will allow you to select the Categorization that we just created:



14. Click on Create and then select the suite that got created:

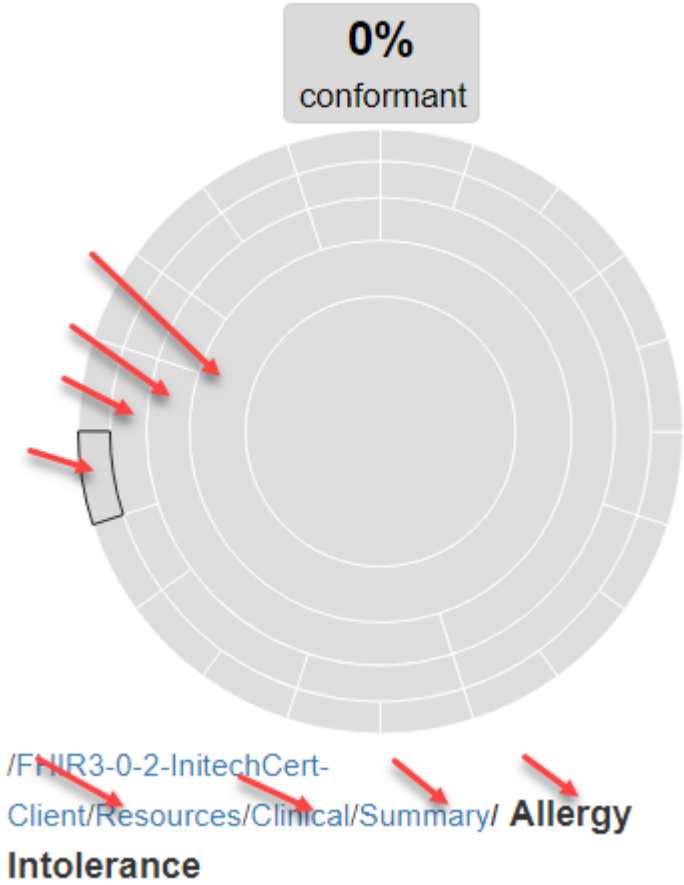


15. Notice that the root node of the Results Summary chart (blue arrows below) matches the name of the Categorization file that we just uploaded.



We have successfully created a conformance suite with two test groups (green arrows above) that uses a Categorization.

When Categorization is used in a suite, the Result Summary chart is entirely driven by the Categorization. The individual **nodes/bands** within the Result Summary chart (red arrows below) will be identical to the individual **categories** within the Categorization (blue arrows below):




```
InitechCert.xml
3  <categ spec="FHIR 3.0.2" type="FHIR-Client"
4      description="Initech Cert Categorization for FHIR 3.0.2 Client">
5      <categ name="Resources">
6          <categ name="Foundation">
48         <categ name="Base">
84         <categ name="Clinical">
85             <categ name="Summary">
86                 <categ name="AllergyIntolerance" path="resource/AllergyIntolerance"/>
87                 <categ name="AdverseEvent" path="resource/AdverseEvent" />
88                 <categ name="Condition" path="resource/Condition" />
89                 <categ name="Procedure" path="resource/Procedure" />
90                 <categ name="FamilyMemberHistory" path="resource/FamilyMemberHistory" />
91                 <categ name="ClinicalImpression" path="resource/ClinicalImpression" />
92                 <categ name="DetectedIssue" path="resource/DetectedIssue" />
93             </categ>
94             <categ name="Diagnostics">
```

The terms **node** and **category** are hence synonymous when Categorization is used. We will use the terms interchangeably in the remainder of this section.

10.3.2 Definition

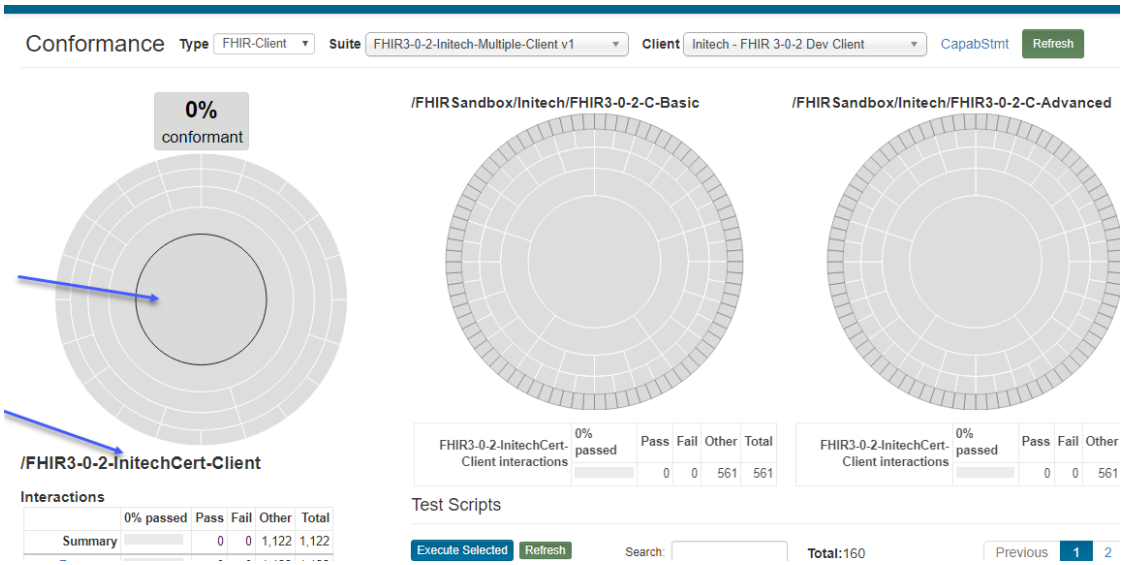
We will dive deeper into the Categorization file contents in this section and see how it impacts the Conformance Results.

10.3.2.1 Root Node

Notice that the root node (blue arrow below) does not have a name:

```
InitechCert.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <categ spec="FHIR 3.0.2" type="FHIR-Client"
4      description="Initech Cert Categorization for FHIR 3.0.2 Client">
5      <categ name="Resources">
6          <categ name="Foundation">
7              <categ name="Conformance">
8                  <categ name="CapabilityStatement" path="resource/CapabilityStatement" />
9                  <categ name="StructureDefinition" path="resource/StructureDefinition" />
10                 <categ name="ImplementationGuide" path="resource/ImplementationGuide" />
11                 <categ name="SearchParameter" path="resource/SearchParameter" />
12                 <categ name="MessageDefinition" path="resource/MessageDefinition" />
13                 <categ name="OperationDefinition" path="resource/OperationDefinition" />
14                 <categ name="CompartmentDefinition" path="resource/CompartmentDefinition" />
15                 <categ name="StructureMap" path="resource/StructureMap" />
16                 <categ name="GraphDefinition" path="resource/GraphDefinition" />
17                 <categ name="DataElement" path="resource/DataElement" />
18             </categ>
19             <categ name="Terminology">
20                 <categ name="CodeSystem" path="resource/CodeSystem" />
21                 <categ name="ValueSet" path="resource/ValueSet" />
22                 <categ name="ConceptMap" path="resource/ConceptMap" />
```

Touchstone will use the Categorization file name (red arrow above) as the name of the root node which in turn gets used as the name of the root node on the Results Summary chart (blue arrows below):

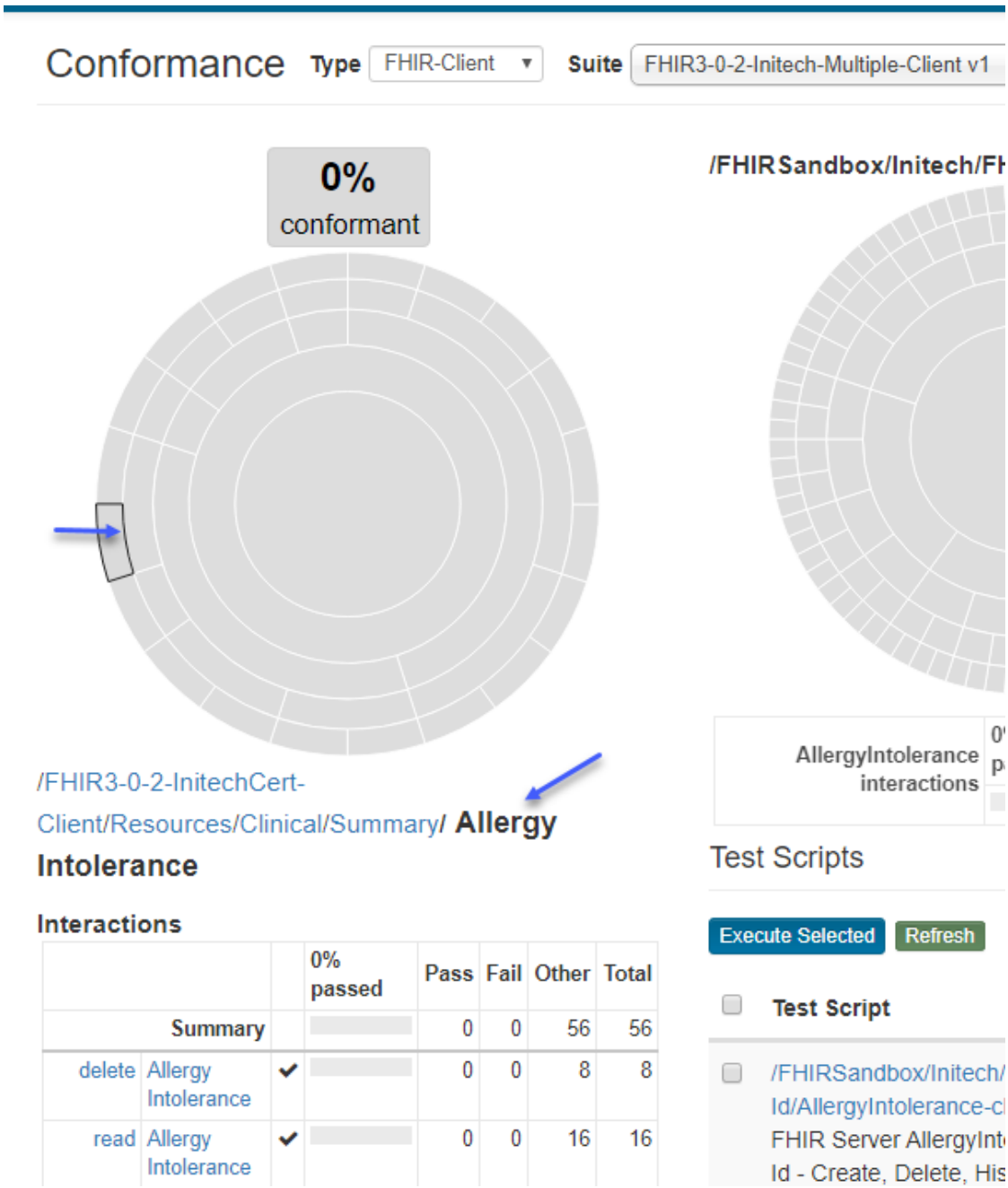


10.3.2.2 Leaf Nodes

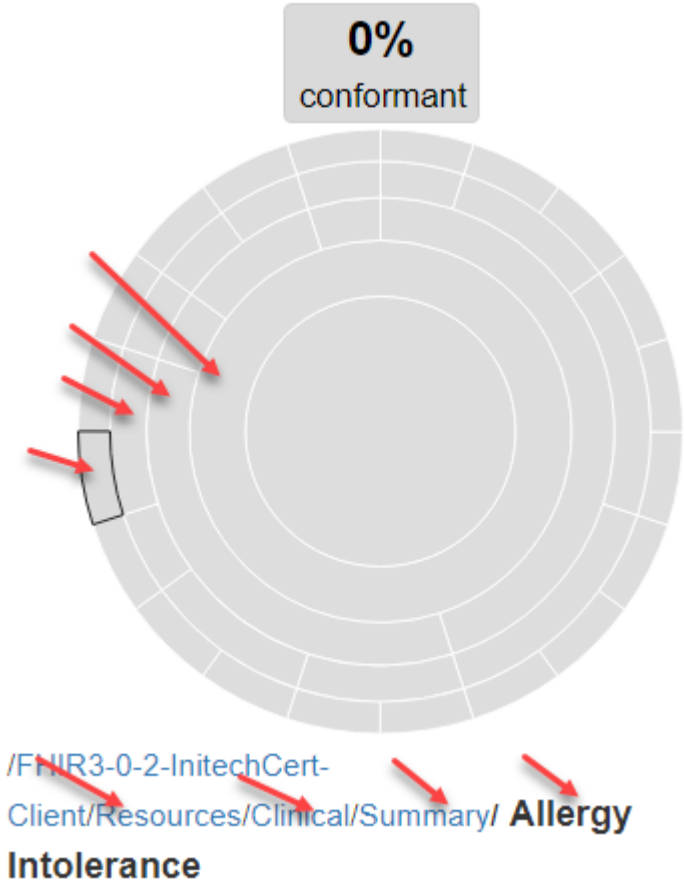
Leaf categories are those that have no more categories beneath them:

```
        </categ>
        <categ name="Management">
            <categ name="Encounter" path="resource/Encounter" />
            <categ name="EpisodeOfCare" path="resource/EpisodeOfCare" />
            <categ name="Flag" path="resource/Flag" />
            <categ name="List" path="resource/List" />
            <categ name="Library" path="resource/Library" />
        </categ>
    </categ>
    <categ name="Clinical">
        <categ name="Summary">
            <categ name="AllergyIntolerance" path="resource/AllergyIntolerance" />
            <categ name="AdverseEvent" path="resource/AdverseEvent" />
            <categ name="Condition" path="resource/Condition" />
            <categ name="Procedure" path="resource/Procedure" />
            <categ name="FamilyMemberHistory" path="resource/FamilyMemberHistory" />
            <categ name="ClinicalImpression" path="resource/ClinicalImpression" />
            <categ name="DetectedIssue" path="resource/DetectedIssue" />
        </categ>
        <categ name="Diagnostics">
            <categ name="Observation" path="resource/Observation" />
            <categ name="DiagnosticReport" path="resource/DiagnosticReport" />
            <categ name="Specimen" path="resource/Specimen" />
            <categ name="BodySite" path="resource/BodySite" />
            <categ name="ImagingStudy" path="resource/ImagingStudy" />
            <categ name="ImagingManifest" path="resource/ImagingManifest" />
            <categ name="QuestionnaireResponse" path="resource/QuestionnaireResponse" /
            <categ name="Sequence" path="resource/Sequence" />
        </categ>
```

They correspond to the outermost nodes in the Result Summary chart when categorization is used:



Notice that the hierarchy of nodes (red arrows below) in the Results Summary chart matches the hierarchy of categories (blue arrows below) in the file definition:




```
InitechCert.xml
3  <categ spec="FHIR 3.0.2" type="FHIR-Client"
4      description="Initech Cert Categorization for FHIR 3.0.2 Client">
5      <categ name="Resources">
6          <categ name="Foundation">
48         <categ name="Base">
84         <categ name="Clinical">
85             <categ name="Summary">
86                 <categ name="AllergyIntolerance" path="resource/AllergyIntolerance"/>
87                 <categ name="AdverseEvent" path="resource/AdverseEvent" />
88                 <categ name="Condition" path="resource/Condition" />
89                 <categ name="Procedure" path="resource/Procedure" />
90                 <categ name="FamilyMemberHistory" path="resource/FamilyMemberHistory" />
91                 <categ name="ClinicalImpression" path="resource/ClinicalImpression" />
92                 <categ name="DetectedIssue" path="resource/DetectedIssue" />
93             </categ>
94             <categ name="Diagnostics">
```

Side note: When categorization is not used then the outermost node corresponds to individual test scripts within the test group as indicated in *this section*.

10.3.2.3 Parent Nodes

The Parent categories are pointed to by the (blue arrows below):

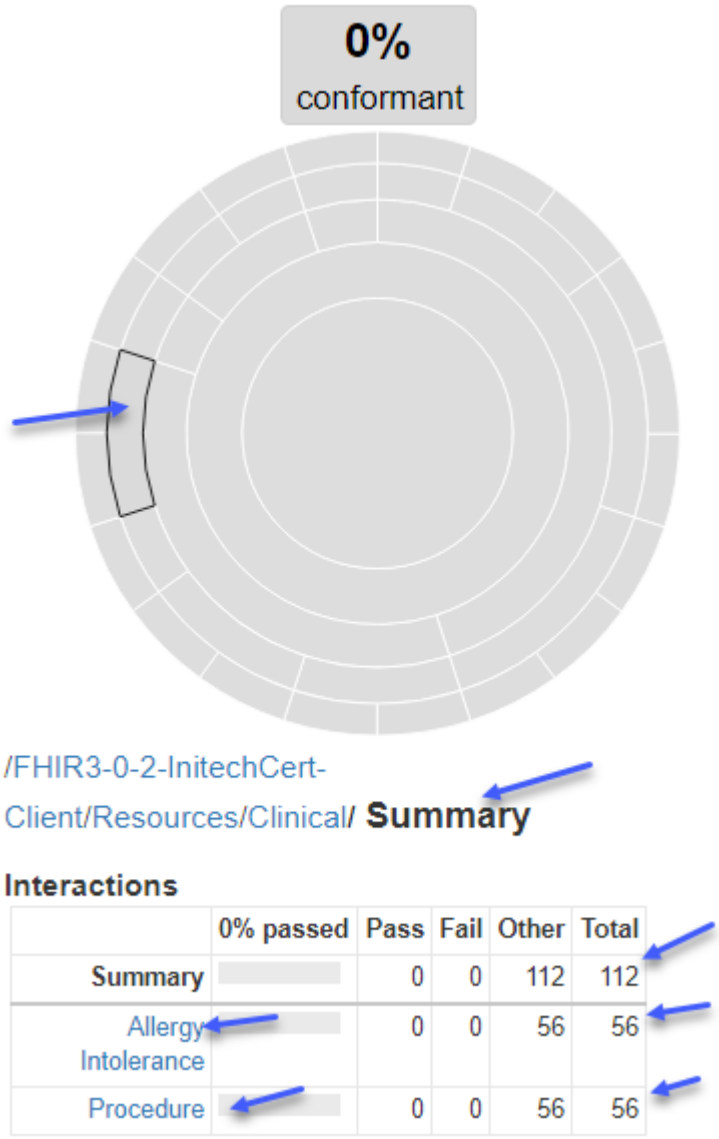
InitechCert.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<categ spec="FHIR 3.0.2" type="FHIR-Client"
    description="Initech Cert Categorization for FHIR 3.0.2 Client">
    <categ name="Resources">
        <categ name="Foundation">
            <categ name="Conformance">
                <categ name="CapabilityStatement" path="resource/CapabilityStatement" />
                <categ name="StructureDefinition" path="resource/StructureDefinition" />
                <categ name="ImplementationGuide" path="resource/ImplementationGuide" />
                <categ name="SearchParameter" path="resource/SearchParameter" />
                <categ name="MessageDefinition" path="resource/MessageDefinition" />
                <categ name="OperationDefinition" path="resource/OperationDefinition" />
                <categ name="CompartmentDefinition" path="resource/CompartmentDefinition" />
                <categ name="StructureMap" path="resource/StructureMap" />
                <categ name="GraphDefinition" path="resource/GraphDefinition" />
                <categ name="DataElement" path="resource/DataElement" />
            </categ>
            <categ name="Terminology">
                <categ name="CodeSystem" path="resource/CodeSystem" />
                <categ name="ValueSet" path="resource/ValueSet" />
                <categ name="ConceptMap" path="resource/ConceptMap" />
                <categ name="ExpansionProfile" path="resource/ExpansionProfile" />
                <categ name="NamingSystem" path="resource/NamingSystem" />
            </categ>
            <categ name="Security">
                <categ name="Provenance" path="resource/Provenance" />
                <categ name="AuditEvent" path="resource/AuditEvent" />
                <categ name="Consent" path="resource/Consent" />
            </categ>
            <categ name="Documents">
                <categ name="Composition" path="resource/Composition" />
                <categ name="DocumentManifest" path="resource/DocumentManifest" />
                <categ name="DocumentReference" path="resource/DocumentReference" />
            </categ>
            <categ name="Other">
                <categ name="Basic" path="resource/Basic" />
                <categ name="Binary" path="resource/Binary" />
                <categ name="Bundle" path="resource/Bundle" />
                <categ name="Linkage" path="resource/Linkage" />
                <categ name="Media" path="resource/Media" />
                <categ name="MessageHeader" path="resource/MessageHeader" />
                <categ name="OperationOutcome" path="resource/OperationOutcome" />
                <categ name="Parameters" path="resource/Parameters" />
                <categ name="Subscription" path="resource/Subscription" />
            </categ>
        </categ>
        <categ name="Base">
            <categ name="Individuals">
                <categ name="Patient" path="resource/Patient" />
                <categ name="Practitioner" path="resource/Practitioner" />
```

You can organize the nodes any way you like.

10.3.2.4 Node Deletion

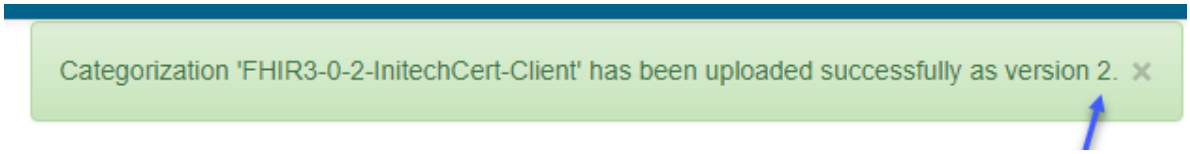
1. Notice the existing interaction counts for the Summary category:



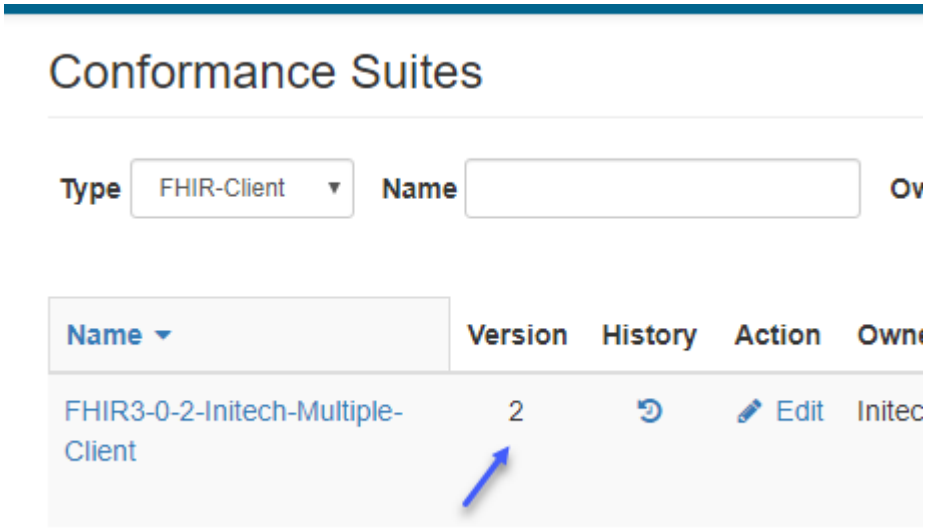
2. Remove the AllergyIntolerance category from the file:

```
<categ name="Foundation">..
<categ name="Base">..
<categ name="Clinical">
    <categ name="Summary">
        <categ name="AllergyIntolerance" path="resource/AllergyIntolerance" />
        <categ name="AdverseEvent" path="resource/AdverseEvent" />
        <categ name="Condition" path="resource/Condition" />
        <categ name="Procedure" path="resource/Procedure" />
        <categ name="FamilyMemberHistory" path="resource/FamilyMemberHistory" />
        <categ name="ClinicalImpression" path="resource/ClinicalImpression" />
```

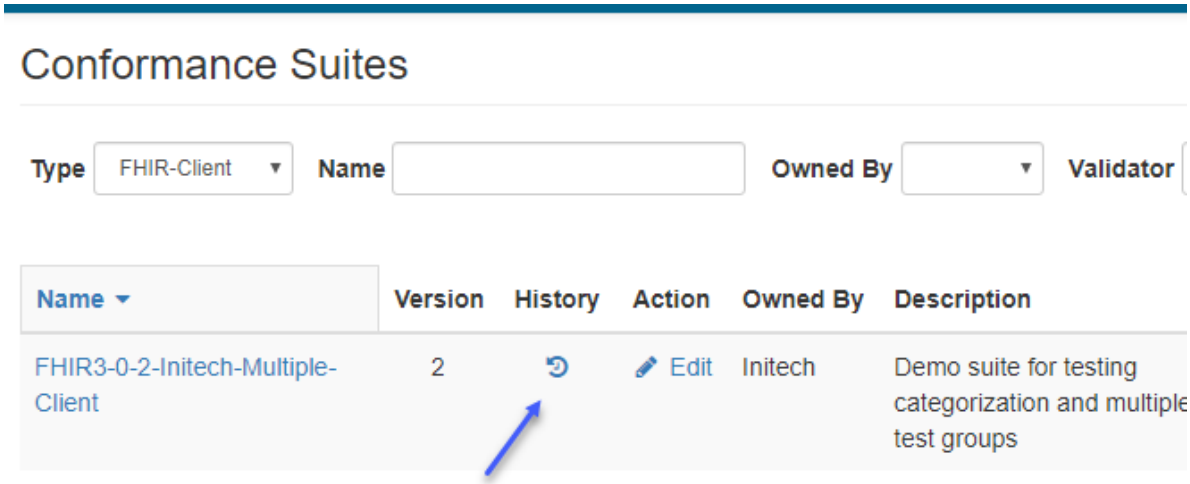
3. Re-upload the categorization file. Notice that the version increments:



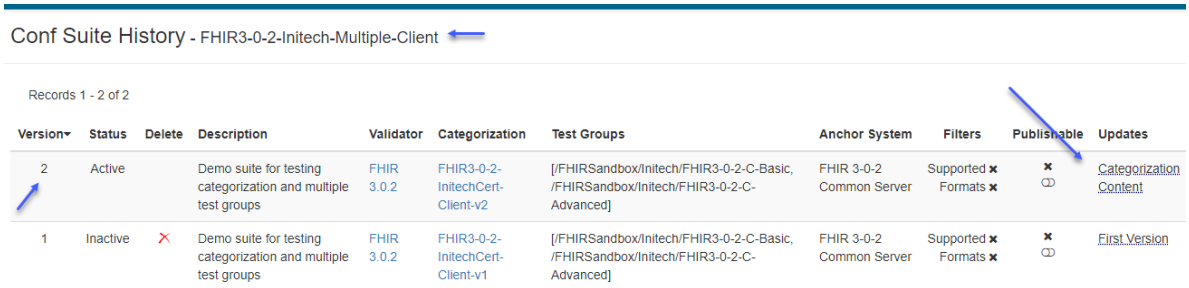
Notice also that the Conformance Suite version increments as well:



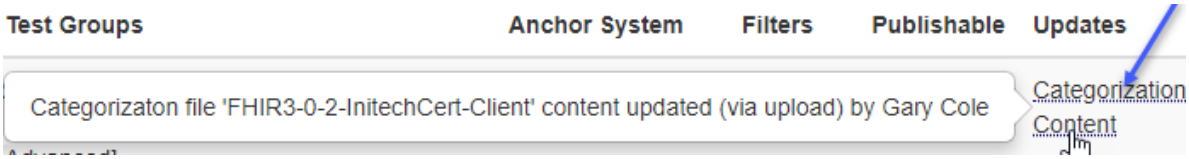
Click on the **history** icon to see the history of the suite:



The cause of the version increments is shown under the **Updates** column:

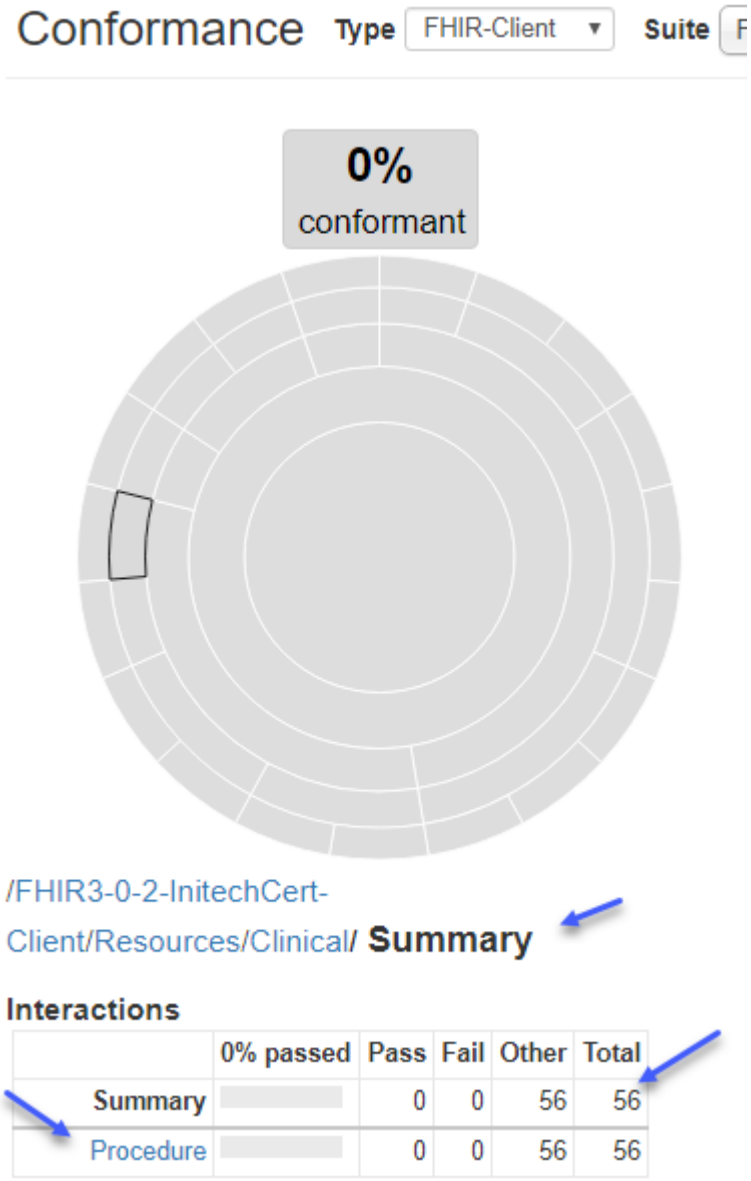


Hovering over the text gives more details about the cause:



4. Go back to the Current page

   Observe the new interaction counts for the Summary category. The AllergyIntolerance node is absent as it was removed from the file:



10.3.2.5 Paths

This section explains how the interactions in the test groups (that are part of a suite with categorization) get categorized into different buckets based on the Categorization definition.

The leaf nodes of the Categorization have the categorization **paths** specified:

```
    <categ name="Foundation">..
    <categ name="Base">
        <categ name="Individuals">
            <categ name="Patient" path="resource/Patient" />
            <categ name="Practitioner" path="resource/Practitioner" />
            <categ name="PractitionerRole" path="resource/PractitionerRole" />
            <categ name="RelatedPerson" path="resource/RelatedPerson" />
            <categ name="Person" path="resource/Person" />
            <categ name="Group" path="resource/Group" />
        </categ>
        <categ name="Entities">..
        <categ name="WorkFlow">..
        <categ name="Management">..
    </categ>
    <categ name="Clinical">..
    <categ name="Financial">..
    <categ name="Specialized">..
</categ>
<categ name="Operations">
    <categ name="$apply" path="operation/apply" />
    <categ name="$closure" path="operation/closure" />
```

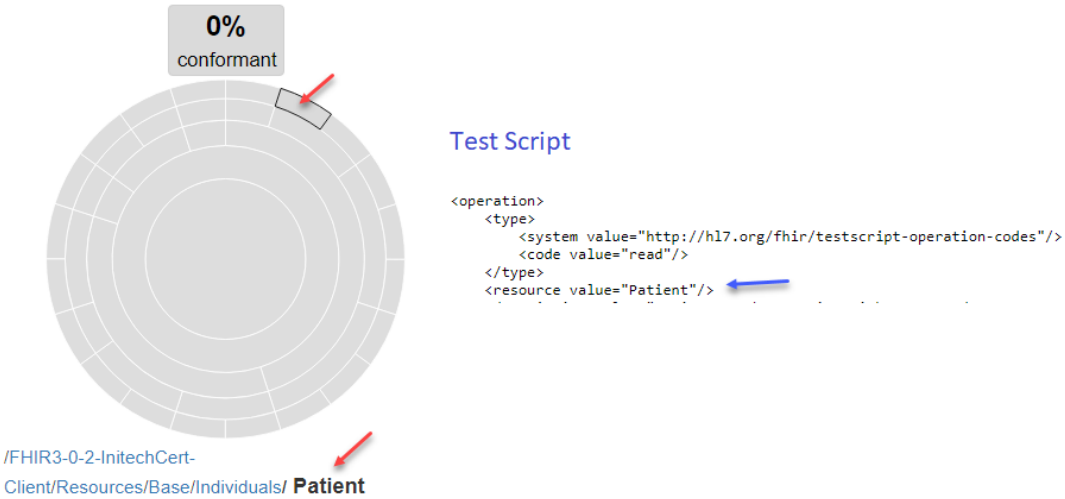
10.3.2.5.1 FHIR

FHIR paths are of three types:

- **resource paths**: whose values take the form "**resource/[Type]**" where **Type** is one of the values listed on FHIR Resource Types.

```
    <categ name="Patient" path="resource/Patient" />
    <categ name="Practitioner" path="resource/Practitioner" />
    <categ name="PractitionerRole" path="resource/PractitionerRole" />
    <categ name="RelatedPerson" path="resource/RelatedPerson" />
    <categ name="Person" path="resource/Person" />
    <categ name="Group" path="resource/Group" />
```

  For example, all test scripts in the suite that invoke operations involving the **Patient** resource (blue arrow below) would get aggregated into the Patient category (red arrow below)



- **operation paths**: whose values take the form "**operation/[Type]**" where **Type** is one of the extended operation values listed on FHIR Extended Operations. There are ways to define extended operations not on this list and use them in Categorization. Please contact Touchstone_Support@aegis.net for details.

```
<categ name="Operations">
    <categ name="$apply" path="operation/apply" />
    <categ name="$closure" path="operation/closure" />
    <categ name="$compose" path="operation/compose" />
    <categ name="$conforms" path="operation/conforms" />
    <categ name="$data-requirements" path="operation/data-requirements" />
    <categ name="$document" path="operation/document" />
    <categ name="$evaluate" path="operation/evaluate" />
    <categ name="$evaluate-measure" path="operation/evaluate-measure" />
    <categ name="$everything" path="operation/everything" />
    <categ name="$expand" path="operation/expand" />
```

```
<operation>
    <type>
        <system value="http://terminology.hl7.org/CodeSystem/testscript-operation-codes"/>
        <code value="expand"/>
    </type>
    <resource value="ValueSet"/>
    <description value="Test $expand operation with text filter 'f' on ValueSet extensional-case-4."/>
```

- **whole system paths**: whose values take the form "**/[Type]**" or "interaction/**[Type]**" where
  - "**/[Type]**" can be "**/capabilities**" or "**/metadata**", and..
  - "interaction/**[Type]**" can be "interaction/**transaction**", "interaction/**batch**", "interaction/**search-system**", or "interaction/**history-system**".

  See this section on FHIR API page:

Whole System Interactions	
capabilities	Get a capability statement for the system
batch/transaction	Update, create or delete a set of resources in a single interaction
history	Retrieve the change history for all resources
search	Search across all resource types based on some filter criteria

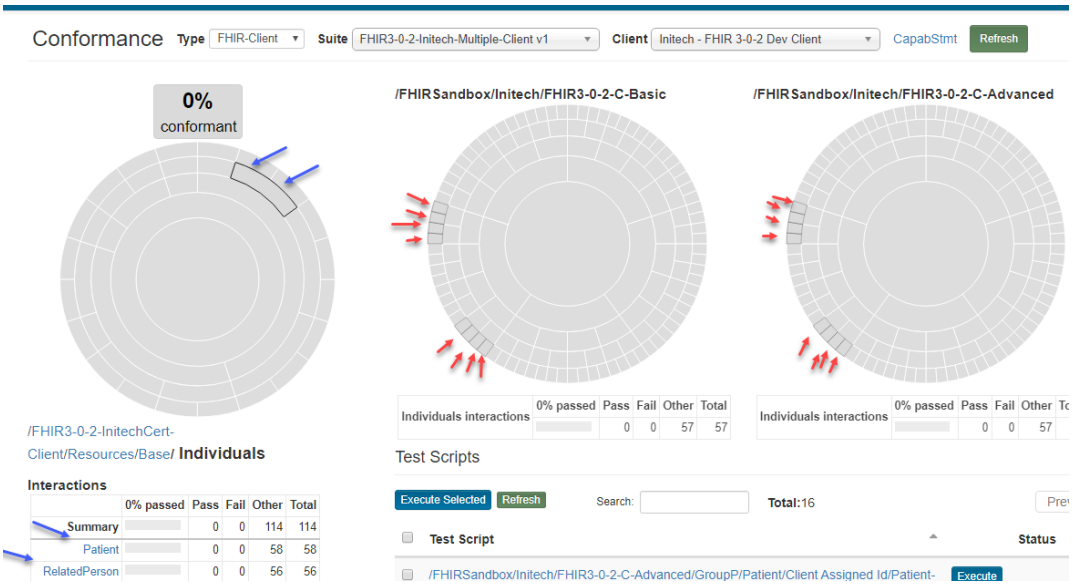
```
<categ name="WholeSystem">
    <categ name="capabilities" path="/capabilities" />
    <categ name="metadata" path="/metadata" />
    <categ name="transaction" path="interaction/transaction" />
    <categ name="batch" path="interaction/batch" />
    <categ name="search-system" path="interaction/search-system" />
    <categ name="history-system" path="interaction/history-system" />
</categ>
```

The **/capabilities** and **/metadata** endpoints are used to retrieve Capabilities Statement from the FHIR system.

Note: If the test groups in a suite do not contain the interactions in a categorization path, the Result Summary chart will not display the category/node even if it's specified in the Categorization. This is true for all types of categorizations.

For example, notice that even though the **Individuals** categorization has many sub-categories (green arrows below) besides **Patient** and **RelatedPerson**, only the **Patient** and **RelatedPerson** nodes display in the Result Summary chart (blue arrows below) because the test groups within the suite only contain operations/interactions for **Patient** and **RelatedPerson**:

```
<categ name="Individuals">
    <categ name="Patient" path="resource/Patient" />
    <categ name="Practitioner" path="resource/Practitioner" />
    <categ name="PractitionerRole" path="resource/PractitionerRole" />
    <categ name="RelatedPerson" path="resource/RelatedPerson" />
    <categ name="Person" path="resource/Person" />
    <categ name="Group" path="resource/Group" />
</categ>
```

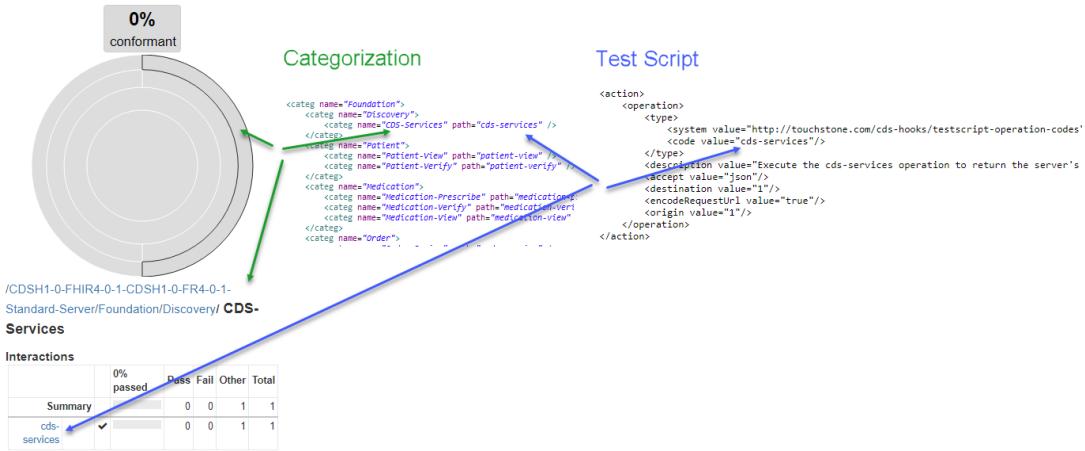


10.3.2.5.2 CDS Hooks

CDS Hooks interactions are not standardized in the specification.

The operation codes in the test script will map directly to the interactions in the Result Summary chart (blue arrows below):

The categories can be arranged in any way. In the example below, the **cds-services** operation/interaction has been assigned to the **CDS-Services** node/category (green arrows below):



10.4 Resource Ownership

An organization must own the **Test Groups** and the **Categorization** that the Conformance Suite uses:

Test Groups

Conformance Suites can be composed of one to three test groups. These test groups need to be owned by the organization that owns the suite.

The organization that's creating the suite and uploading a test group is the one that **owns** the suite and test group, respectively.

This constraint is imposed by Touchstone to avoid unintentional impact on the conformance results of a program when the Test Group owner modifies the test scripts and fixtures in the test group. Such modifications will increment the Conformance Suite version and can drastically change the conformance results of test systems.

Categorizations

Categorization is required for suites that are composed of more than one test group. The Categorization itself needs to be owned by the organization that's creating the suite. This is again to avoid unintentional impact on the conformance results of a program when the Categorization owner modifies the categorization. Such modifications will increment the Conformance Suite version and can drastically change the conformance results of test systems.

Please refer to Categorizations for more information

What about Anchor Systems?

Anchor systems are needed to achieve comparable Conformance results among SUTs. If a suite uses different Anchor Systems, then the results would not be meaningful.

- If the suite is a **Server** suite, then the Anchor System will be a **Client** system that all **Server** SUTs will use for test executions so conformance results would be consistent in the suite.
- If the suite is a **Client** suite, then the Anchor System will be a **Server** system that all **Client** SUTs will use for test executions so conformance results would be consistent in the suite.

Anchor systems must be accessible by the organization that's creating the suite. Although anchor systems do affect the conformance results, Touchstone does not require organizations to own the anchor systems in order to select them as part of their suites. This exception has been made because anchor systems are difficult to provision (as opposed to test groups and categorizations).

It is highly recommended though to use reliable anchor systems for your suites. This is to avoid unintended consequences on the program's conformance testing if the anchor system is deleted or its behavior is modified by the owning organization.

You can learn more about **client** anchor systems *here* and *here*.

You can learn more about **server** anchor systems *here* and *here*.

10.5 Versioning

Conformance Suites are versioned in Touchstone.

This is to make conformance results meaningful when Conformance Suite definitions change over time. One version of a Conformance Suite could be composed of entirely different test scripts from another version. By extension, the Conformance Results for the different versions need to be tracked separately to make the results more meaningful.

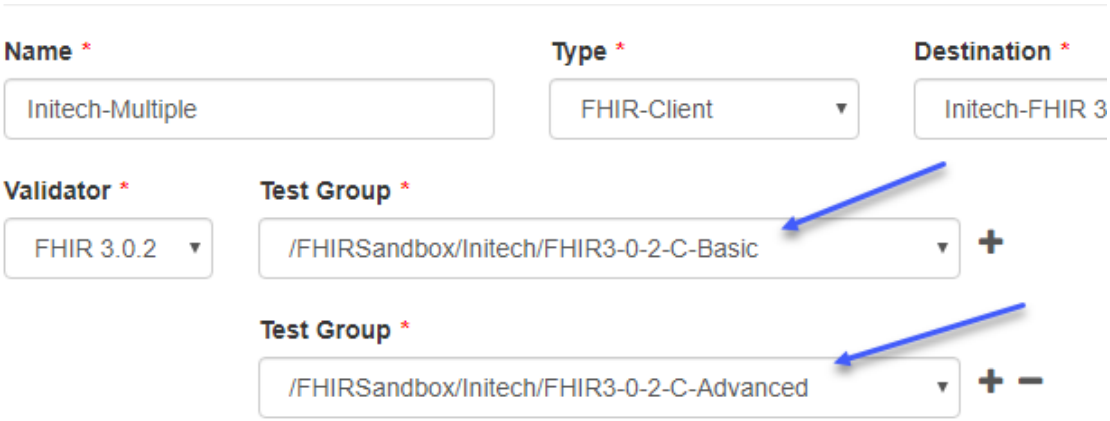
Another reason why Conformance Suites are versioned is to maintain existing Results Summary results (both published and unpublished) when the Conformance Suite definition changes. For example, if the version of a Conformance Suite changes from version 1 to version 2, the **Unpublished** as well as the **Published** results on the Results Summary page for version 1 are maintained.

10.5.1 Version Increments

The following changes to a Conformance Suite cause the version of the suite to increment:

Change in Test Group selections

If test groups are added/removed or a different test group is selected, then that causes the suite version to increment:



Change in Test Group content

If the content of the test groups (that the suite uses) changes then the suite version increments.

The content of a test group changes when:

a. The **content** of one or more test definitions (test scripts, fixtures, or rules) **within** the Test Group **changes**.

b. One or more sub groups or test definitions **within** the Test Group is **deleted**.

For example, consider a suite that uses **/FHIRSandbox/Initech/FHIR3-0-2-Basic** test group. If one or more test definitions whose qualified name starts within '*/FHIRSandbox/Initech/FHIR3-0-2-Basic/*' **changes** or is **deleted**, then that **would increment** the Conformance Suite: That includes, for example:

- '*/FHIRSandbox/Initech/FHIR3-0-2-S-Basic/*'GroupA **sub-group deletion**.
- '*/FHIRSandbox/Initech/FHIR3-0-2-S-Basic/*'GroupA/AllergyIntolerance/Client Assigned Id **sub-group deletion**.
- '*/FHIRSandbox/Initech/FHIR3-0-2-S-Basic/*'GroupA/AllergyIntolerance/_reference/resources/AllergyIntolerance-create-client-id.json **fixture content change**.
- '*/FHIRSandbox/Initech/FHIR3-0-2-S-Basic/*'GroupA/AllergyIntolerance/Client Assigned Id/AllergyIntolerance-client-id-json **testscript content change**.

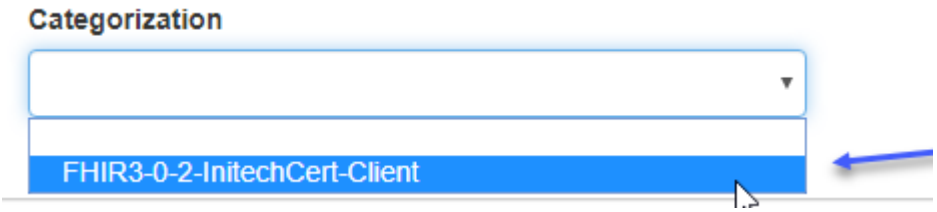
Even though some of the test scripts within '**/FHIRSandbox/Initech/FHIR3-0-2-Basic/**' test group references fixtures or rules in another test group, changes to those fixtures or rules **would NOT increment** the Conformance Suite if their qualified names does not begin with '*/FHIRSandbox/Initech/FHIR3-0-2-Basic/*'. That includes, for example:

- '*/FHIRSandbox/Initech//FHIRCommon/*'_reference/rule/ **sub-group** deletion.
- '*/FHIRSandbox/Initech//FHIRCommon/*'_reference/rule/AssertBodyIfHeader.groovy rule content change.

These test definitions do not start with '*/FHIRSandbox/Initech/FHIR3-0-2-Basic/*' and therefore do not meet the criteria of changes **within** the test group(s) of a suite.

Change to Categorization selection

If you remove, select, or change the Categorization that a suite uses, then the suite version increments:



Change to Categorization content

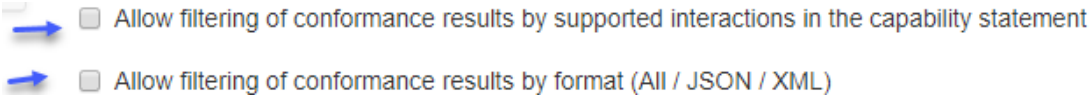
Changes to the Categorization content can change '**% Conformance**' and therefore causes the suite version to increment:

```
<categ name="Foundation">
<categ name="Base">
<categ name="Clinical">
    <categ name="Summary">
        <categ name="AllergyIntolerance" path="resource/AllergyIntolerance" />
        <categ name="AdverseEvent" path="resource/AdverseEvent" />
        <categ name="Condition" path="resource/Condition" />
        <categ name="Procedure" path="resource/Procedure" />
        <categ name="FamilyMemberHistory" path="resource/FamilyMemberHistory" />
        <categ name="ClinicalImpression" path="resource/ClinicalImpression" />
```

We demonstrated the affect of the change above on the suite in *this section*.

Changes to Formats or Supported offering

Changes to these checkboxes do affect '**% Conformance**'. If either flag changes, then the suite version increments:

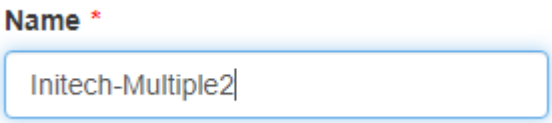


10.5.2 No Version Increments

The following changes to a Conformance Suite do **NOT** cause the suite version to increment as these changes do not cause the '**% Conformance**' to change in the conformance results of a suite.

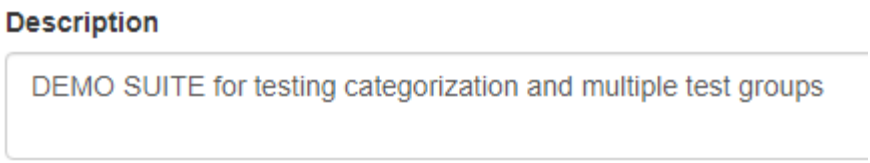
Change to Name of the suite

The **name** of the suite changes across **all** versions of the suite and across all conformance suite results.



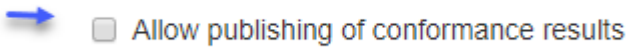
Change to Description of the suite

The **description** of the suite changes for the suite version that the change took place on. But the version does not increment.



Change to Publishable attribute of the suite

The **Publishable** attribute of the suite changes for the suite version that the change took place on. But the version does not increment.



Change to Access attributes of the suite

The **Can be viewed/modified by** attributes of the suite changes for the suite version that the change took place on. But the version does not increment.

10.6 Best Practices

10.6.1 Suite Versioning

It is highly recommended to use a **separate** Conf Suite (inaccessible to users outside your organization) to test changes to the suite definition that would cause the suite version to increment. Events that cause a suite version to increment are covered in Versioning section. Uncontrolled and frequent version increments can be very annoying to users of your suite as Conformance Current page is only accessible for the latst version of a suite.

For example, if you have a Suite called "FHIR4-0-1-Initech-**Cert**-Client" that's the primary suite to be used by end-users and it uses a test group called "FHIR4-0-1-Initech-Basic**Cert**", you can create another suite called "FHIR4-0-1-Initech-**Dev**-Client" that's accessible to fewer users. This **Dev** suite will **not** be referencing the same test group(s) (and categorization) as the **Cert** suite. It would instead use a different test group e.g. "FHIR4-0-1-Initech-Basic**Dev**" that's being actively developed and frequently uploaded to Touchstone (thereby causing many version increments to "FHIR4-0-1-Initech-**Dev**-Client" suite).

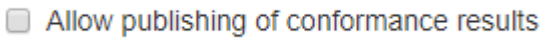
Once "FHIR4-0-1-Initech-Basic**Dev**" test group has reached stability (along with any Categorization you may be using) and you have verified Conformance results of "FHIR4-0-1-Initech-**Dev**-Client" suite, you can upload "FHIR4-0-1-Initech-Basic**Dev**" test group as "FHIR4-0-1-Initech-Basic**Cert**" test group and that will cause only one version increment for the "FHIR4-0-1-Initech-**Cert**-Client" suite.

10.6.2 Anchor Systems

If possible, try to use anchor systems that are either owned by your organization or an organization you can rely upon. Doing so will avoid unintended consequences on the conformance results of large numbers of test systems and users that are relying on suite results to be stable.

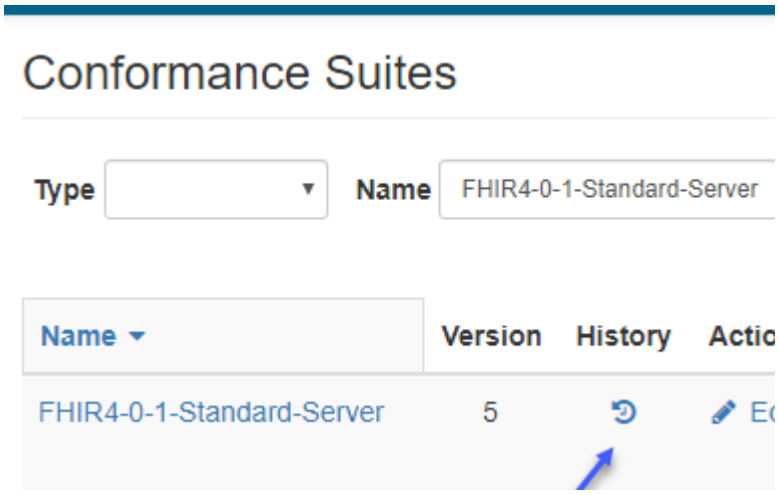
10.6.3 Publishing

You can change the Publishable attribute of a conformance suite at a later time. It's best not to check the box initially when a conformance suite is under active development.



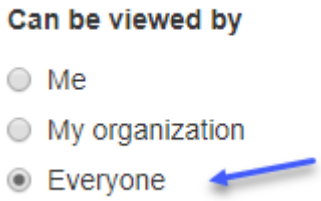
10.6.4 Deletion

Touchstone allows Conformance Suites to be deleted entirely on the Conformance Suite History page. This feature is offered primarily for clean-up purposes when conformance suites are under intense development.



Deletion of a suite version causes permanent deletion of all conformance results (for all test systems) that uses that version of the suite. Such deletion can negatively impact all the organizations that are using your suite.

It is **highly recommended** not to delete conformance suites whose **Can by Viewed by** attribute is set to either **My Org Groups** or **Everyone**.



10.7 FAQ

1. I'm unable to see a test group on Conformance Current page even though I've selected it as one of the test groups in my suite

   In order to be able to properly select test groups that were uploaded prior to Touchstone 5.0.0 release, you must re-upload them to Touchstone.

2. How come conformance suites cannot be deleted by users that have Conf Suite Editor role?

   Conf Suite deletion can have a negative impact on all organizations that use the suite. As such, only Org Reps of the organization that owns the suite can delete the suite.

   This section describes this topic a bit more.

3. How come the anchor system for a suite is not indicated on Conformance Current page

   The **Anchor** system is indicated on Conformance Suites, Result Summary, and various Test Execution screens. The Conformance Current page is crowded as it is. Indicating the **Anchor** system on the screen can confuse the user-selectable SUT with the **Anchor** system. The main focus of a conformance suite is to test SUTs. The **Anchor** system will be the same for all SUTs in a given conformance suite.

CHAPTER

ELEVEN

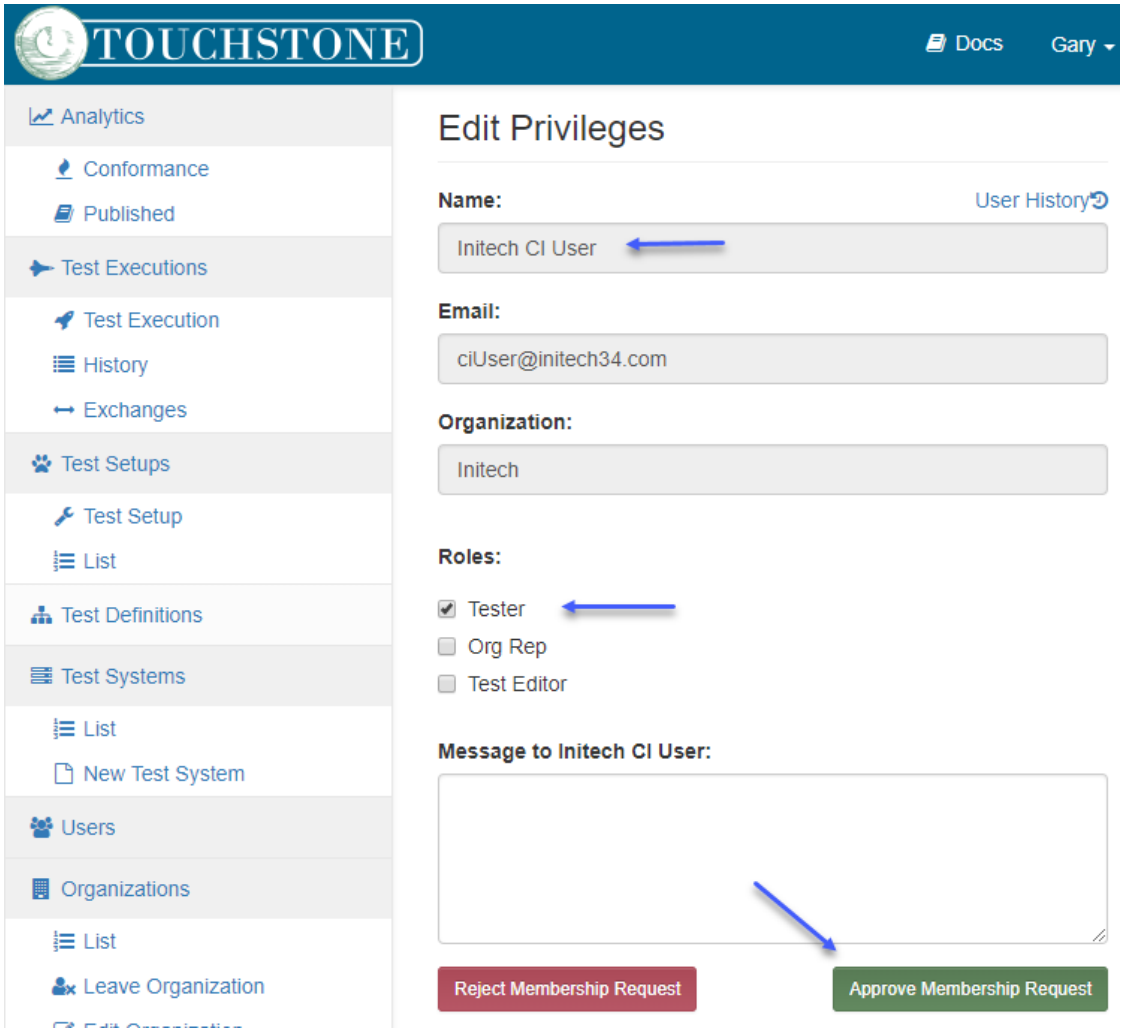
CONTINUOUS INTEGRATION

11.1 API

Test executions can be launched and monitored via remote RESTful web services. This allows for integration of Touchstone test executions as part of your internal automated regressions tests (e.g. CI runs). The XML and JSON schemas for Touchstone API are available within the schemas folder in touchstone-api.zip

We recommend that you use a separate test user account for Touchstone API. Test executions launched on behalf of this test user via Touchstone API will be visible to all members of your organization on the Touchstone UI. So if the Touchstone API does not provide enough details on what went wrong in a test execution, any member of your organization can log in as self on the Touchstone UI and investigate the test failures further.

You can register a test user account within your organization and assign only the `Tester` role to this account. As the Org Rep, when you approve the registration of this test user account, here's where you assign the Tester role:



11.1.1 Authentication

This API call takes the user credentials in the request body and returns an `API-Key` in the response. Alternatively, the user credentials could be passed in the `Authorization` header using Basic Authentication. The `API-Key` value must then be passed in the request header for all subsequent API calls.

URL	https://touchstone.aegis.net/touchstone/**api/authenticate**
Method	POST
Examples	See **goodBasic-authenticateRequest-xxx** and **goodBasic-authenticateResponse-xxx** files within **authenticate** folder in touchstone-api.zip

Before you could call any of the other APIs, you must authenticate with Touchstone. Successful authentication will start a new API "session" with Touchstone. There are no limits on how long an API "session" can last. It is recommended that you re-use the same `API-Key` for all subsequent API calls as re-authenticating would require another trip to the server which will start a new API "session".

11.1.2 Launching Executions

This API call takes the Test Setup name in the request body and returns the id of the test execution (that was launched) in the response. This will be needed in *Retrieve Execution status* API call later.

URL	https://touchstone.aegis.net/touchstone/**api/testExecution**
Method	POST
Examples	See **good-executeTestRequest-xxx** and **good-executeTestResponse-xxx** files within **executeTest** folder in touchstone-api.zip

You should have Test Setups pre-created on the Touchstone UI before launching test executions using Touchstone API. Details on how to do that are covered in the Creating Test Setup section in this guide. It is also recommended to have your Test Executions reach expected results on the Touchstone UI before you integrate them via Touchstone API. The UI offers a lot more navigation and information than Touchstone API.

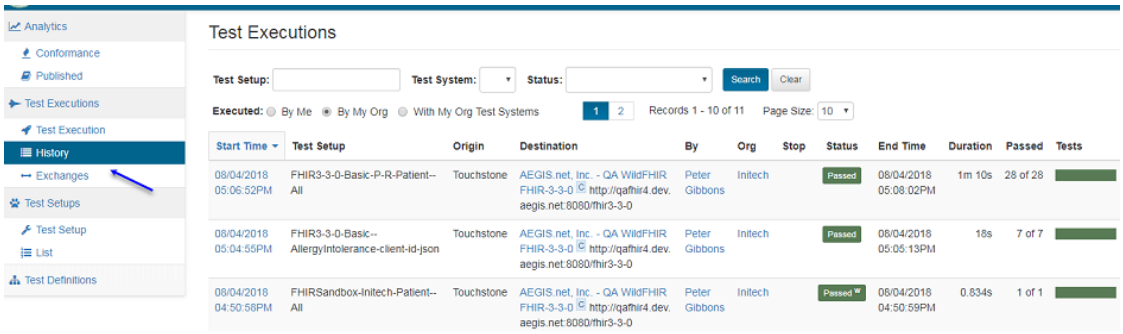
For target test systems that are configured to use OAuth2 tokens, you can refer to the *OAuth2 Static Token* and *OAuth2 Grant Flow* sections for details.

11.1.3 Retrieve Execution status

This API call takes the id of the test execution in the request URL and returns the test execution status.

URL	https://touchstone.aegis.net/touchstone/**api/testExecution**/[testExecId]
Method	GET
Example URL	https://touchstone.aegis.net/touchstone/**api/testExecution**/20160516101258248
Examples	See **good-testExecStatusRequest-xxx** and **good-testExecStatusResponse-xxx** files within **testExecStatus** folder in touchstone-api.zip

You do not need to wait between "**/api/testExecution**" (**POST**) and "**/api/testExecution**" (**GET**) calls. But you do need to wait for 4 seconds between repeated "**/api/testExecution**" (**GET**) calls. The response returned from "**/api/testExecution**" (**GET**) aligns with a row on the following screen in Touchstone UI:

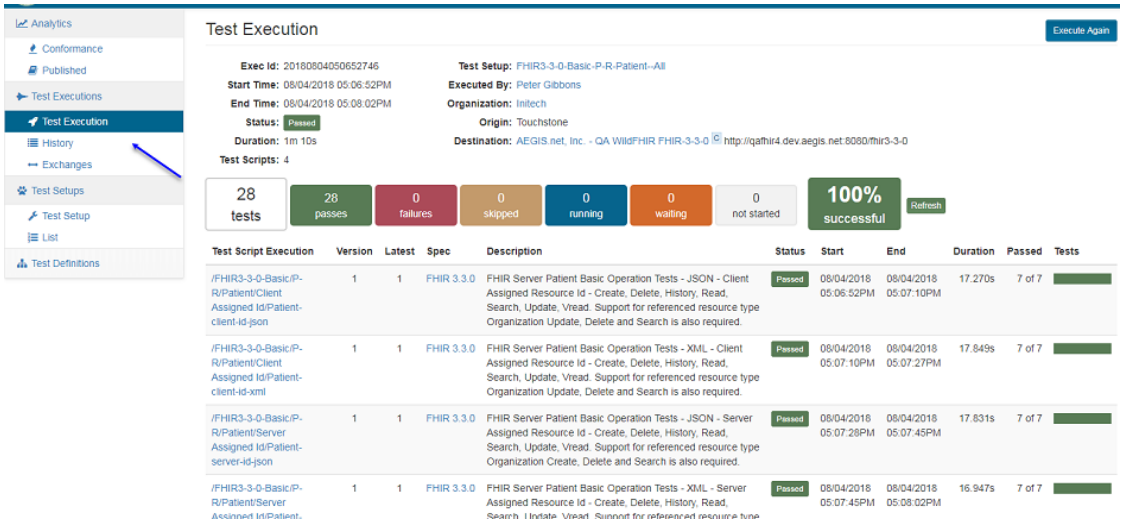


11.1.4 Retrieve Execution Detail

This API call takes the id of the test execution in the request URL and returns the test execution status along with summary status for all the test script executions within the test execution.

URL	https://touchstone.aegis.net/touchstone/**api/testExecDetail**/[testExecId]
Method	GET
Example URL	https://touchstone.aegis.net/touchstone/**api/testExecDetail**/20160516101258248
Examples	See **good-testExecDetailRequest-xxx** and **good-testExecDetailResponse-xxx** files within **testExecDetail** folder in touchstone-api.zip

You do not need to wait between "**/api/testExecution**" (**GET**) and "**/api/testExecDetail**" calls. But you do need to wait for 15 seconds between repeated "**/api/testExecDetail**" calls. The reason for the long wait time is to discourage bypassing of "**/api/testExecution**" (**GET**) calls. You should call "**/api/testExecDetail**" only when the test execution has reached completion. To determine if a test execution has reached completion, you can call "**/api/testExecution**" (**GET**) repeatedly every 4 seconds. Once it has reached completion, you only need to call "**/api/testExecDetail**" once. The response returned from "**/api/testExecDetail**" aligns with the following screen in Touchstone UI:



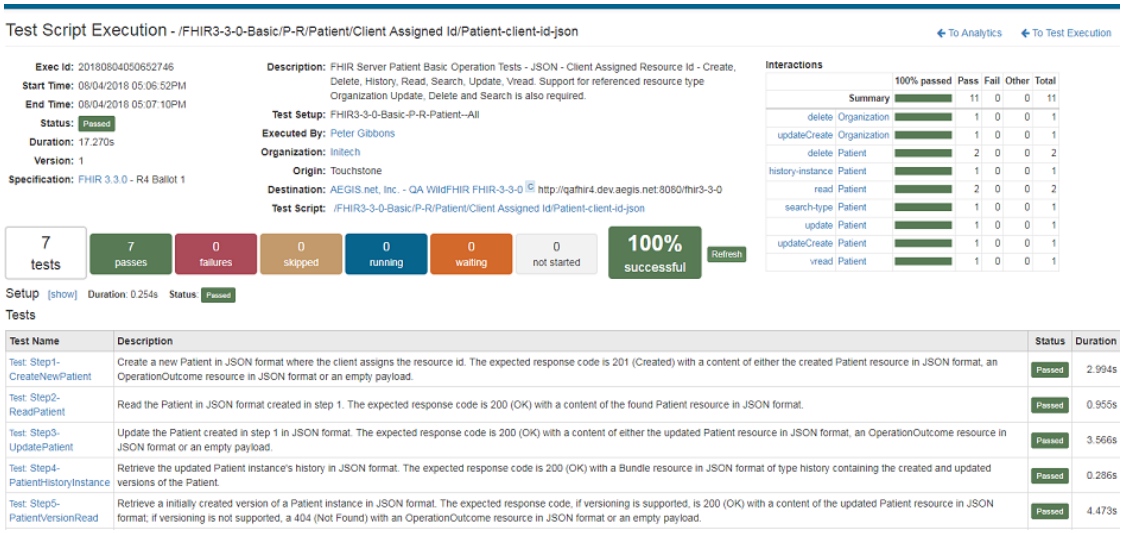
11.1.5 Retrieve Script Exec Detail

This API call takes the id of the test execution in the request URL as well as the name of a test script and returns the script execution details.

URL	https://touchstone.aegis.net/touchstone/**api/scriptExecDetail**/[testExecId]?testscript=[value]
Method	GET
Example URL	https://touchstone.aegis.net/touchstone/**api/scriptExecDetail**/20180507121755387?testscript=/FHIR4-0-1-Basic/A-C/Account/Client Assigned Id/Account-client-id-json
Examples	See **good-scriptExecDetaillRequest-xxx** and **good-scriptExecDetailResponse-xxx** files within **scriptExecDetail** folder in touchstone-api.zip

You do not need to wait between "**/api/testExecDetail**" and "**/api/scriptExecDetail**" calls. But you do need to wait for 4 seconds between repeated "**/api/scriptExecDetail**" calls. Repeated "**/api/scriptExecDetail**" calls would presumably be for different test scripts within the test execution. You should call "**/api/scriptExecDetail**" when the test execution has reached completion (determined by "**/api/testExecution**" GET call) and after calling "**/api/testExecDetail**". The "**/api/testExecDetail**" response will contain the parameters needed to make "**/api/scriptExecDetail**" call

(i.e. the testExecId and the name of the test script you want to dig deeper into). The response returned from "**/api/scriptExecDetail**" call aligns with the following screen in Touchstone UI:



11.1.6 Pseudo code

You do not need to make "**/api/testExecDetail**" and "**/api/scriptExecDetail**" calls when all is well in "**/api/testExecution**" (GET) response. These calls are available when your test execution has some test script execution errors that you expect (because of current capabilities of the test system) and you want to confirm that the errors are coming from the same scripts. Other uses for "**/api/scriptExecDetail**" include confirmation of negative assertion results, operation response times, etc. But these are more advanced use cases. You can start out with getting "**/api/authenticate**" and "**/api/testExecution**" calls working in your environment before integrating "**/api/testExecDetail**" and "**/api/scriptExecDetail**" calls " if needed.

Here's some pseudo-code for launching test execution and getting test execution status:

```
// Authenticate First
AuthenticateResponse authenticateResponse = RestClient.useRelaxedHTTPSValidation()
                                                .headers("Accept: "application/json'")
                                                .url("https://touchstone.aegis.net/
↪touchstone/api/authenticate")
                                                .post("{ "email' : "myOrgCIUser@myOrg.com
↪','password' : "password'}")

// Launch test execution using the API-Key retrieved and a test setup that was pre-
↪created on UI
ExecuteTestResponse executeTestResponse = RestClient.useRelaxedHTTPSValidation()
                                                .headers("API-Key:"+ authenticateResponse.
↪getApiKey()+"Accept: "application/json',
                                                                        Content-Type:
↪"application/json'")
                                                .body("{"testSetup': "MyFavoriteTestSetup'}
↪")
                                                .post("https://touchstone.aegis.net/
↪touchstone/api/testExecution"))

// Retrieve the test execution status in a loop until completion.
```

(continues on next page)

(continued from previous page)

```
TestExecStatusResponse execStatusResponse = RestClient.useRelaxedHTTPSValidation()
                                                        .headers("API-Key:␣
↪authenticateResponse.getApiKey(), "Accept: "application/json'")
                                                        .get("https://touchstone.aegis.net/
↪touchstone/api/ testExecution/"+
                                                                    executeTestResponse.
↪getTestExecId())

while (execStatusResponse.getStatus()==null or execStatusResponse.getStatus()=="Not␣
↪Started" or execStatusResponse.getStatus()=="Running") {
   waitFor(4 seconds)
   execStatusResponse = RestClient.useRelaxedHTTPSValidation()
                        .headers("API-Key: authenticateResponse.getApiKey(), "Accept:
↪"application/json'")
                        .get("https://touchstone.aegis.net/touchstone/api/testExecution/
↪"+ executeTestResponse.getTestExecId())
}
assert execStatusResponse.getStatus()=="Passed"
```

There is a lot more information in the responses. Many more assertions can be performed on the response body and headers. That and the "**/api/testExecDetail**" and "**/api/scriptExecDetail**" calls are left as an exercise for the reader. Once you've had a chance to look into touchstone-api.zip and get the "**/api/authenticate**" and "**/api/testExecution**" calls working in your environment, it should not be difficult to get "**/api/testExecDetail**" and "**/api/scriptExecDetail**" calls working as well. They follow the same paradigm.

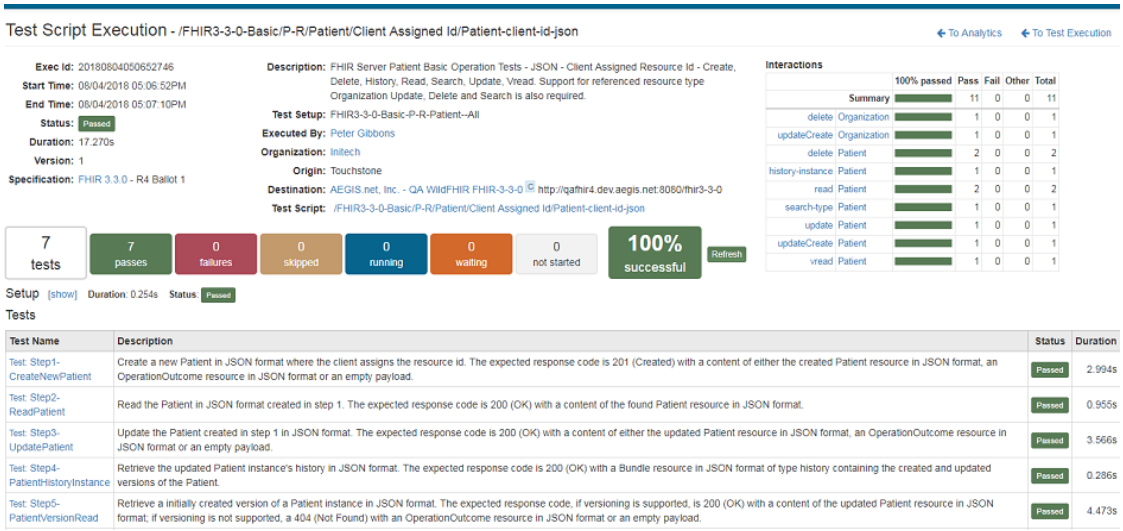
11.1.7 Retrieve FHIR Test Report

This API call ("**/api/testReport**") is available as an alternative to "**/api/scriptExecDetail**". It has a subset of information available in "**/api/scriptExecDetail**" responses. The only advantage of this API is that it returns TestReport which is compliant to the FHIR specification. This API call takes the id of the test execution in the request URL as well as the name of a test script and returns the script execution details.

URL	https://touchstone.aegis.net/touchstone/**api/testReport**/[testExecId]?testscript=[value]
Method	GET
Example URL	https://touchstone.aegis.net/touchstone/**api/testReport**/20180507121755387?testscript=/FHIR4-0-1-Basic/A-C/Account/Client Assigned Id/Account-client-id-json
Examples	See **good-testReport-xxx** and **good-testReport-xxx** files within **testReport** folder in touchstone-api.zip

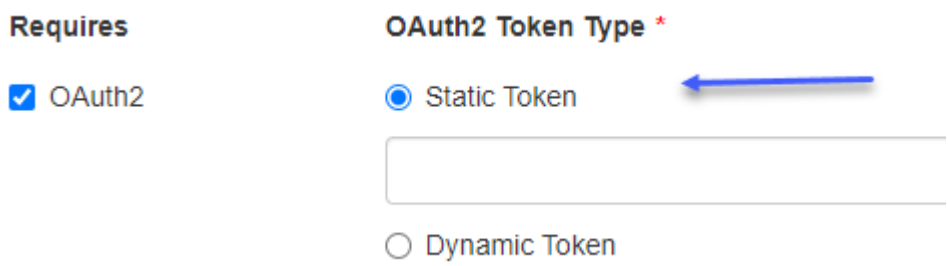
You do not need to wait between "**/api/testExecDetail**" and "**/api/testReport**" calls. But you do need to wait for 4 seconds between repeated "**/api/testReport**" calls. Repeated "**/api/testReport**" calls would presumably be for different test scripts within the test execution. You should call "**/api/testReport**" when the test execution has reached completion (determined by "**/api/testExecution**" GET call) and after calling "**/api/testExecDetail**". The "**/api/testExecDetail**" response will contain the parameters needed to make "**/api/testReport**" call (i.e. the testExecId and the name of the test script you want to dig deeper into). The response returned from "**/api/testReport**" call aligns with the screen below in Touchstone UI:

If "**/api/testReport**" call fails because of a bad request, an OperationOutcome response will be returned.



11.1.8 OAuth2 Static Token

Destination test systems can be configured to support OAuth2 with static OAuth2 tokens:



Touchstone API allows the token that has been configured on Test Setup (on the UI) for such test systems to be overwritten as part of the API call.

URL	https://touchstone.aegis.net/touchstone/**api/testExecution**
Method	POST
Examples	See **good-oauth2TokenexecuteTestRequest-xxx** and **good-oauth2TokenexecuteTestResponse-xxx** files within **executeTest** folder in touchstone-api.zip

As detailed in ExecuteTestRequest, the request payload would normally have the TestSetup name to execute:

```
{
  "testSetup" : "FHIR4-0-1-Basic-P-R-Patient-Client Assigned Id--All"
}
```

To specify the static OAuth2 token for the destination test system as part of the ExecuteTestRequest, the token would be specified along with the full name of the test system:

```
{
  "testSetup" : "FHIR4-0-1-Basic-P-R-Patient-Client Assigned Id--All",
  "destOAuth2Tokens" : [
    {
```

(continues on next page)

(continued from previous page)

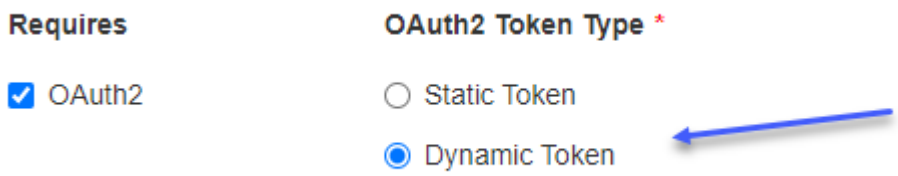
```
      "dest" : "Org1-TestSystemA",
      "token" : "token1"
    },
    {
      "dest" : "Org1-TestSystemB",
      "token" : "token2"
    },
    {
      "dest" : "Org1-TestSystemC",
      "token" : "token3"
    }
  ]
}
```

In the example above, we have three separate destination test systems. In most cases, there will only be one destination involved. The number of destinations is specified in the test script.

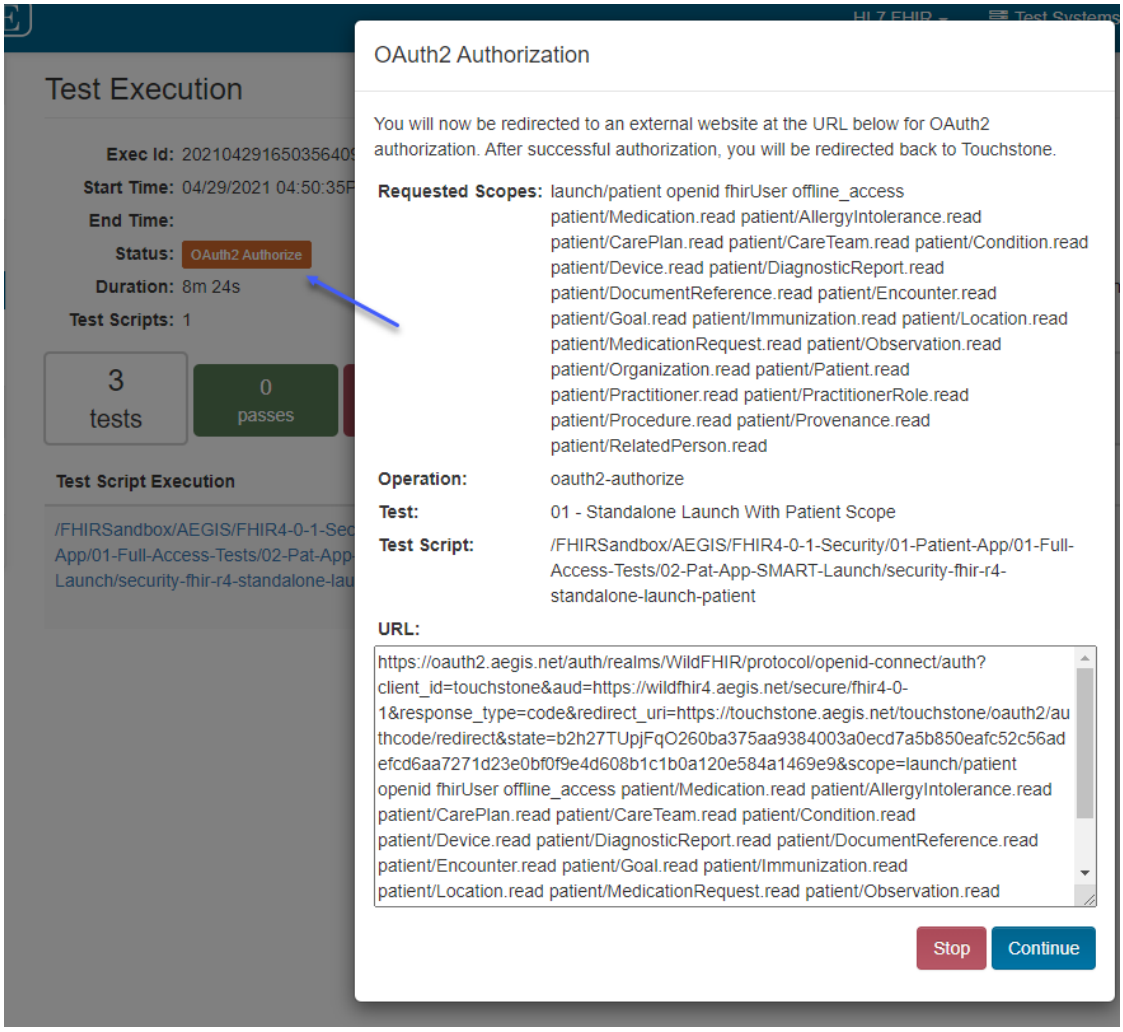
The **dest** value would be [Organization]-[Test System] from Test Systems where [Organization] is the value under **Organization** column and [Test System] is the value under **Test System** column. The **token** value would be the current OAuth2 token for the target test system.

11.1.9 OAuth2 Grant Flow

This section pertains to operation calls that require OAuth2 Authorization for test system servers configured with Dynamic OAuth2:



When test execution reaches such an operation, the status changes to **OAuth2-Authorize** and a popup is displayed on the UI requiring the user to take action:



This section covers how to successfully get past this operation using REST APIs.

In the popup above, the user is required to either cancel or continue to the OAuth2-Authorize URL displayed in the popup. This URL is now returned in the API responses from Touchstone when the test execution has reached OAuth2-Authorize status. The **oauth2AuthzUrl** element value in **GET** "**/api/testExecution**", "**/api/testExecDetail**", and "**/api/scriptExecDetail**" responses will be identical to the OAuth2-Authorize URL value in the popup above.

There are two ways of resuming the test execution using the APIs:

1. Test execution can be launched with **oauth2AuthzApiRedirected** absent or specified as **false** in the payload of "**/api/testExecution**" (**POST**) request:

```
{
  "testSetup" : "FHIRSandbox--security-fhir-r4-standalone-launch-patient",
  "oauth2AuthzApiRedirected": false
}
```

This will require the client to grab the **oauth2AuthzUrl** value from the response to "**/api/testExecution**" (**GET**) call and use that in automated browser (e.g. Selenium) to hit the test system's OAuth2 server and authorize the requested scopes. The end of that process will result in browser redirecting to Touchstone's OAuth2 Redirect URL. This mechanism will require the automated browser (e.g. Selenium) to enter credentials in the Touchstone Login screen (after being redirected) to resume the test execution in Touchstone.

Note that no additional API calls are needed with this mechanism to resume the test execution. The downside is that the client's Selenium browser will need to interact with Touchstone UI (in addition to the test system's OAuth2 server).

2. Alternatively, Test execution can be launched with **oauth2AuthzApiRedirected** specified as **true** in the payload of "**/api/testExecution**" (**POST**) request:

```
{
  "testSetup" : "FHIRSandbox--security-fhir-r4-standalone-launch-patient",
  "oauth2AuthzApiRedirected": true
}
```

This will again require the client to grab the **oauth2AuthzUrl** value from the response to "**/api/testExecution**" (**GET**) call and use that in automated browser (e.g. Selenium) to hit the test system's OAuth2 server and authorize the requested scopes. The end of that process will again result in browser redirecting to Touchstone's OAuth2 Redirect URL. But this time, the browser will not be redirected to the Login screen. It will instead stay on the Redirect URL in the browser to allow the automated test software to grab the content of the OAuth2-Authorize Redirect URL. This URL will contain data that will be needed to resume the test execution.

The OAuth2-Authorize Redirect URL obtained from previous step has to be provided in payload of request to a new API endpoint:

URL	https://touchstone.aegis.net/touchstone/**api/oauth2grant**
Method	POST
Payload	{ "oauth2RedirectUrl" : "[the redirect url in browser after OAuth2 grant]"}

Test execution should resume after that and **GET** "**/api/testExecution**" can be called as usual to get the status of the test execution.

11.2 Jenkins Integration Example

The Pseudo Code example is used as the basis for the following use case where a Jenkins CI server job, which builds a FHIR server, defines a post build task that calls the Touchstone API RESTful web services. The Jenkins Groovy plugin is used where we create an externally called Groovy script that executes the authenticate service, test execution service and iterative status check service calls.

- The installation and configuration of a Jenkins CI server, the corresponding FHIR server build job definition within the Jenkins environment, and installation of the Jenkins Groovy plugin is described in the Jenkins documentation and left as an exercise to the user.
- This use case was successfully implemented within a Jenkins 2.x CI server installed on a CentOS Enterprise Linux server.

11.2.1 Groovy Script Definition

The Groovy script uses the HTTPBuilder class library and corresponding supporting libraries in order to provide the means to call the Touchstone API HTTP / RESTful web service endpoints.

