# CONTENTS

1 Introduction 1

2 Registration 3
  2.1 Register ............................................. 3
  2.2 Membership .......................................... 4
    2.2.1 New Organization ............................... 4
    2.2.2 Become a member .................................. 5
    2.2.3 Approving Membership ............................ 6
  2.3 Subscribe ............................................ 8
  2.4 FAQ .................................................. 9

3 Test Systems 13
  3.1 Create Test System ................................. 13
  3.2 Capability Statement .................................. 17
  3.3 Test System List ..................................... 17

4 Executing Tests 21
  4.1 Creating Test Setup ................................ 21
  4.2 Test Executions ...................................... 24
  4.3 Test Execution Results ............................... 26
  4.4 FAQ ................................................ 31

5 Multi-Profile Testing 33
  5.1 Validator Testing ................................... 33
    5.1.1 Uploading ...................................... 34
    5.2 Updating Validators ................................ 36

6 Org Groups 39
  6.1 Creating Org Group .................................. 39
  6.2 Joining Org Group ................................... 39
  6.3 Leaving Org Group ................................... 41
  6.4 Access .............................................. 42
    6.4.1 Test System access ............................... 42
    6.4.2 Test Definition access ........................... 43
    6.4.3 Test Results access .............................. 46

7 Conformance Testing 49
  7.1 Conformance Suites ................................. 49
    7.1.1 Listing .......................................... 50
    7.1.2 Launching Executions ............................ 52
  7.2 Current ............................................. 52
# Table of Contents

## 7. Single Test Group

- 7.2.1 Single Test Group .......................................................... 52
- 7.2.2 Multi Test Group ............................................................ 62

## 7. Results Summary ................................................................. 70

## 7. Publishing Results ............................................................. 82

## 7. FAQ ......................................................................................... 89

## 8. Client/Peer-to-Peer Testing ................................................ 95

- 8.1 Launching Executions ......................................................... 95
- 8.2 Execution Matching ........................................................... 102
- 8.3 Exchanges Screen ............................................................... 106
- 8.4 Common Errors .................................................................... 107
- 8.5 FAQ ...................................................................................... 110

## 9. TestScript Authoring ............................................................. 113

- 9.1 FHIR TestScript ................................................................. 113
- 9.2 Upload on UI ...................................................................... 113
- 9.3 TestScript Editor ............................................................... 118
  - 9.3.1 Download .................................................................... 119
  - 9.3.2 Install a JVM ............................................................... 119
  - 9.3.3 Start TestScript Editor ............................................... 121
  - 9.3.4 Create TestScript Project ........................................... 121
  - 9.3.5 Create a Test Group ................................................... 123
  - 9.3.6 Create a TestScript ................................................... 125
  - 9.3.7 Code Completion ...................................................... 131
  - 9.3.8 Touchstone Integration .............................................. 132
    - 9.3.8.1 Upload to Touchstone ........................................... 132
  - 9.3.9 Simplifier Integration ................................................ 142
    - 9.3.9.1 Simplifier Preferences ......................................... 142
    - 9.3.9.2 Create New Project .............................................. 142
    - 9.3.9.3 Download from Simplifier .................................... 146
    - 9.3.9.4 Upload to Simplifier ............................................ 147
  - 9.3.10 Updating TestScript Editor ........................................ 149
  - 9.3.11 References ............................................................... 155
- 9.4 Best Practices ................................................................. 156
  - 9.4.1 Version Control ........................................................... 156
  - 9.4.2 Location ................................................................. 156
  - 9.4.3 Paths to resources ..................................................... 156
- 9.5 Exclusions ................................................................. 157
  - 9.5.1 Props Location ........................................................... 157
  - 9.5.2 Props format .............................................................. 158
  - 9.5.3 Upload Exclusions .................................................... 159
  - 9.5.4 Parsing Exclusions ................................................... 165
  - 9.5.5 Validation Exclusions .............................................. 170
  - 9.5.6 Conformance Exclusions ......................................... 171
- 9.6 Test Groups ................................................................. 172
  - 9.6.1 Access ................................................................. 172
  - 9.6.2 Deactivate ............................................................. 174
  - 9.6.3 Deletion .............................................................. 177
- 9.7 Rule Authoring ............................................................... 179
  - 9.7.1 Rule Basics ............................................................. 179
    - 9.7.1.1 Definition ........................................................ 179
    - 9.7.1.2 Declaration ....................................................... 183
    - 9.7.1.3 Assertion .......................................................... 183
    - 9.7.1.4 Summary and Description .................................. 184