```
Imports
import groovyx.net.http.HTTPBuilder
import static groovyx.net.http.ContentType.XML
import static groovyx.net.http.Method.GET
import static groovyx.net.http.Method.POST
```

11.2.1.1 Authentication

The **body** attribute of the request will contain the Touchstone user account email address and password. These values are to be replaced with the actual values for the given user.

The println statements provide execution output to the Jenkins job console log.

The **apiKey** variable is set for subsequent use in the next web service calls.

If the assert of the apiKey string value evaluates to false, the script will fail and therefore cause the Jenkins job to fail.

```
// Authenticate with the Touchstone Server and get an API Authorization Token
String apiKey
def httpAuth = new HTTPBuilder('http://touchstone.aegis.net/touchstone/api/authenticate')

httpAuth.request(POST, XML) {
    headers.'Accept' = 'application/xml;charset=UTF-8'
    headers.'Content-Type' = 'application/xml;charset=UTF-8'
    body = '<?xml version="1.0" encoding="UTF-8"?><authenticateRequest xmlns="http://
↪touchstone.aegis.net/api"><email>test.user@example.com
    </email><password>password</password></authenticateRequest>'

    response.success = { resp, authenticateResponse ->
        println "POST Response Status : ${resp.statusLine}"
        println "Authenticate Response info : ${authenticateResponse.info}"
        apiKey = "${authenticateResponse.'API-Key'}"
        println "Authenticate Response API-Key : ${apiKey}"
    }

    response.failure = { resp, authenticateResponse ->
        println "Failure Status : ${resp.statusLine}"
        println "Authenticate Response error : ${authenticateResponse.error}"
    }
}

assert apiKey != null : 'Authentication failed! No API-Key returned.'
```

11.2.1.2 Execute Test

Note that the apiKey value is set for the custom HTTP header 'API-Key'.

The **body** attribute of the request will contain the previously defined Touchstone Test Setup name that belongs to the authenticated user. This value is to be replaced with the actual Test Setup name for the given user.

The println statements provide execution output to the Jenkins job console log.

The **execId** variable is set for subsequent use in the next web service calls.

If the assert of the execId string value evaluates to false, the script will fail and therefore cause the Jenkins job to fail.

```
// Execute a Test Setup
String execId
def httpExecute = new HTTPBuilder('http://touchstone.aegis.net/touchstone/api/
↪testExecution')

httpExecute.request(POST, XML) {
```

(continues on next page)

(continued from previous page)

```
    headers.'API-Key' = apiKey
    headers.'Accept' = 'application/xml;charset=UTF-8'
    headers.'Content-Type' = 'application/xml;charset=UTF-8'
    body = '<?xml version="1.0" encoding="UTF-8"?><executeTestRequest xmlns="http://
↪touchstone.aegis.net/api"><testSetup>
    FHIR3-2-0-Connectathon17-Patient-02-Formal-FHIRServer-03-PatientRead--All</testSetup>
↪</executeTestRequest>'

    response.success = { resp, executeTestResponse ->
        println "POST Response Status : ${resp.statusLine}"
        println "Test Execute Response info : ${executeTestResponse.info}"
        println "Test Execute Response exec url : ${executeTestResponse.testExecURL}"
        execId = "${executeTestResponse.testExecId}"
        println "Test Execute Response exec id : ${execId}"
    }

    response.failure = { resp, executeTestResponse ->
        println "Failure Status : ${resp.statusLine}"
        println "Test Execute Response error : ${executeTestResponse.error}"
    }
}

assert execId != null : 'Test Execution not started! No test execution id returned.'
```

11.2.1.3 Wait for Test Execution Completion

Note that the **execId** value is appended to the testExecution web service call path where the httpExecStatus variable is declared.

There is no **body** attribute for this GET request.

The println statements provide execution output to the Jenkins job console log.

The **isDone**, **isPassed** and **execStatus** variables are set and used to determine when to exit the wait while loop.

If none of the loop exit criteria is met, the sleep(15000) call will pause the loop for 15 seconds in order to provide sufficient time to pass and allow more of the test execution to complete. Although this time can be set to a different amount, the recommended minimum time to pause is 5 seconds (or 5000 milliseconds).

If the assert of the execStatus string value evaluates to false, the script will fail and therefore cause the Jenkins job to fail.

```
// Wait for test execution completion
// IMPORTANT - The Touchstone API requires a minimum wait time of 4 seconds between GET␣
↪calls to '/api/testExecution'
boolean isDone = false
boolean isPassed = false
String execStatus
def httpExecStatus = new HTTPBuilder('http://touchstone.aegis.net/touchstone/api/
↪testExecution/' + execId)

while (!isDone) {
```

(continues on next page)

(continued from previous page)

```
    httpExecStatus.request(GET, XML) {
        headers.'API-Key' = apiKey
        headers.'Accept' = 'application/xml;charset=UTF-8'

        response.success = { resp, testExecStatusResponse ->
            println "GET Response Status : ${resp.statusLine}"
            println "Execute Status Response info : ${testExecStatusResponse.info}"
            execStatus = "${testExecStatusResponse.status}"
            println "Execute Status Response status : ${execStatus}"
            println "Execute Status Response period : ${testExecStatusResponse.startTime}
↪ - ${testExecStatusResponse.endTime}"
        }

        response.failure = { resp, executeTestResponse ->
            println "Failure Status : ${resp.statusLine}"
            println "Test Execute Response error : ${executeTestResponse.error}"
            isDone = true
        }
    }

    // Test for all valid execution completion Touchstone statuses
    if (execStatus != null && (execStatus.equals("Passed") || execStatus.equals(
↪"PassedWarning") || execStatus.equals("Failed")
         || execStatus.equals("Warning") || execStatus.equals("Skipped") || execStatus.
↪equals("Stopped")
         || execStatus.equals("Completed"))) {
        isDone = true
        // Test for valid execution passing Touchstone statuses
        if (execStatus.equals("Passed") || execStatus.equals("PassedWarning")) {
            isPassed = true
        }
    }
    else if (isDone) {
        // Catch isDone true before waiting 15 seconds
    }
    else {
        // Wait 15 seconds before checking completion again
        sleep(15000)
    }
}

assert isPassed : 'Test Execution did not pass! Please examine the test results via the␣
↪Touchstone web interface.'
```

11.2.2 Jenkins Job Configuration

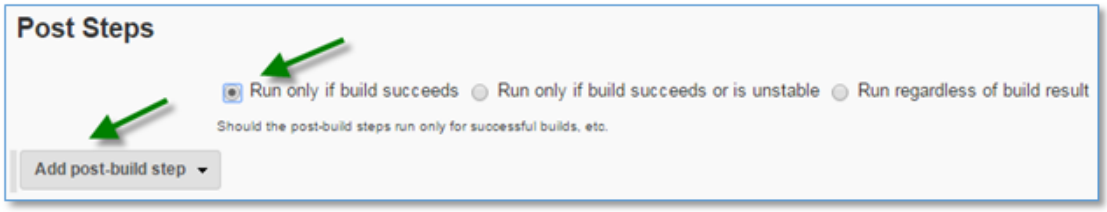
The Jenkins job will be configured to execute the Groovy script in a post steps task. The following sections describe the required entries.

11.2.2.1 Post Steps Initial Configuration

Select the Jenkins job to configure and scroll down toward the bottom of the configuration page to the Post Steps section.

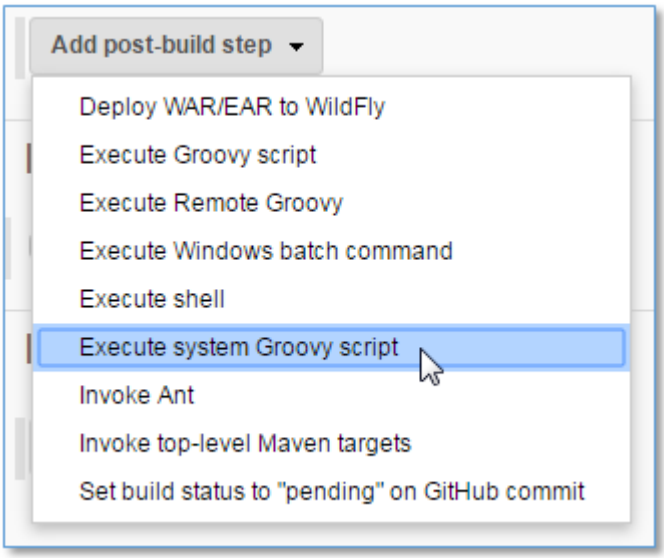
Select the option `Run only if build succeeds` to insure the Touchstone API calls are only triggered after a successful build.

Then select the `Add post-build step` drop down list.



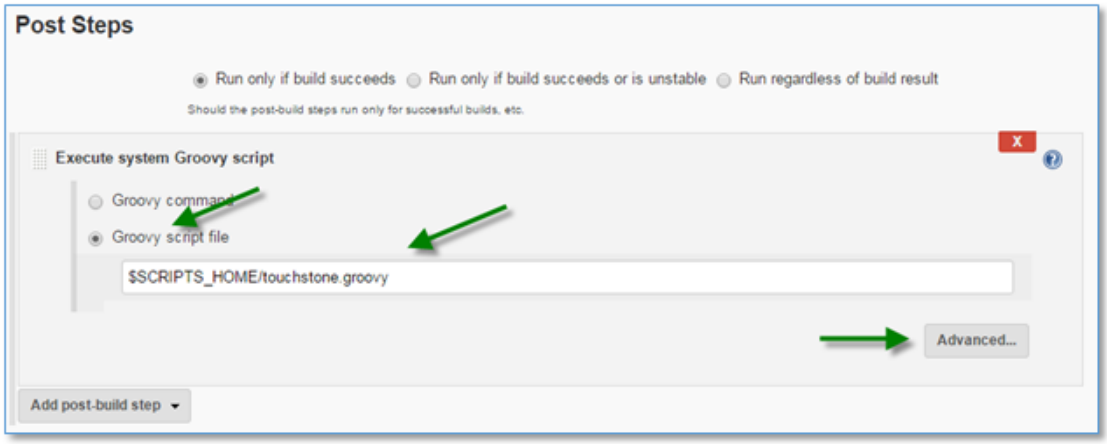
11.2.2.2 Execute system Groovy script

Select `Execute system Groovy script` from the drop down list:



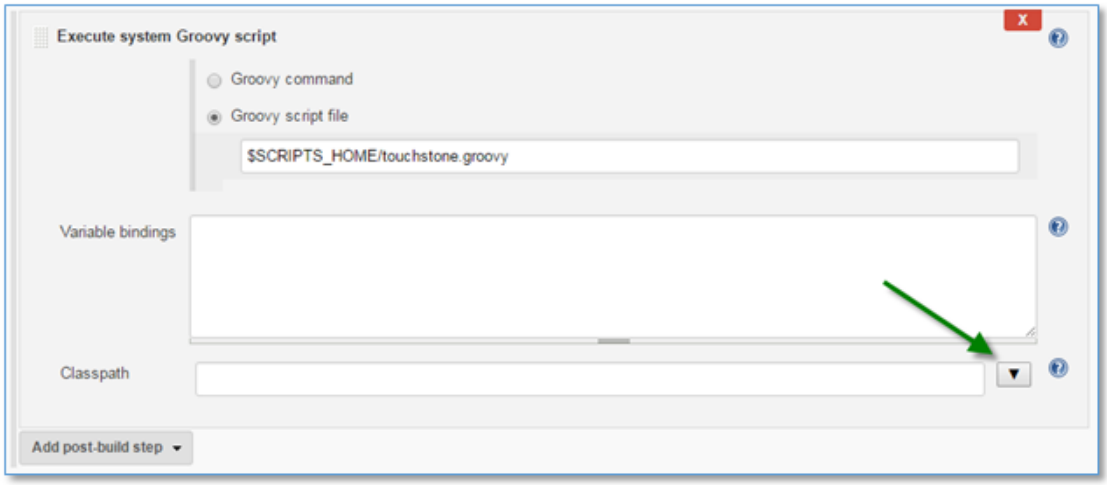
11.2.2.3 Groovy script file

Select the `Groovy script file` option and enter the fully qualified path to the touchstone.groovy script file. Replace the `$SCRIPTS_HOME` variable with the fully qualified directory path. Then select the `Advanced` link.



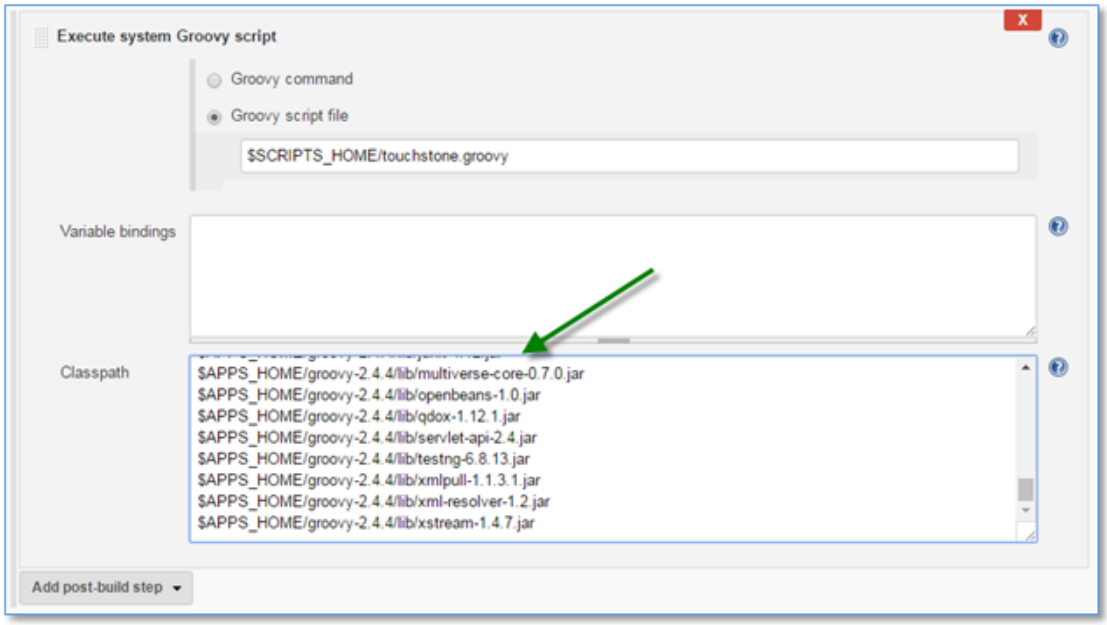
11.2.2.4 Advanced - Expand Classpath

After the `Advanced` section is displayed, expand the Classpath text entry field by selecting the down arrow link to the right.



11.2.2.5 Advanced - Enter Classpath

The Groovy script is executed using an external Groovy SDK and therefore requires all Groovy and supporting libraries be listed in the Classpath field.



The following is a complete listing of the Groovy library jars and additional supporting library jars that are required in order to successfully execute the touchstone.groovy script. Replace the `$APPS_HOME` variable with the fully qualified directory path to the Groovy SDK library files. The highlighted files are supporting jars that were separately downloaded and added to the directory.

- $APPS_HOME/groovy-2.4.4/lib/ant-1.9.4.jar
- $APPS_HOME/groovy-2.4.4/lib/ant-antlr-1.9.4.jar
- $APPS_HOME/groovy-2.4.4/lib/ant-junit-1.9.4.jar
- $APPS_HOME/groovy-2.4.4/lib/ant-launcher-1.9.4.jar
- $APPS_HOME/groovy-2.4.4/lib/bsf-2.4.0.jar
- $APPS_HOME/groovy-2.4.4/lib/commons-cli-1.2.jar
- $APPS_HOME/groovy-2.4.4/lib/commons-collections-3.2.1.jar
- $APPS_HOME/groovy-2.4.4/lib/commons-logging-1.2.jar
- $APPS_HOME/groovy-2.4.4/lib/gpars-1.2.1.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-ant-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-bsf-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-console-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-docgenerator-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-groovydoc-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-groovysh-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-jmx-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-json-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-jsr223-2.4.4.jar

- $APPS_HOME/groovy-2.4.4/lib/groovy-nio-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-servlet-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-sql-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-swing-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-templates-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-test-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-testng-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/groovy-xml-2.4.4.jar
- $APPS_HOME/groovy-2.4.4/lib/hamcrest-core-1.3.jar
- $APPS_HOME/groovy-2.4.4/lib/http-builder-0.7.1.jar
- $APPS_HOME/groovy-2.4.4/lib/httpclient-4.3.6.jar
- $APPS_HOME/groovy-2.4.4/lib/httpcore-4.3.3.jar
- $APPS_HOME/groovy-2.4.4/lib/ivy-2.4.0.jar
- $APPS_HOME/groovy-2.4.4/lib/jansi-1.11.jar
- $APPS_HOME/groovy-2.4.4/lib/jcommander-1.47.jar
- $APPS_HOME/groovy-2.4.4/lib/jline-2.12.jar
- $APPS_HOME/groovy-2.4.4/lib/json-lib-2.4.jar
- $APPS_HOME/groovy-2.4.4/lib/jsp-api-2.0.jar
- $APPS_HOME/groovy-2.4.4/lib/jsr166y-1.7.0.jar
- $APPS_HOME/groovy-2.4.4/lib/junit-4.12.jar
- $APPS_HOME/groovy-2.4.4/lib/multiverse-core-0.7.0.jar
- $APPS_HOME/groovy-2.4.4/lib/openbeans-1.0.jar
- $APPS_HOME/groovy-2.4.4/lib/qdox-1.12.1.jar
- $APPS_HOME/groovy-2.4.4/lib/servlet-api-2.4.jar
- $APPS_HOME/groovy-2.4.4/lib/testng-6.8.13.jar
- $APPS_HOME/groovy-2.4.4/lib/xmlpull-1.1.3.1.jar
- $APPS_HOME/groovy-2.4.4/lib/xml-resolver-1.2.jar
- $APPS_HOME/groovy-2.4.4/lib/xstream-1.4.7.jar

CHAPTER

TWELVE

DOWNLOADS

TestScript Editor (Touchstone IDE)

Comprehensive suite of development tools for creating, managing and publishing TestScript resources.

Version	Release Date	
2.0.0	October 18, 2024	Linux Mac Windows

Touchstone-API zip

Contains schemas and sample messages for using Touchstone API in Continuous Integration.

Version	
5.4.0+	touchstone-api.zip

CHAPTER

THIRTEEN

RELEASE NOTES

13.1 Touchstone

TS 6.4.0 – February 24, 2025

- Update Subscription page to reflect the intention of the Basic subscription level

Bug Fixes

- Corrected errors received when a user attempts to reset their password

TS 6.3.0 – December 23, 2024

Enhancements

- Increased Test Definitions folder name limit to allow up to 40 characters
- Enhanced password security to prevent use of last 20 passwords
- Enhance the user password reset process to allow users to reset their own passwords
- Touchstone execution of oauth2-get-token operation corrected to add PKCE code_verifier value if not present
- Touchstone proxy now supports both TLS v1.2 and v1.3 for secured HTTPS message exchange
- Enhanced Test System functionality to allow user to set a test system as active or inactive
- All Touchstone WildFHIR test systems have been upgraded to use TLS v1.3 for secured HTTPS message exchange

Bug Fixes

- Corrected false error being thrown when sourceId is missing from a create, update and/or patch operation
- Corrected Test Setup window to only show FHIR test systems in the Destination dropdown when testscript.destination is not defined
- Corrected error occurring when changing a Test System defined as FHIR-Client to both FHIR-Client and FHIR-Server
- Corrected issue with Conformance Suites when inactive test scripts become active
- Corrected header assert failure occurring due to case of header name
- Corrected Test Execution status to reflect as Passed with Warning when there are tests that both Pass and Pass with warning

TS 6.2.2 – October 18, 2024

Enhancements

- Enhanced the Touchstone validators updated to HL7 FHIR Core Library v6.3.19

- Enhanced the WildFHIR application dependency updated to HL7 FHIR Core Library v6.3.19
- Enhanced the Touchstone validators to allow for the new FHIR Extensions IG package upload
- Updated the Touchstone validation messages to be prefixed with the source that generated the message
- Updated the WildFHIR client administrative interface for configuration setting maintenance
- Enhanced the FHIRPathEngine 'ofType' and 'as' functions updated to handle conversions between date, dateTime and instant data types
- Updated the Touchstone validation package upload issues to now display in the IG Validation Packages UI

Bug Fixes

- Corrected the FHIRPathEngine to properly follow the defined regex to limit the time milliseconds to 3 digits
- Corrected the validators to make validation terminology errors for unknown code systems reduce to warning messages
- Corrected the WildFHIR date search parameter logic updated to properly handle date and period range comparisons
- Corrected the WildFHIR numeric search parameters to accept decimal values

Maintenance and Security

- Continuity of operations planning included testing the Touchstone environment version rollback procedures

TS 6.2.1 – January 23, 2024

- Update Subscription page to reflect 2024 Subscription and Commercial pricing.

Bug Fixes

- Terminology server errors for an unknown code system are now preserved and generates a validator warning message.

TS 6.2.0 – December 18, 2023

Enhancements

- Implemented a proof of concept (POC) of HL7 Version 2 and HL7 Version 3 testing capabilities. Look for more details in upcoming releases.
- Touchstone Testing IG v2.0.0 released with support for HL7 V2 and V3
- Enhanced the WildFHIR servers to support the use of the fhirVersion mime type attribute
- Enhanced the WildFHIR servers to recognize the FHIR version in the Accept Header and return an OperationOutcome if it doesn't match FHIR version of the WildFHIR instance being tested

Bug Fixes

- Corrected Touchstone to honor a trailing forward slash '/' in endpoint URLs
- Corrected Touchstone to honor encodeRequestURL setting in Peer-To-Peer testing
- Corrected Touchstone to allow users to access Test Setups when the list was empty or searched for a Test Setup that did not exist
- Corrected Touchstone to allow test executions in a 'Waiting' state to be stopped if the related TestScript has been removed
- Corrected Touchstone validation logic to correct duplicate CodeableConcept sub-elements when evaluating profile slicing
- Corrected Touchstone terminology code validation error response handling when code not found in a value set

- Corrected Touchstone FHIR 3.0.2 FHIRPathEngine 'replace()' method to focus element reference
- Corrected Touchstone FHIR 4.0.1 FHIRPathEngine 'dateTime' equivalency when comparing two dateTime values without time zone information
- Corrected Touchstone FHIR 4.0.1 FHIRPathEngine 'exists()' method to correctly handle arguments

TS 6.1.1 – August 16, 2023

Bug Fixes

- Corrected Touchstone to no longer attempt to use Test System's authorization when requesting metadata.
- Corrected the Whitelist FAQ in Touchstone documentation.

TS 6.1.0 – June 27, 2023

Enhancements

- Moved the CDSHSandbox folder to the top of the list of Test definitions.
- Per recent FHIR Terminology Server (tx.fhir.org) update to change the incorrect code display warning messages to error message, the Touchstone Validators have been modified to reset these error messages back to warning messages.

Bug Fixes

- Corrected Touchstone to always run and display asserts in the order that they appear in the TestScript.
- Corrected encoding the hash (#) value in test executions when TestScript operations request encoding.
- Corrected hover text for CDSHooks Test Systems.
- Corrected pulling of CDS Services statements for Test Systems that do not yet have one associated to the Test System setup at test execution.
- Corrected error text when CDSHooks Test System CDS Services do not support the hooks being tested.

TS 6.0.0 – May 1, 2023

Enhancements

- The HL7-FHIR and CDS Hooks domains have been merged, allowing users to have FHIR Tests and CDS-Hooks Tests execute in a single Test Execution, as well as create and run TestScripts with both HL7 FHIR and CDS Hooks operations in the same TestScript or test execution. This enhancement removes the need for the user to specify a "Domain" in Touchstone.
- Implemented support for Order-Dispatch hook for CDS Hooks.
- Provide a read-only view to the Implementation Guide (IG) Packages loaded into the public Validators.
- Test Definitions page updated to display the full qualified Test Definition path selected at the top of the page and then each TestScript display the path after the selected root folder.
- Test Executions page updated to display the assert.description, assert.label, and operation.label when specified in the TestScript.
- Created WildFHIR instances to support CDS Hooks JWT backend authentication for both RS384 and ES384 signing algorithms.
- WildFHIR updated to FHIR R4 support of CDS Hooks order-select and order-sign hooks.
- Implemented validation support for toDate(), toDateTime() and toTime()
- Updated Rule Outputs documentation to provide additional guidance.
- FHIR validator modification to allow for valid base64 encoded contents in payload.

Bug Fixes

- Corrected Conformance Suite behavior so that when certain variables like "${CURRENTDATE}" are used in TestScripts, they render properly in the Suite.
- Conformance Results Summary screen corrected to allow selection of any previous Conformance Suite.
- Corrected CDS-Hooks JWT Assertion to use the CDS Hooks defined interaction when testing with CDS-Hooks systems (see https://cds-hooks.org/specification/current/#security-and-safety).
- Corrected TestScript upload and execution errors when an operation defined uses a rule output as a sourceid (fixture) in a test.
- Corrected WildFHIR implementation of: global search parameters in chained search parameters, Bundle fullURL values when _include used, the 'or' of 'less than or equals' and 'greater than or equals' conditions.
- Corrected validation logic to: Bundle searchset validation of self link to handle search results from extended '$' operations, appropriately ignore TestScript ${variable} declarations, correctly handle extension context.

Notes

- There is a known issue impacting CDS Hooks security enabled Test Systems. Touchstone is not able to automatically download the Services Statement for a secured CDS Hooks Test System.
  - Symptom: The user will get a TestScript execution failure with an error "Capability Statement is required in CDS-Hooks testing. Please download it on the Test System screen."
  - Fix to be implemented in subsequent release: Touchstone will be corrected to appropriately download the Services Statement of a secured CDS Hooks Test System.
  - Workaround: The user can manually upload the Services Statement via the Test System screen using the Upload hyperlink.

TS 5.9.1 – February 2, 2023

Bug Fixes

- Corrected Touchstone parsing of external client request URL to extract the path components - scheme, host, port, path and query - when period characters are present in the path
- Corrected Touchstone to include PKCE flow elements in the Authorization Request when testing non-OAuth TestScripts against a PKCE-Enabled Test System
- Corrected Touchstone logic validating OAuth2 Scopes when the requested and/or supported scopes contain wildcards

TS 5.9.0 - December 9, 2022

Enhancements

- Updated Test System to allow support for PKCE

  > **NOTE:** Specific coding will be required in any TestScript executed against a Test System that is defined with PKCE enabled. Please reference the online Docs for the specifics of what will need to be added to the TestScript(s): (Touchstone Docs PKCE guidence)

Bug Fixes

- Corrected logic when Touchstone performs Oauth Scope comparison where wildcards were used in the scope settings
- Corrected Conformance Suite behavior to properly recognize all TestScripts when TestScripts are added to a suite folder via the Touchstone IDE
- Corrected Vaidator logic when verifying code values against ValueSets that include ValueSet(s) either as a union or an interaction

- Corrected Validator comparison precision handling of fixed date and datetime values
- Corrected WildFHIR processing of conditional create and update search logic to account for the possibility of a returned OperationOutcome in the internal search results
- Corrected WildFHIR search operation with datetime comparison involving time zone
- Corrected WildFHIR operations that return Bundles to populate the Bundle.entry.fullUrl with the full web context path
- Corrected WildFHIR search operation results with multiple sort values
- Corrected WildFHIR search operation query logic for date typed search parameters when the search parameter's corresponding element is defined as a polymorphic typed element with potential date and period data types
- Corrected WildFHIR search operation query logic to perform exact matching with reference data type values

TS 5.8.0 - November 4, 2022

Enhancements

- Updated the MinimumId logic used by Touchstone to match out of order elements (in non-arrays) and to allow for extra element in arrays while preserving array order. TestScripts that previously used MinimumId logic may need to be updated to properly utilize new MinimumId logic
- Enhanced Touchstone to do additional pre-parsing of TestScripts at upload to capture and return any non-XML schema conformant elements
- Enhanced Touchstone Peer-to-Peer TestScript matching process to more accurately match incoming messages to the correct waiting test
- Enhanced Touchstone registration to allow users change the primary email address associated to their user registration and to have an alternate email address that can receive automated emails from Touchstone in addition to their primary email address
- Updated Conformance Suites to allow a Test Group to be associated to an unlimited number of Conformance Suites
- Updated the AEGIS Validators to utilize the updated FHIR Java Core Library (based on HL7 Core Library v5.6.xx). Slight changes in validation messages and validation logic should be expected
- As part of the AEGIS Validator FHIR Java Core Library updates, enhanced Touchstone to properly ignore primitive values containing Testing Platform variable declarations using the ${} delimiters
- Enhanced AEGIS Validator behavior to relax Warning messages to become Informational messages when evaluating referenced resources against ElementDefinitions with multiple targetProfile values where the referenced resource conforms to at least one of the targetProfile entries
- As part of the AEGIS Validator FHIR Java Core Library updates, enhanced Touchstone to properly validate conditional profile definitions on Bulk Data TestScripts
- Enhanced the AEGIS Validator logic allow for a configured selection of which SNOMED-CT code system specific edition of FHIR terminology server is used in validation
- As part of the AEGIS Validator FHIR Java Core Library updates, enhanced the AEGIS WildFHIR instances to utilize the updated validators

Bug Fixes

- Corrected Touchstone user registration to not allow identical email addresses to be used across multiple registrations
- Corrected Conformance Suites Interactions table totals and progress bars to correctly match the conformance results

- Corrected MinimumId logic to correctly process single quotes in XML payload comparisons
- Corrected Conformance Results Summary screen to default to version=All on initial display
- Corrected Conformance Suites to only show active Test Groups in the Test Group selection drop-downs
- Corrected the informational message text to be more clear when a user deletes a Test System Setup
- Corrected the Test System Setup screen to properly display the correct data entry fields when users switch between Oauth Token Types
- Corrected Touchstone to leave the user on the Test Setup screen to allow them to make a correction when the Test Setup name is in error
- Corrected display of User names when navigating between User pages
- Corrected logic and error messaging for fixture variables that do not populate at Testscript upload
- Corrected the WildFHIR server operation response to correctly populate the Bundle.entry.fullUrl of any additonally returned resources

TS 5.7.0 - April 22, 2022

Enhancements

- Subscription page updated to include hover over text for all features and provide additional feature information
- Increased the Base URL length on the New Test System and Edit Test System screens from 128 to 512 characters
- Increased Touchstone session inactivity timeout duration to 30 minutes
- Enhanced validation in the check special code systems logic to allow for ValueSets that reference other ValueSets
- Login process now allows user to login with the email used during registration or a Touchstone generated login based on user name provided during registration. Touchstone generated loginIDs can be found on the User Settings page
- Enhanced HL7 Core Library validator logic to not cache error messages received from a terminology server enabling validation to proceed when there is a communication issue with a terminology server
- Updated the automated email text received when Subscription levels change to notify the user to logout and login to see the Subscription updates applied to their user experience

Bug Fixes

- Corrected exception error encountered on peer to peer testing when ifSupported() executed
- Running tests from Conformance screen no longer requires variables from non-selected tests
- Test Execution no longer uses static authorization headers from previous configurations of the Test System
- Touchstone now allows user to utilize "Execute Again" functionality when a Test Setup fixture name has been changed
- Corrected FHIRPath evaluation errors related to the prioritization of "and"/"in" operators
- Touchstone now ignores search Bundle entries containing an OperationOutcome or with a search.mode = 'outcome' when doing Smart Deletes in Testscript Setup and Teardown operations
- Corrected Conformance Suites starburst graphic to show orange when tests passed with warnings
- Corrected Conformance Suites starbursts to display correct interactions totals for Pass/Warn when hovering over the main starburst
- Corrected state parameter size for OAuth2 Dynamic testing in Conformance Suite execution to match the 100 byte state parameter size in Test Setup execution

- Corrected sorting issue in Conformance Suites which was causing unexpected results as test were not run in the correct order
- Updated validator logic to correctly handle timezones when comparing date and timestamp precision values
- Updated validator logic to correctly handle required bindings for Codeable Concept type elements to ensure that the rules for them are being evaluated correctly:
  - at least one Coding element SHALL be present
  - one of the Coding values SHALL be from the specified value set
  - text can be provided as well, and is always recommended, but is not an acceptable substitute for the required code
- Corrected hover-over text on CapabilityStatement on Test System Configuration to remove references to month and year
- Corrected Touchstone landing page URLs for the Developer's Integration Lab and the AEGIS.net main website so that they directed users to the proper website

TS 5.6.0 - December 10, 2021

Enhancements

- Enhanced test execution warning status display in UI to represent Passed with Warnings as orange in the status bar, include the count of the warnings in the test count summary, and the execution status will represent Passed with Warnings rather than just Passed when there are warnings
- Enhanced navigation in Test Executions with more intuitive hyperlinks and expansions
- Enhanced extractScopesNotGranted() to accept and process a context wildcard
- Implemented FHIRPath $index token support
- Updated Touchstone Testing Implementation Guide for the TestScript profile to include 'history-instance', 'history-type', and 'history-system' operations in FHIRPath expression
- Implemented support for history-instance, history-type and history-system
- Enhanced Validator to data drive the terminology server used during validation
- Enhanced Validator to handle slicing definitions with single and multiple discriminators
- Enhanced Validator timezone processing when comparing date and timestamp precision values
- Added functionality to secured version of WildFHIR to allow page caching
- Added Touchstone IPs for Whitelist to documentation

Bug Fixes

- Added check and warning message to prevent user from uploading a TestScript with the same name but a different extension
- Public Test System no longer requires client secret to be specified
- Fixed "Execute Again" populating incorrect Destination Server
- Resolved issue with auto-pagination incorrectly applying encoding a second time
- Resolved duplicate Authorization Header caused by auto-pagination
- Added logic to prevent test setup execution when no destination specified
- Corrected Touchstone to recognize json extension at test execution
- Implemented validation to prevent a user from executing a test mid-load

- Resolved caching errors caused by unavailable HL7 terminology server

TS 5.5.0 - October 8, 2021

Enhancements

- Added 'nonce' and 'response_mode' parameters
- Added ability to manually upload Capability Statement
- Validator now returns as much of the response as it has validated within the timeout period instead of just returning an error
- Developed extension to support TestSetup Dynamic Fixture such that a user can enter their own fixture at test setup time
- Enhanced Touchstone to allow testing of peer to peer secure systems (OAuth Clients and Servers)
- Added support for history operation, described in Touchstone Testing IG
- Allow organizations to create Test Systems with the same IP address and/or base URL within Touchstone

Bug Fixes

- Unsupported Validators are no longer displayed on User Admin 'Validator' selection box
- Updated the Error message thrown when failing a test setup
- Adjusted heirarchy of importance so that "Skipped" is not displayed over "Passed" or "Passed+W"
- Resolved Unexpected error in Conformance Suite when Toggle to XML is selected
- Fixed Incorrect Handling of Placeholders in requestHeader
- Corrected exception thrown when duplicate tests are run in a Conformance Suite
- Conformance Suites - Passed with "W" now showing at 'Current' view
- Conformance Suite Published result for a previous suite version is now viewable
- Correction to the resolution of references with Bundle entry nesting so that those references are correctly attributed to their respective Bundles
- Validator now recognizes when Code System Content Mode is "Example" the validation can be performed internally and a call to the terminology server for validation is no longer made
- Validator no longer restricts refererence types in standalone validation
- Validator with FHIR Path Engine funcResolve method now accounts for local references within a Bundle
- Updated validator to provide more precise error text when a non-existent code system validation error is encountered