## 9. TestScript Authoring ............................................................. 113

- 9.1 FHIR TestScript ............................................................. 113
- 9.2 Upload on UI .............................................................. 113
- 9.3 TestScript Editor ........................................................... 118
  - 9.3.1 Download ............................................................ 119
  - 9.3.2 Install a JVM ........................................................... 119
  - 9.3.3 Start TestScript Editor ............................................ 121
  - 9.3.4 Create TestScript Project ....................................... 121
  - 9.3.5 Create a Test Group ............................................... 123
  - 9.3.6 Create a TestScript ............................................... 125
  - 9.3.7 Code Completion ................................................... 131
  - 9.3.8 Touchstone Integration ........................................... 132
    - 9.3.8.1 Upload to Touchstone ....................................... 132
  - 9.3.9 Simplifier Integration ............................................. 142
    - 9.3.9.1 Simplifier Preferences ....................................... 142
    - 9.3.9.2 Create New Project .......................................... 142
    - 9.3.9.3 Download from Simplifier ................................ 146
    - 9.3.9.4 Upload to Simplifier ....................................... 147
  - 9.3.10 Updating TestScript Editor .................................... 149
  - 9.3.11 References .......................................................... 155
- 9.4 Best Practices ............................................................ 156
  - 9.4.1 Version Control ........................................................ 156
  - 9.4.2 Location ............................................................. 156
  - 9.4.3 Paths to resources .................................................. 156
- 9.5 Exclusions ............................................................. 157
  - 9.5.1 Props Location ........................................................ 157
  - 9.5.2 Props format .......................................................... 158
  - 9.5.3 Upload Exclusions ................................................ 159
  - 9.5.4 Parsing Exclusions ................................................ 165
  - 9.5.5 Validation Exclusions ............................................ 170
  - 9.5.6 Conformance Exclusions ....................................... 171
- 9.6 Test Groups .............................................................. 172
  - 9.6.1 Access ............................................................. 172
  - 9.6.2 Deactivate ........................................................... 174
  - 9.6.3 Deletion ............................................................ 177
- 9.7 Rule Authoring ........................................................... 179
  - 9.7.1 Rule Basics .......................................................... 179
    - 9.7.1.1 Definition ..................................................... 179
    - 9.7.1.2 Declaration .................................................... 183
    - 9.7.1.3 Assertion ...................................................... 183
    - 9.7.1.4 Summary and Description ................................ 184
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.
2.1 Register

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.
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:

![New Organization Link]

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

![Organization Registration Form]

**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 the 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 FAQ

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

When you registered into Touchstone, you should have received a Reset Key in your email. Please check your Spam folder if you can’t find this email. Once you’ve found your Reset Key, click on the Reset Password link:

![Password Reset Form](image)

Enter your Reset Key value and your email address on the next screen and you should get a new email with your new password and your new Reset Key value. Do not use the Reset Key as your password when you sign in later. Use the temporary password that was generated and sent in the email. The Reset Key (new one in the latest email) will be needed the next time you need to reset your password.
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 Password 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.
3.1 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 enable selection of your test system as the Destination (or ‘target’) of message exchanges in Test Setup, select the Server option for Profiles Supported:
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.
- **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, supply an OAuth2 token. Authorization request header will be set to this value by Touchstone when your test system is the target of an interaction.

• 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.

3. To enable OAuth, check the OAuth2 checkbox and enter the full expected value for the Authentication Header

• A default value is required if the OAuth2 checkbox is checked, but the value can be overridden at Test Setup if needed.
4. 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

---

**Chapter 3. Test Systems**
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.

### 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.

### 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 into Touchstone:

You can get rid of that by downloading the Capability Statement 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.

<table>
<thead>
<tr>
<th>AEGIS.net, Inc.</th>
<th>WildFHIR FHIR-1.4.0</th>
<th>FHIR 1.4.0 - STU3 CQF</th>
<th><a href="http://wildfhir2.aegis.net/fhir1-4-0">http://wildfhir2.aegis.net/fhir1-4-0</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>AEGIS.net, Inc.</td>
<td>WildFHIR FHIR-1.6.0</td>
<td>FHIR 1.6.0 - STU3 Ballot</td>
<td><a href="http://wildfhir2.aegis.net/fhir1-6-0">http://wildfhir2.aegis.net/fhir1-6-0</a></td>
</tr>
<tr>
<td>AEGIS.net, Inc.</td>
<td>WildFHIR FHIR-1.8.0</td>
<td>FHIR 1.8.0 - STU3 Candidate</td>
<td><a href="http://wildfhir2.aegis.net/fhir1-8-0">http://wildfhir2.aegis.net/fhir1-8-0</a></td>
</tr>
<tr>
<td>AEGIS.net, Inc.</td>
<td>WildFHIR FHIR-3.0.1</td>
<td>FHIR 3.0.1 - STU3 Official</td>
<td><a href="http://wildfhir3.aegis.net/fhir3-0-1">http://wildfhir3.aegis.net/fhir3-0-1</a></td>
</tr>
</tbody>
</table>

**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 into Touchstone.
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:

   ![Touchstone Test Setup Interface](image)

   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 Sets / List on the left menu:
4.2 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 by clicking the **Refresh** button every so often:
Warning: The screen does not refresh automatically at this time. You will need to click on the Refresh button or the browser refresh to see your test execution progress.