- The validation engine honors the warningOnly = 'true' setting even when there is a Touchstone-related error present

TS 5.4.0 - July 9, 2021

Enhancements

- Touchstone API enhanced to allow passing of a static OAuth token in test execution call
- Automated the Authorization grant flow Oauth login for use via TS API
- Added support for slicing by pattern for FHIR Primitive Data Types
- Updated the slice match by pattern logic to be more lenient when pattern value does not define extension(s)
- Added a pop up message to the "Join Org Group" functionality to provide explanation of test execution visibility

- In support of pagination, the Touchstone Testing Implementation Guide was updated to add new TestScript Extension for operation pagesNext
- Updated WildFHIR capability statement to include support for $document operation
- Increased the Validator & Touchstone timeout thresholds

Bug Fixes

- Addressed drop down sort order to be case insensitive
- Addressed issues with result sets being too large for Touchstone
- Addressed null pointer references
- Addressed pull failure on Capability Statement
- Addressed issue with Validator giving warnings when code values were in the required valueset

TS 5.3.0 - March 26, 2021

Enhancements

- Populate Variables for multiple tests with a single input
- Allow Conformance Suites to be set as 'Inactive'
- Increased Validator timeout parameter
- Increased Rule timeout parameter
- Added filter "waiting for auth" in testscript history

Bug Fixes

- Touchstone doesn't support _include and _revInclude
- Adding IGV Upload Roles & Validators gives Error
- Smart Config not persisted in Test Systems definition
- Some TestScripts unable to be seen on the Conformance page
- Static Token in Test Systems & Test Setup functionality broken

TS 5.2.3 - February 26, 2021

- Maintenance to Subscription Page

TS 5.2.2 - February 15, 2021

Enhancements

- Security settings, DH key size key size must be able to accommodate larger key size of 8192
- Need rule-accessible operation to read restful links

Bug Fixes

- Get 'phantom' Test System Proxy Port has Changed error when certain errors occur in Test Execution
- SourceId declarations in rule asserts not working
- Smart Config unavailable for some testscripts in some circumstances
- Validation slicing logic now accounts for Extension slicing where the extension contains a coded value
- Validator no longer issues Fatal error when ill-formed element is found
- Validator no longer locks up in an Uploading state when it runs into conformance artifacts that cannot be parsed

- Validator resolved "ERROR: This element does not match any known slice for profile" encountered when running CARIN Validation Tests
- Validator giving invalid Warning - "Expansion not defined" when valid values for a valueset are present

TS 5.2.1 - January 11, 2021

Enhancement

- Conformance Suite Description allows for up to 1,000 characters

Bug Fixes

- Conformance Suite Org Group visibility does not save properly - includes ALL org groups
- Test Setup List is loading blank pages
- Test Systems that have been deleted are stopping Orgs from building a new Test System with the same base url
- Getting 'value was null' for SpecEnum error on validation when capability FHIR version is not supported for server
- Oauth2 operations not showing up on Categorizations

TS 5.2.0 - December 14, 2020

Enhancements

- FHIR® Bulk Data Access (Flat FHIR) and SMART Backend Services Support
  - Support for testing of SMART Backend Services conformance
    - Updates to Test System Setup to allow for JWT Assertion and details for registering Touchstone to a SMART Backend Service
  - Support for testing of Bulk Data, including validation of ndjson file contents
  - Updates to the Touchstone Testing IG to aid in Bulk Data ndjson file content validation - https://touchstone.aegis.net/touchstone/fhir/testing/history.html
- Enhancement to allow for Invalid Handshake Security Testing
  - Allows for test launching for invalid handshake testing that does NOT expect a return
  - New 'manual pass' test button for visual inspection and passing of tests
- Enhancement to increase the Touchstone Validator runtime timeout to 90 seconds, allowing for large or complex validations to complete

Bug Fixes

- FHIRPath exists(criteria : expression) support corrected
- Slicing logic correction; added logic to slice matches for discriminator of type 'type' when multiple types are defined
- Correct UTF8 encoded character handling
- Message Bundle reverse references validated properly
- 'Test System Proxy Port has Changed' error when proxy port has not actually changed is no longer displayed on Test Execution results

TS 5.1.0 - October 29, 2020

Enhancements

- OAuth2 and SMART-on-FHIR Support Test Support

  - Dynamic authorization for both the OAuth2 Authorization Code and the Client Credentials flows
  - Dynamic retrieval of Smart Configuration from SMART-enabled test systems in Touchstone
  - Test Support for SMART-on-FHIR Discovery, OpenID Connect (OIDC)
  - Ability to perform Stand-Alone Launch and EHR-Launch testing
  - Enhanced TestScript Authoring to include OAuth2 capabilities (Refer to the Touchstone IG)
  - OAuth2 authorizations support in explicit form via new TestScript operations: oauth2-authorize, oauth2-get-token, oauth2-refresh-token, and oauth2-revoke-token
- Enhancement to allow for Assertion-Only Tests
- Enhancement to allow Tests to continue when Test Assertions fail
- New FHIR4-0-1-Security test scripts to support SMART-on-FHIR testing
- New section in TestScript Authoring Guide that describes the OAuth2 functionality

TS 5.0.0 - June 26, 2020

Enhancements

- **Multi-Profile Validator**
  - Enabling testing against an existing profile and a newer version of that profile for the same FHIR version.
  - Touchstone will allow for different validators to exist for a FHIR Version, be associated to a TestScript at upload, be selected for a test setup at runtime.
  - Users with appropriate authority will be able to upload IG Validation Packages, upload testscripts and associate them to one or more validator packages.
  - Please refer to Multi-Profile Testing for more information.
- **Conformance Suites**
  - Enabling organizations to build their own certification program and easily see results of systems who test against it.
  - Conformance Suites can be defined by an organization, can include only the tests needed for certification, can automatically build conformance reports.
  - Conformance Suites are versioned so that organizations can know which version of the suite they tested against and so that an organization knows which version of the suite organizations are certified against.
  - Users can see and select the Conformance Suite they want to certify against.
  - Testers and Organizations can easily and instantly see conformance in both a graphical and tabular view.
  - Please refer to Conformance Testing and Conformance Suite Authoring for more information.

TS 4.7.4 - Nov 8, 2019

Bug Fixes

- Request URL assertion fails when request parameter contains multiple values for a given parameter with the AND condition.
- Touchstone does not allow POST requests without a request body.
- Some special characters (e.g. '/' and '+') are not being encoded in request query parameters.
- HTML entity references get resolved in request URLs on Test Script Execution screen. This makes it difficult to compare what was submitted to what was expected in Peer-to-Peer tests.

TS 4.7.3 - Sept 6, 2019

Enhancement

- Add support for FHIR Connectathon 22 in Analytics/Conformance.

TS 4.7.2 - August 30, 2019

Bug Fixes

- Presence of "_format" parameter on its own in Peer-to-Peer request causes test failure.
- Touchstone appends forward slash ('/') to request URLs before query parameters.
- User gets error when accessing Org Groups page before logging in.

TS 4.7.1 - August 05, 2019

Bug Fixes

- Peer-to-Peer 'requestURL' assertion is not performing exact match on URL path.
- Peer-to-Peer 'requestURL' assertion fails when query parameter contains multiple values in different order from actual request.

TS 4.7.0 - July 29, 2019

Enhancements

- Improve the performance of Groovy rule assertions in Rules Engine.
- Inform user of incorrect proxy port in submitted URL in peer-to-peer testing.
- Security Improvements.

Bug Fix

- User gets 'Unexpected error' if Test Setup is executed without waiting for page to completely load.

TS 4.6.1 - July 01, 2019

Bug Fixes

- Users are blocked from using the same Origin IP address for IP-only peer-to-peer testing at the Connectathon.
- Two users within the same organization that have marked their client test systems for IP-matching are not prevented from launching test execution at the same time in peer-to-peer testing even when the test systems share the same IP address.
- Touchstone does not inform user on Test Script Execution screen if proxy port of a destination test system changed after the execution was launched in peer-to-peer testing.
- Touchstone inserts request URL assertion for peer-to-peer tests even when resource id in URL is dynamically generated on client test system in peer-to-peer testing. This assertion will always fail.
- Previous user agreement version getting displayed to user even after its updated in database.
- Navigating to a page on Test systems list screen throws error if first test system name on list has unencoded special characters.
- Touchstone is not evaluating variables in assertion FHIR expressions.

TS 4.6.0 - May 29, 2019

Enhancements

- Improve support for IP-only Peer-to-Peer testing. Refer to Peer-to-Peer Testing for details.

TS 4.5.1 - May 24, 2019

Bug Fix

- User getting security violation if multiple browser tabs are used to sign into Touchstone.

TS 4.5.0 - May 23, 2019

Enhancements

- Add support for CORS in Touchstone API.
- Remove Touchstone-specific USER_KEY and ORG_KEY headers before forwarding request to destination server in Peer-to-Peer exchanges.
- Add warning message on Test Setup when Test System specification does not match Test Definition specification.
- Security improvements.

Bug Fix

- Touchstone sends POST to Bundle endpoint instead of Base for batch operations.

TS 4.4.6 - May 05, 2019

Bug Fix

- Test Definition upload allows 'Viewable By' access to get widened for child test groups via Org Groups.

TS 4.4.5 - Apr 02, 2019

Enhancement

- Support for FHIR 4.0.0 testing in FHIR Connectathon 21.

TS 4.4.4 - Mar 21, 2019

Bug Fix

- User getting error when attempting to access expected request message in peer-to-peer test.

TS 4.4.3 - Mar 21, 2019

Bug Fixes

- User getting error when attempting to download the Touchstone IDE.
- Email notifications being sent without subject line.
- Request URL assertion should work with or without forward slash before query string.

TS 4.4.2 - Mar 18, 2019

Bug Fix

- User getting error when attempting to edit a Test Group.

TS 4.4.1 - Mar 17, 2019

Bug Fix

- User getting error when attempting to delete Test Setup.
- Clicking on 'CapabStmt' link on Analytics/Conformance or Published screens throws error.

TS 4.4.0 - Mar 13, 2019

Enhancements

- Validate email authenticity for new user registrations.

- Automatically stop test executions that have been in Waiting status for long period of time.
- Security improvements.

TS 4.3.6 - Mar 03, 2019

Bug Fixes

- Org Reps that are also Org Group Reps cannot assign the Test Editor role to an org user.
- Paging to last page on Exchanges screen can produce UI error if more exchanges took place during the request.

TS 4.3.5 - Feb 06, 2019

Bug Fix

- Exchange attributes not getting resolved correctly for test systems that have been deleted and recreated.

TS 4.3.4 - Jan 25, 2019

Bug Fixes

- Navigation to resolved rule contents within an assertion on "Test Script Execution" screen causes unexpected error.
- Navigation to test setup that had destination missing causes unexpected error.

TS 4.3.3 - Jan 15, 2019

Bug Fix

- Navigation to previous Test Setup throws error if Test System used in the setup got deleted and one or more test scripts in the setup used variables.

TS 4.3.2 - Jan 14, 2019

Enhancement

- Avoid appending default ports (80 for HTTP and 443 for HTTPS) for 'Host' header in Peer-to-Peer requests.

Bug Fix

- Test Script Execution and Exchanges screens reporting incorrect 'Host' header in Peer-to-Peer requests.

TS 4.3.1 - Jan 10, 2019

Bug Fixes

- Edit Test System screen redirects user to Sign-In screen with error when test system name is changed to a deleted test system name within the organization.
- When Test Script Execution and its resource screens are accessed without signing in, the user is not redirected to those screens after signing in.

TS 4.3.0 - Jan 9, 2019

Enhancements

- Support for FHIR 4.0.0. See http://www.hl7.org/fhir/directory.cfml and http://hl7.org/fhir/R4.
- Allow test script authors to deactivate test groups to reduce clutter on Test Definitions tree.
- Resolve placeholders in variable default value during test setup.
- Offer explicit Approve and Reject buttons for membership requests on "Edit Privileges" screen.
- Support for HTTP Method in Test Script operation. See http://build.fhir.org/testscript-definitions.html#TestScript.setup.action.operation.method
- Security improvements.

- Section 508 improvements on Subscriptions screen.

Bug Fix

- User gets error when Test Script or Fixture links are accessed from Test Script Execution and newer versions of the resources have restricted access.

TS 4.2.2 - Dec 26, 2018

Bug Fixes

- Request URL assertion fails in Peer-to-Peer tests when '_format' parameter is present.
- Operation counts are miscalculated on TestScript Execution screen when request assertion fails in Peer-to-Peer testscripts.

TS 4.2.1 - Dec 16, 2018

Bug Fix

- Users can bypass duplicate hostname checks on Edit Test System screen by deleting and recreating a test system.

TS 4.2.0 - Nov 26, 2018

Enhancements

- Add support for exclusions of test definitions during upload, parsing, and validations. Refer to Docs for details.
- Improved support for CDS-Hooks testing.
- Section 508 improvements on Test Setup and Test Execution screens.

Bug Fixes

- FHIR profile validation fails in CDSH domain when invoked from Groovy rules.
- Conditional delete should not require Bundle.total on internal search response.

TS 4.1.0 - Oct 18, 2018

Enhancements

- Add support for HTTPS in Touchstone Proxy for SSL/TLS peer-to-peer exchanges.
- Support relative 'DATE' and 'DATETIME' placeholder in fixtures, request payloads, and URLs.
- Allow INVALID floater (that displays testscript and fixture validation errors) to stick so user can copy the messages.
- Break up validation error and warning messages into separate line items on Test Script Execution screen.
- Support Media types (MIME types) in TestScript operation 'accept' and 'contentType' elements.
- Redirect legacy TS Release Notes URL to new HTML version URL.

TS 4.0.2 - Sep 28, 2018

Bug Fixes

- Error message on Exchanges screen is misleading when submitted USER_KEY is incorrect.
- Rules Engine does not handle a capability statement with an invalid FHIR spec version.

TS 4.0.1 - Sep 26, 2018

Bug Fixes

- Upload of sub groups removes other sub groups from Analytics/Conformance screen in non-Sandbox test groups.
- Analytics/Conformance screen errors if user's organization has not created any test systems.

TS 4.0.0 - Sep 24, 2018

Enhancements

- Support for CDS Hooks 1.0. See https://cds-hooks.hl7.org/1.0.
- Support for FHIR 3.5.0. See http://hl7.org/fhir/directory.html and http://hl7.org/fhir/2018Sep/index.html.

Bug Fix

- Users with Pending registration status get unexpected error when they navigate to My Placeholders.

TS 3.9.3 - August 20, 2018

Enhancement

- Performance improvements to response time of the Exchanges screen.

Bug Fix

- Trailing space in testscript name causes Test Setup to fail.

TS 3.9.2 - August 6, 2018

Enhancement

- Rebranding of Touchstone UI.

Bug Fix

- Changing values on MySettings page and then switching to Test Definitions page causes error.

TS 3.9.1 - July 24, 2018

Enhancement

- Allow invalid fixtures to get skipped during scheduled validation-runs for negative testing.

Bug Fixes

- Validation-errors permanently disappear on some test definitions that haven't changed after an upload.
- Validation-errors panel gets clipped on Test Definitions page when user is on lower screen resolutions.
- User gets "No invalid fixtures" immediately upon upload without any indication of a pending validation-run.

TS 3.9.0 - July 23, 2018

Enhancements

- Validate Sandbox test definitions on a scheduled basis.
  - Touchstone checks for new and modified test definitions every 5 minutes and validates them using FHIR Validator.
  - Validation errors become visible to owners of the Sandbox test group on Test Definitions screen e.g. Only members of 'Organization0005' will be able to view validation errors under '/FHIRSandbox/Organization0005' test group.
- Remove subscription requirement for the following read-only Touchstone APIs:
  - Retrieve Execution Status
  - Retrieve Execution Detail
  - Retrieve Script Exec Detail
  - Retrieve FHIR Test Report
- New Rules Authoring documentation and clean up of the Rules API.

Bug Fix

- Some links in Touchstone email notifications point to the old Touchstone user guide.

TS 3.8.4 - July 10, 2018

Bug Fix

- Special characters are not handled properly in RequestURL assertions for Peer-to-Peer tests.

TS 3.8.3 - June 17, 2018

Enhancement

- Allow Org Reps of subscribing organizations to control who can upload test scripts. See Upload on UI for more details.

Bug Fix

- When user signs in after attempt to download touchstone-api.zip, the user is redirected to Test Definitions instead of Downloads page.

TS 3.8.2 - May 29, 2018

Enhancement

- Allow non-subscribers to execute FHIRSandbox test scripts.

TS 3.8.1 - May 28, 2018

Enhancement

- Disallow uploads of test scripts that contain dot in the name.

Bug fix

- Upload of test group with wider access to test group with Org Group restrictions causes unexpected error.

TS 3.8.0 - May 13, 2018

Enhancements

- TestScript Editor release
  - To learn more about it, visit TestScript Editor.
  - It can be downloaded from TestScript Editor Download.
- Online HTML-based Documentation
  - Can be accessed at Touchstone Docs.
- Add placeholder support for generated UUID values.

TS 3.7.0 - April 23, 2018

Enhancements

- Support for FHIR 3.3.0. See http://hl7.org/fhir/directory.html and http://hl7.org/fhir/2018May/index.html
- Test Result Publishing
  - New Analytics/Published screen.
  - Organizations at the Project subscription level and above can now publish their test results to make them publicly viewable.
- Performance improvements to response time of the Test Definitions screen.
- Ability for Groovy rule writers to raise arbitrary warnings and errors in TestScript execution assertions.

- Warning message on Analytics/Conformance screen when Interaction Counts are reset.

Bug fix

- Org short name change does not propagate to test definitions

TS 3.6.2 - Jan 25, 2018

Bug fixes

- The extended operation '$validate-code' shows up under '$validate' band on Analytics/Conformance screen.
- The TestScript and HL7 FHIR links on home page point to old github URLs.

TS 3.6.1 - Jan 22, 2018

Enhancement

- Add support for $versions operation. See http://wiki.hl7.org/index.php?title=201801_Versioned_API

TS 3.6.0 - Jan 14, 2018

Enhancements

- Add support for FHIR 3.2.0. See http://hl7.org/fhir/directory.html and http://hl7.org/fhir/2018Jan/index.html
- Third-party library upgrades.

TS 3.5.3 - Dec 13, 2017

Enhancement

- Allow deletion of Org Groups and edits to their names.

TS 3.5.2 - Dec 04, 2017

Enhancements

- Allow Org Group members to filter by "My Org Groups" on the Test Executions and Exchanges screens.
- Allow Org Group Reps to set the visibility of member-organization test-executions to "Org Group (Open)" or "Org (Private)". Organizations that are part of "Org Group (Open)" org groups can view each other's test executions. Those that are part of "Org (Private)" cannot. Org Group Reps can view test executions of member organizations regardless of the flag setting.
- Allow Touchstone to be selected as the default FHIR Client in peer-to-peer test scripts. This allows for the same test scripts to be used for both client-server (peer-to-peer) testing as well as server-only testing.
- Allow Test System owners to bypass origin IP checks in peer-to-peer testing.

Bug fix

- Users that are not signed-in can view test scripts that have been uploaded with access of "Can be viewed by My organization".

TS 3.5.1 - Nov 21, 2017

Enhancements

- Improved email notifications during registration and other events.

TS 3.5.0 - Nov 12, 2017

Enhancements

- Subscription services. You can learn more at the new Subscription page.
- Users can upload Test Scripts to Touchstone under the Starter plan.
- Improvements to operation paths on Analytics/Conformance screen.

TS 3.4.16 - Sept 27, 2017

Bug fix

- Status-count meter is broken on Test Execution and Analytics/Conformance screens. This issue can be seen on latest Chrome release (Version 61.0.3163.100).

TS 3.4.15 - Sept 11, 2017

Bug fix

- In Peer-to-Peer exchanges (client-side testing), Touchstone forwards requests to the old Base URL of a test system when the test system Base URL changes.

TS 3.4.14 - Sept 05, 2017

Enhancements

- Improve error reporting in Setup Deletes when test system returns invalid JSON response during test execution.
- Improve error reporting when variables cannot be resolved during test execution.

TS 3.4.13 - Sept 03, 2017

Enhancements

- Improve error reporting when conformance statement cannot be parsed.
- Improve error reporting when response from Test System cannot be parsed.

TS 3.4.12 - July 31, 2017

Enhancement

- Add support for different sets of Test Groups for the same FHIR version across different months on Analytics/Conformance screen.

Bug fix

- Default Accept and Content-Type headers are created alongside explicit Accept and Content-Type request headers. This issue arose in new test scripts that define Accept and Content-Type headers explicitly using <requestHeader> element in TestScript operation definition and not the usual <accept> and <contentType> elements.
- Test System formats field is not updated in Touchstone based on actual CapabilityStatement format field value after CapabilityStatement download

TS 3.4.11 - July 19, 2017

Enhancement

- Improve error reporting when Bundle total is non-numeric in search response during Setup.

Bug fix

- Do not display response links on Exchanges screen if a response has not been received by Touchstone.

TS 3.4.10 - July 10, 2017

Enhancements

- Add support for GraphQL operation in Touchstone. See http://build.fhir.org/graphql.html and http://graphql.org/ for details.
- Add support for FHIRPath PATCH in Touchstone. See http://build.fhir.org/http.html#patch and http://build.fhir.org/fhirpatch.html for details.
- Exclude external resources referenced in HTML responses from Peer-to-Peer interceptions on Exchanges screen.

TS 3.4.9 - July 2, 2017

Enhancements

- Reject Peer-to-Peer exchanges originating from web crawlers e.g. GoogleBot.
- Validate actual origin of the peer-to-peer request against Origin test system in Test Setup.
- Do not capture CORS Pre-Flight OPTIONS request in peer-to-peer message exchanges.

Bug fix

- Interaction counts on TestScript Execution screen get reset after each Peer-to-Peer exchange.

TS 3.4.8 - June 21, 2017

Bug fix

- Touchstone is not handling versionId in FHIR resource when it's specified in wrong format.

TS 3.4.7 - May 28, 2017

Enhancements

- Change "Conformance Statement" to "Capability Statement" where appropriate in Touchstone.
- Improve error reporting when invalid operator value is used for confOperator in rule templates.

Bug fixes

- Supported flag is incorrectly set on some of the main sunburst bands on Analytics/Conformance screen.
- Message on parent of outermost band is misleading on Analytics/Conformance screen when some but not all interactions are supported.
- Rule parameter values are not properly listed on Ruleset popup.
- Clicking on Ruleset link causes error if ruleset uses same rule more than once.

TS 3.4.6 - May 4, 2017

Enhancement

- Allow users to filter for failures on Test Execution and TestScript Execution screens. This is helpful when the number of test scripts and tests is large in the run.

Bug fix

- Incorrect icon/flag is displayed for unsupported bands in main sunburst on Analytics/Conformance screen.

TS 3.4.5 - May 3, 2017

Bug fix

- The sunbursts on Analytics/Conformance screen shift a little when user hovers over them. This makes it harder for user to click on a particular band.

TS 3.4.4 - May 2, 2017

Enhancement

- Add support for FHIR 3.0.1. See http://hl7.org/fhir/directory.html and http://hl7.org/fhir/STU3/index.html

Bug fixes

- Patch interaction shows up as supported on Analytics/Conformance screen even though capability statement does not declare support for it.
- Search param '_include' at the resource level shows up as unsupported on Analytics/Conformance screen even though capability statement declares support for it.

TS 3.4.3 - April 14, 2017

Bug fix

- Upgraded Groovy to latest version (2.4.10). This got rid of memory leak during TestScript rule execution. See https://issues.apache.org/jira/browse/GROOVY-7683 for details.

TS 3.4.2 - March 22, 2017

Enhancement

- Add support for FHIR 3.0.0. See http://hl7.org/fhir/directory.html and http://hl7.org/fhir/STU3/index.html

Bug fix

- Formats-Supported checkboxes are misaligned in Firefox on New/Edit Test System screens.

TS 3.4.1 - March 1, 2017

Enhancement

- Test Groups renamed
  - Top-level test groups under Test Definitions are now listed with latest version at the top.
  - Basic test scripts have been broken up into alphabetical groupings to allow for future growth and easier navigation.

TS 3.4.0 - Feb 17, 2017

Enhancement

- Assertion dependencies
  - Touchstone can now conditionally evaluate assertions based on the state of message headers/content as well as support in the test system's conformance statement.

TS 3.3.8 - Jan 14, 2017

Bug fixes

- Conformance statement parsing fails when conformance statement contains lists with empty items.
- Test Systems screen shows no items on some pages.

TS 3.3.7 - Jan 13, 2017

Enhancement

- Msc improvements to UI response times.

TS 3.3.6 - Jan 12, 2017

Bug fix

- Extended-Operation interactions are not matched correctly against the Conformance Statement. Even though conformance statement declares e.g. "'reference' : 'OperationDefinition/ValueSet-validate-code'", Touchstone expects "'reference' : 'OperationDefinition/valueset-validate-code'", and flags the interaction as unsupported. Either reference will now work.

TS 3.3.5 - Jan 11, 2017

Enhancements

- Support for FHIR TestReport. Users can now retrieve test results via Touchstone API in FHIR TestReport format. Details can be found in the UserGuide.

- Peer-to-Peer Testing: Touchstone should not auto-retrieve conformance statements during test execution and once a day for test systems that are marked as purely client test systems. Users should still have the option of manually downloading the conformance statement on the UI.
- Peer-to-Peer Testing: Client request-submission attributes should be more prominent on the Script Execution screen.
- Peer-to-Peer Testing: Default checks should be made during test execution for HTTP method and URL submission by the client so bad submits can be caught upstream and reported to user.

TS 3.3.4 - Jan 05, 2017

Bug fix

- Analytics/Conformance screen throws error when number of test scripts exceeds 256 for a given specification.

TS 3.3.3 - Jan 03, 2017

Enhancement

- Support for SSL- cert communication between Touchstone and Test Systems.

TS 3.3.2 - Dec 29, 2016

Bug fixes

- Resource profile links are broken for relative references on Conformance/Capability statement popup.
- Navigating to XML view on Conformance/Capability statement popup still displays JSON content.

TS 3.3.1 - Dec 26, 2016

Enhancements

- Add system-generation assertion for Bundle/total presence in system-generated search response in Setup.
- Display more descriptive text on the UI when Touchstone is unable to connect to target test system.

Bug fixes

- Assertion for Content-Type does not check for correct FHIR mime-type.
- Detect infinite loop when deletes fail for system-generated search/delete operations in Setup.

TS 3.3.0 - Dec 20, 2016

Enhancements

- Add support for FHIRPath evaluation when used in TestScript "variable.expression", "assertion.expression", and "assertion.compareToSourceExpression" elements.

  The Advanced-FHIR1-8-0, Basic-FHIR1-8-0, and Connectathon14 test scripts use FHIRPath expressions for some assertion evaluations.

  Details can be found at http://hl7.org/fhirpath/index.html

  Touchstone continues to support "variable.path", "assertion. path", and "assertion. path" elements where built-in XPath/JSONPath is executed.

TS 3.2.5 - Dec 19, 2016

Bug fixes

- System errors when user attempts to update the access token on Test Setup screen (instead of Test System screen).

TS 3.2.4 - Dec 13, 2016

Enhancements

- Add support for FHIR 1.8.0.
- Attempt more mime-types during Conformance Statement retrieval. Update Test System "Spec" and "Formats-Supported" fields in Touchstone based on values in retrieved Conformance Statement.

Bug fixes

- Required check for Destination field fails in Safari on TestSetup screen causing test execution to fail.
- Test Scripts table does not show up on Analytics/Conformance screen in Safari.

TS 3.2.3 - Dec 03, 2016

Enhancements

- Allow users to see the state of their existing conformance statements in Touchstone on the Test Systems, Test System, and Edit Test System screens without having to download every time.
- Keep search field on Analytics/Conformance screen visible at all resolutions.

Bug fixes

- Peer-to-Peer Test Execution matching does not work by IP address.
- Interaction counts do not get updated during test execution in peer-to-peer exchange when source fixture is missing.

TS 3.2.2 - Dec 01, 2016

Enhancements

- Add placeholder support for relative date and date time computations and replacements in fixtures.

Bug fixes

- Do not allow users outside an organization to delete a test system even if the test system is publicly editable.
- Monthly dropdown on Analytics/Conformance screen does not show Sept 2016.

TS 3.2.1 - Nov 29, 2016

Enhancements

- On a daily basis, Touchstone will pull conformance statement for all test systems that have checked "Allow Touchstone to pull conformance statement on scheduled basis" on Test System screen. Touchstone will also attempt to pull conformance statement during test execution if previous attempts had failed.
- Enforce timeout in TestScript operation calls.

TS 3.2.0 - Nov 28, 2016

Enhancements

- Ability for users to delete test system even if it has test executions.
- Ability for users to change test system name, base URL, and specification even if the test system has test executions.
- Ability for two organizations to choose the same test system name. Display Organization in TestSetup test system drop-down.
- New filter option on Test Executions screen that shows all test executions against the organization's test systems including those launched by users in other organizations.
- Ability to download Conformance statement for the month within Analytics/Conformance screen.
- Display link to Conformance statement on Analytics/Conformance screen.
- Show test systems for the signed-in user's organization on Test Systems screen by default.

- Conformance icon on Test Execution screen indicates one or more test scripts have unsupported interactions but screen does not indicate which test script in the list is the culprit. User has to click on each test script execution to find out. Test Execution screen now shows which test scripts in the list have unsupported interactions.
- Display icon for unsupported interactions in orange instead of red on Test Execution and Script Execution screens.

Bug fixes

- Clicking on XML on Conformance statement popup retrieves JSON Conformance statement. It should retrieve XML Conformance statement.
- Conformance floater overflows page if conformance statement supports over 7 formats.

TS 3.1.0 - Nov 14, 2016

Enhancements

- Improve error handling and reporting during Test Script upload by Test Editors.
- Add support for http://build.fhir.org/testscript-definitions.html#TestScript.setup.action.operation.requestId .
- Improve handling of non-existent elements during JSON path evaluation in TestScript assertions.

Bug fixes

- Execution fails for test script that does not contribute towards Analytics/Conformance.
- Request parameters are being decoded twice which has caused errors during paging of some TestSetups.

TS 3.0.7 - Oct 18, 2016

Enhancements

- Improve error reporting when minimum check fails for unexpected element type.

Bug fixes

- Addition of Other column on Analytics/Conformance screen is causing overlap of tables
- ConditionalDelete in Setup fires unnecessary Search operations for JSON.
- When user selects unsupported Domains (e.g. NwHIN instead of HL7 FHIR), Analytics/Conformance screen throws error.

TS 3.0.6 - Oct 10, 2016

Enhancements

- Conformance statements are validated after download and validation errors are displayed on the conformance popup.
- Resolve variables in minimum fixtures.

Bug fixes

- TestScript setup operations/interactions are not counted correctly against the conformance analytics.
- Assertions against header values should be case-insensitive.
- TestScript popup is not handled properly when session has expired.

TS 3.0.5 - Oct 04, 2016

Enhancements

- Add 'Other' column on Analytics/Conformance and Script Execution screens that captures the rest of the interaction counts besides Pass and Fail.

Bug fixes

- Background color for C icon for conformance statements on Test Execution and Test Execution list screens is inconsistent with C icon on Script Execution screen.

TS 3.0.4 - Oct 02, 2016

Enhancements

- Add support for Basic Authentication in Touchstone API.

TS 3.0.3 - Sept 29, 2016

Enhancements

- Exclude entire test script from "% conformant" if any interaction is unsupported and user selects "Exclude Unsupported". This is needed because of dependencies between tests/operations with a TestScript.
- Add support for conditionDelete/single, conditionalDelete/multiple, updateCreate, and conditionalRead to conformance-based testing.
- Move Test Systems link to the top menu so it's visible at all times.
- New set of options for execute access on New Test System and Edit Test system screen allows separation of view-access from execute-access. Test systems that use access tokens can have execute-access restricted without affecting view-access.
- Admin screens for maintenance.

TS 3.0.2 - Sept 18, 2016

Bug fix

- HTML returned in HTTP header values throws off the ScriptExecution and Message detail screens.

TS 3.0.1 - Sept 17, 2016

Enhancements

- Allow users to access test executions by other users from Analytics/Conformance screen if target system is owned by user's org.

Bug fixes

- Search in test assertions for headers in HTTP requests and responses should be case-insensitive based on HTTP spec.
- System throws error when user has not defined any test systems yet and attempts to go to Stats/Conformance screen.
- Popups for request and response message detail lost their CSS styles in 3.0.0

TS 3.0.0 - Sept 15, 2016

Enhancements

- Conformance-based Testing
  - New Analytics/Conformance screen allows users to monitor the conformance of their tests systems to a specification based on test scripts available in Touchstone.
  - Tests can be executed from within this new screen.
  - Details can be found in the Touchstone User Guide here.
- Improve default TestSetup name generation.
- Allow Access Token to get set on Test System Edit screen for test systems that require OAuth2.

- Inform users of interactions that are supported (or unsupported) by a test system's conformance statement on Script Execution screen.

TS 2.4.8 - Sept 11, 2016

Enhancement

- Add support for STU3 Ballot+ FHIR Mime Type ('application/fhir+json' and 'application/fhir+xml')

TS 2.4.7 - Sept 05, 2016

Enhancement

- Change Test Group names to better align with FHIR specification versions.

TS 2.4.6 - Aug 12, 2016

Enhancement

- Added support for FHIR STU3 Ballot.

TS 2.4.5 - July 07, 2016

Bug fix

- Test engine producing errors with negative testing of update operation.

TS 2.4.4 - June 26, 2016

Bug fixes

- Upload of TestScript sub-folder does not overwrite existing test scripts.
- Header value in TestScript-operation that contains a colon will not appear on Script Execution screen.
- Table columns on TestScript-Execution screen widen and then shrink on screen refresh.

TS 2.4.3 - June 18, 2016

Enhancements

- Add support for verifying request URL contents.
  - This is useful in Client-side/Peer-to-Peer testing.
- Do not set 'Content-Type' and 'Accept' headers in request when TestScript has explicitly specified 'none' for operation contentType and accept.
  - This is useful in verifying test-system behavior when these headers have not been set.

TS 2.4.2 - June 13, 2016

Bug fix

- Test Engine ignores response assertions when request assertions are present in peer-to-peer operations.

TS 2.4.1 - June 07, 2016

Bug fix

- Upload of fixtures-only sub-directory by Test Editor causes Test Definitions to become inaccessible.

TS 2.4.0 - May 23, 2016

Enhancements

- Touchstone API

  - Test executions can be launched and monitored via remote RESTful web services. This allows for integration of Touchstone test executions as part of your internal automated regressions tests (e.g. as part of Continuous Integration).
  - The XML and JSON schemas for Touchstone API are available within the schemas folder in touchstone-api.zip.
  - Details can be found in the Touchstone User Guide here.
  - Details on CI integration with Jenkins can be found here.
- CAPTCHA during registration to prevent robot registrations.
- Software upgrades

TS 2.3.2 - May 06, 2016

Bug fix

- Clicking on History for a test definition on Test Definitions screen throws error.

TS 2.3.1 - May 05, 2016

Enhancements

- Added XML-Patch support.

TS 2.3.0 - April 20, 2016

Enhancements

- Rules Engine
  - Complex rules against message headers and body (that go beyond what TestScript assert previously supported) can now be evaluated using the new Rules Engine.
  - These rules can be used via the new TestScript.rule, TestScript.ruleset, TestScript assert.rule, and TestScript assert.ruleset elements.
  - Rules can be implemented in the following languages:
    * Groovy
    * Schematron
    * XSLT

      Support for additional languages will be added in the future.
  - Test editors can create, upload, and edit rule and ruleset definitions on the UI.
  - Access to rule and rule definitions is controlled the same way that access to test scripts and fixtures are controlled (by user, organization, and organization-group).

Bug fix

- Fixed XXE vulnerabilities.

TS 2.2.3 - April 13, 2016

Bug fix

- Assertion in TestScript.setup system-generated delete operation is checking for response code 200. It should also check for 204.

TS 2.2.2 - April 11, 2016

Enhancements

- Rename "FHIR DSTU 2.2" to "FHIR-STU-3-Candidate"

TS 2.2.1 - April 01, 2016

Enhancements

- Add support for FHIR DSTU 2.2.

TS 2.2.0 - March 24, 2016

Enhancements

- Test Script and Fixture Versioning
  - When the content of an existing test script or fixture changes (via an upload), the test execution screens will inform the user that newer version of the test script and/or fixture is available.
  - The system will continue to pull in latest test scripts and fixtures on test re-execution.
  - Old test scripts and fixtures (before versioning became available) will assume the version 0 in old test executions.
- Capture Test System conformance statement for each test execution
  - End-users can now see the conformance statement of each test system involved in test execution at the time of execution on Test Execution screens. Until now, Touchstone only made the latest conformance statements available to end-users on Test System screens.
- Improvements in conformance checks and display of conformance statements.
  - Some FHIR servers can only return conformance statements in one format. Touchstone now attempts to retrieve conformance statements first in JSON and then in XML if JSON fails. In the past, Touchstone only attempted XML conformance statement retrieval during conformance checks.
- Validations per specification version.
  - Profile and resource validations during execution of test script FHIR spec 2.1 will be performed against FHIR spec 2.1 Validator. Similarly, profile and resource validations in 2.0 test script executions will be performed against FHIR spec 2.0 Validator, etc. In the past Touchstone validated only against latest FHIR validator.
    - FHIR spec 2.0 corresponds to FHIR DSTU 2.0 (v1.0.2-7202) - DSTU 2 (Official version) with 1 technical errata (Permanent home); http://hl7.org/fhir/DSTU2/index.html .
    - FHIR spec 2.1 corresponds to FHIR DSTU 2.1 (v1.2.0-7447) - Connectathon 11 Snapshot (temporary); http://hl7.org/fhir/2016Jan/index.html .

TS 2.1.9 - March 11, 2016

Bug fix

- Concurrent test script execution can cause operation execution data collisions.

TS 2.1.8 - February 28, 2016

Bug fixes

- XML Minimum assertion evaluation failure when minimum file contains attribute with resource name.
- JSON Minimum assertion evaluation failure for resource "List" which has nested entry elements within Bundle.

TS 2.1.7 - February 27, 2016

Bug fix

- Test Definition tree expands and contracts on page load.

TS 2.1.6 - February 25, 2016

Bug fix

- Have system-generated search and delete operations in Setup and Teardown honor the Accept header specified in the delete operation.

TS 2.1.5 - February 25, 2016

Enhancements

- New Landing Page for guest users with following information:
  - What Touchstone is.
  - Links to tutorials.
  - Software updates.
  - Feeds.

    Users can bypass the landing page by bookmarking the "Sign In" page. Users continue to get redirected to their last set of Test Definitions after signing in. You can always get to the Landing Page by clicking on the Touchstone logo.

TS 2.1.4 - February 21, 2016

Enhancements

- Org Group membership
  - Admins can now make users the Org Group Rep of Org Groups.
  - Org Reps can request their organizations to become members of Org Groups.
  - Org Group Rep can approve or reject such requests.

    Access to resources such as Test Definitions and Test Systems can be limited to a group of organizations via Org Group functionality.

TS 2.1.3 - February 08, 2016

Enhancements

- Display the org groups that a user and organization belong to on the appropriate screens.
- Capture changes to Organization name and to Test System name and baseUrl in User History.

Bug fixes

- Bad baseURL throws off Proxy URL generation during Test System creation
- 'No test setups' is displayed twice when a user initially lands on 'Test Setup' screen.

TS 2.1.2 - February 06, 2016

Bug fixes

- XPath functions are not always taken into account in XPath evaluation during test execution

TS 2.1.1 - February 04, 2016

Bug fixes

- Multiple variables in URL path not getting resolved properly in some cases during operation execution.
- System does not handle relative references to fixtures (in test scripts) that contain '..'.

TS 2.1.0 - February 03, 2016

Enhancements

- Allow Org Reps to view User History for members of the organization. The following events are captured:
  - User registers
  - User requests membership
  - User membership gets approved
  - User membership gets rejected
  - User membership gets changed back to Pending
  - User roles change
  - User creates an organization
  - User cancels a membership request
  - User status get deactivated by admin
  - User status gets reactivated by admin
  - User gets locked because of too many password attempts
  - User gets locked manually by admin
  - User gets unlocked by admin
  - User resets his/her password
  - User password gets reset by org rep or admin
  - User regenerates the Org Key for his/her organization
  - User regenerates his/her User Key
- Support for Organization Groups
  - Admins can now create organization groups. An organization can belong to multiple organization groups. Test scripts, fixtures, and test systems can now be tied to organization groups. This allows members of multiple organizations to share resources and control access across specific organizations.
- Redirect user to Test Definitions if an old test script has been removed.
- After sign in, redirect org reps to users screen if they have pending registrations.

Bug fixes

- System errors when test setup is deleted in dual session and executed in another.

TS 2.0.4 - January 25, 2016

Enhancements

- Allow users to switch organizations more easily.
- Improve error reporting in test script SETUP execution when entries cannot be found.
- Search criteria not included in message to user when 0 records returned in Test Definitions search.

Bug fixes

- Old sort cookie value is not being handled by Touchstone. This causes error when some users land on Test Definitions screen.
- Touchstone attempts to parse FHIR resource out of HTML response (when operation errors).

TS 2.0.3 - January 10, 2016

Enhancements

- Add support for FHIR transaction operation

Bug fixes

- Resource type assertions are failing when operation response contains BOM characters.
- The presence of bad html in the operation response throws off the page's styles.

TS 2.0.2 - January 08, 2016

Enhancements

- Improve Test Setup default name generation.
- Email other org reps when one org rep approves or rejects user registration.
- Indicate the expected Content-Type and Accept headers for client-side operations to help users submit the right request the first-time.

Bug fixes

- If the test scripts referred to by a Test Setup have all be removed, then system throws misleading error when user tries to re-execute the Test Setup.
- Client-side patch operation was not working properly.
- Execution Status filter drop-down on Test Executions screen is out-of-order, has spaces, and duplicates
- Touchstone refuses to process an XML response with a DOCTYPE declaration.

TS 2.0.1 - January 07, 2016

Enhancements

- Improve error reporting when a request or response message cannot be parsed.
- Increase the width of the Origin and Destination drop-downs in Test Setup.
- Touchstone 2.0 User Guide.

Bug fixes

- The Supported Profiles column gets squashed on Test System list screen.
- Conformance retrieval needs to use access tokens for test systems that require OAuth2.

TS 2.0.0 - January 06, 2016

Enhancements

- Support for Client-side (Peer-to-Peer) Testing.
  - Origin elements are now presented on Test Setup and Test Execution screens as Touchstone may not be the origin for all message exchanges.
  - Support for multiple destinations in Test Setup and Test Execution.
  - Conformance checks are being done against multiple origins and destinations that could be involved in test executions.
- Support for user-defined variables in Test Setup.
- Test Setup (and reexecution of new Test Executions) will pull latest test scripts.
  - There is no need any more to create new Test Setups when test scripts change in the system.

  - When user clicks on Test Script, Fixture, and Profile links in old test executions, the latest Test Scripts and Fixtures are NOT displayed. The ones that were used at the time of execution are displayed.
- New Exchanges screen that displays all messages exchanges (Touchstone and Client initiated messages).
  - This screen allows users to filter for Response codes and overall Operation assertion status for their test systems. This allows users to find all "bad" response codes from their test systems, for example.
  - This screen is also useful in Client-side testing as request messages that do not match the user test execution can be found here.
- Improvements to Test Script Execution screen layout.
- Support for PATCH operation.
- Support for conformance operation via HTTP OPTIONS.
- Addition of Connectathon11 Test Scripts.
  - Track 1 - Patient
  - Track 2 - Terminology
  - Track 3 - CDS on FHIR
  - Track 4 - DAF
  - Track 6 - FHIR Genomics
  - Track 7 - Lab Order Lab Report
  - Track 9 - PATCH
- Upgrade of FHIR Validator to FHIR DSTU2 (v1.2.0-7493) a.k.a DSTU 2.1 Java RI - current trunk as of 12/30/2015.
- TestSetup search is now wild-carded on Test Execution search page.
- Link to test system's Conformance statement is now provided on Test System and Test System list screens.

13.2 TestScript Editor

IDE 2.0.0 - October 18, 2024

Enhancement

- Enhanced the Touchstone IDE to be bundled with Java 11 JDK so it no longer has dependence on any Java installation on the host system where it is installed

IDE 1.4.2 - December 9, 2022

Bug Fix

- Corrected issue impacting Conformance Suites when a single TestScript was uploaded via the TS IDE

IDE 1.4.0 - June 26, 2020

Enhancements

- Support for Multi-Profile Testing and Conformance Testing

IDE 1.3.0 - Jan 9, 2019

Enhancement

- Support for FHIR 4.0.0. See http://www.hl7.org/fhir/directory.cfml and http://hl7.org/fhir/R4.

IDE 1.2.2 - Nov 26, 2018

Enhancement

- Upload dialog should default to the last Spec selected by user.

IDE 1.2.1 - Sep 24, 2018

Enhancements

- Added features like "Check for Updates" and "Install New Software..", so users can find the latest released versions within the TestScript Editor itself.
- Added support to automatically check for updates and notify the user when new updates are available.
- Removed Java JRE bundled with TestScript Editor to give more flexibility for user to use a different JRE.
- Added support for FHIR 3.5.0 and CDS Hooks 1.0 specifications.

Bug Fix

- The "Spec" dropdown in "Upload to Touchstone" dialog was editable. It is now read-only.

IDE 1.1.0 - August 6, 2018

Enhancements

- Rebranding of TestScript Editor.
- Java JRE is now included in the TestScript Editor so user would not need to install Java.
- New toolbar menu item for contacting Touchstone Support.

IDE 1.0.0 - June 17, 2018

Enhancements

- Performance improvements during upload to Touchstone via new authentication mechanism.
- Added support for creation of Groovy Rules.
- Integration with new Simplifier API for JWT authentication.

IDE Beta-0.6.0 - May 29, 2018

Enhancement

- Add support for Touchstone IDE on Mac

IDE Beta-0.5.0 - May 13, 2018

First Release

The TestScript Editor is an Eclipse-based desktop development environment. It provides a comprehensive suite of development tools for creating, managing and publishing FHIR TestScript resources. It is designed to simplify test script development and accommodate a large number of users, ranging from beginners to experts.

The TestScript Editor can be used to:

- Upload Test Groups and TestScript resources to Touchstone.
- Upload Test Groups to and download them from Simplifier.
- Manage TestScript resources by integrating with Version Control systems such as SVN, GIT etc.

You can learn more about the TestScript Editor at Touchstone Docs.