4.3 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 ... 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:

4.3. Test Execution Results
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:
4.4 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 separate Test Setups for any Tests that require a different Validator.
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).

![Test Definitions](image)

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
• Selecting multiple Validators. Note that the Additional TestSetup Validators are checked to give the tester options at test setup.
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 Tool for testing and validating the conformance of profiled FHIR® resources. Validation packages can be uploaded in two different formats:
  - 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

---

**5.2. Updating Validators**
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)
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 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:

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

![Conformance Suites Table]

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 Current (Conformance) page by clicking on 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 drop-downs 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). It’s `/FHIR 4.0.1-Basic` in this case - the root of the single test group that this Conformance Suite is composed of.
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. We’re a long ways from reaching 100%.
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 818 Patient interactions available in 154 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 140 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 423 JSON Patient interactions available in 89 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 70 JSON Patient interactions. See black arrows.
- In Advanced test group, there are a total of 219 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:

![Test Script Table](image)

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 5%.
8. Select a bunch of test scripts and click on **Execute Selected**:
Notice that ‘% Conformance’ has gone up to 7%. The “% conformant” does take into account all the interactions that have not been executed yet. It’s at 7% because we have only executed a few test scripts. We’re a long ways from reaching 100%.
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:

![Conformance Results Summary - FHIR4-0-1-Basic-Server v1](image)

We had executed JSON test scripts against both test systems.

Hover over the text to get more information about the results:
Conformance Results Summary - FHIR4-0-1-Basic-Server v1

Graphical Tabular Type FHIR-Server ▼ Suite FHIR4-0-1-Basic-Server

Interactions / (All) ▼ % Pass 0%

☑ Supported Only Format JSON ▼ My Org Test Systems

Records 1 - 2 of 2

Hover over the central band to get the aggregated counts:

Test System Initech-FHIR 4-0-1 QA Server
Conf Suite FHIR4-0-1-Basic-Server v1
Last Execution 06/15/2019 12:12:19PM
Oldest Execution 06/15/2019 10:39:04AM
Validator FHIR 4.0.1
Anchor System TouchstoneFHIR

73
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:

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

2. **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:

<table>
<thead>
<tr>
<th>Test System</th>
<th>% Pass</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initech-FHIR 4-0-1 QA Server</td>
<td>100%</td>
<td>/FHIR4-0-1-Basic/P-R/Pers</td>
</tr>
<tr>
<td>Initech-FHIR 4-0-1 Dev Server</td>
<td>50%</td>
<td>/FHIR4-0-1-Basic/P-R/Pers</td>
</tr>
</tbody>
</table>
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:

![Conformance Results Summary](image)

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

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

![Unpublish button](image)

9. Confirm that you want to unpublish:

![Unpublish confirmation](image)
10. Unpublished successfully:

![Conformance Results Summary](image)

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. After launching the execution, you may have to wait a few seconds before the Start time appears:

Test Scripts

<table>
<thead>
<tr>
<th>Test Script</th>
<th>Status</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>/FHIR4-0-1-Basic/D-H/HealthcareService/Client Assigned Id/HealthcareService-client-id-json</td>
<td>Running</td>
<td>06/24/2020 11:58:17AM</td>
</tr>
</tbody>
</table>

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 conformance 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.
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:

<table>
<thead>
<tr>
<th>Delete</th>
<th>Test Script</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>/FHIR4-0-0-Connectathon21/Patient-02-Formal/FHIRClient/01-RegisterPatient/patient-formal-c21-fhirclient-01-register-client-id-json</td>
<td>1</td>
<td>Patient Track - Formal Testing - tests external resource id and minimum data elements: id support the read operation and conditional variable example</td>
</tr>
</tbody>
</table>
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:

2. 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:

<table>
<thead>
<tr>
<th>Exec id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019062215095138699227</td>
<td>Patient Track - Formal Testing - Tests external FHIR Client and Server to register (i.e., JSON formatted Patient with a client assigned resource ID and minimum data elements such as identifier, name, family, name given, gender, and birthDate. The origin and destination must support the read operation and conditional create using the update (PUT) operation.</td>
</tr>
</tbody>
</table>

Test Setup: FHIR-4.0.0-Connection-21-patient-formal-c21-thrcient-01-register-client-id-json

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:

Profiles Supported

- FHIR-Client
- FHIR-Server

Match Peer-to-Peer client request to test execution using

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 mechanism, 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 mechanisms, the instructions are as follows:

```
<table>
<thead>
<tr>
<th>Test Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test: RegisterNewPatient</td>
<td>Create a new patient where the client assay</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateCreate - Patient</td>
<td></td>
<td>Waiting for Request</td>
</tr>
</tbody>
</table>

Submit the following request:

- **Method**: PUT
- **URL**: http://D2C73762.aegis.net:54910/fhir4-0-0/Patient/example
- **Request**: Body
  - **Headers**: ORG_KEY
  - **Content-Type**: application/fhir+json
  - **Accept**: application/fhir+json

Each organization is assigned a unique 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:

```
Test: RegisterNewPatient
- Create a new patient where the client a
```

**Submit the following request:**

- **From IP:** 10.0.75.1
- **Method:** PUT
- **URL:** http://D2C73762.aegis.net:54910/fhir/4-0-0/Patient/example
- **Request:** Body
- **Headers:**
  - Content-Type: application/fhir+json
  - Accept: application/fhir+json

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:
8.4 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:

![Edit Test System](image)

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.

![Match Peer-to-Peer client request to test execution using](image)

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.

![Match Peer-to-Peer client request to test execution using](image)

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:

![Test Script Execution](image)

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 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:
1. 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:

![Edit Test System](image1)

2. Uncheck this box on your test system.

![Profiles Supported](image2)

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

![Exchanges](image3)

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 recommended 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. Common Errors
8.5 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. 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.
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.

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:
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 Install a JVM

Note:
• TestScript Editor is a Java-based application and it requires a Java runtime environment (JRE) in order to run.

Regardless of your operating system, you will need to install Java virtual machine (JVM). You may either install a Java Runtime Environment (JRE), or a Java Development Kit (JDK), depending on what you want to do with TestScript Editor. A Java Development Kit (JDK) includes many useful extras for Java developers including the source code for the standard Java libraries.

Current releases of TestScript Editor require Java 8, click this link below for detailed instructions to install JDK 8 or JRE 8 on Microsoft Windows, Linux, and OS X computers.

• Instructions to Download & Install JDK or JRE

Set JAVA_HOME:

1. Microsoft Windows:
• Go to “Control Panel / System Properties / Advanced tab” and select “Environment Variables”.

• Set the JAVA_HOME variable value to the location of the JDK or JRE e.g. “C:/Program Files/Java/jdk1.8.0_181”.

• Add “%JAVA_HOME%/bin” to the PATH variable and click ‘OK’ to save the changes.

• To verify, open command line and run ‘java -version’. This should display the details of the JDK or JRE selected above.

2. Linux:

• To set JAVA_HOME in Linux for all users, we can use /etc/profile or /etc/environment (preferred).

• Open /etc/environment in any text editor and add:

  JAVA_HOME=/path/to/java_installation

Note:
• Please note that /etc/environment is not a script, but a list of assignment expressions (that is why export is not used). This file is read at the time of login.

• To set JAVA_HOME using /etc/profile, open the file and add:

  export JAVA_HOME=/path/to/java_installation

• Run the source command to load the variable:

  source /etc/profile

• Now we can check the value of the JAVA_HOME variable.

  echo $JAVA_HOME

• The result should be the path to the JDK or JRE installation, for example:

  /usr/lib/jvm/jdk1.8.0_181

3. Mac OS X:

• To set JAVA_HOME in Mac OS X for single user, we can use ~/.bash_profile

• Open ~/.bash_profile in any editor and add:

  export JAVA_HOME=/path/to/java_installation

• Save and close the file.

• Open a Terminal and run the source command to apply the changes:

  source ~/.bash_profile

• Now we can check the value of the JAVA_HOME variable:

  echo $JAVA_HOME

• The result should be the path to the JDK/JRE installation, for example:

  /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home

Note:
• To set JAVA_HOME globally for all users, the steps are the same as above, but the file /etc/profile is used.
9.3.3 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:

9.3.4 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.
9.3.5 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:

![Project Explorer with Client Assigned Id folder](image)

### 9.3.6 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 /FHIR4-0-1-Basic/P-R/Patient/Client Assigned
Id/Patient-client-id-json test script from Touchstone:

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:
9.3.7 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.

9.3.8 Touchstone Integration

Note:

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

9.3.8.1 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:

![Touchstone Interface]

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. You can download all of the Patient test scripts and fixtures from here. They are the same as /FHIR4-0-1-Basic/P-R/Patient test group on Touchstone UI:

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:
9.3.9 Simplifier Integration

9.3.9.1 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.

9.3.9.2 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.

9.3.9.3 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
9.3.9.4 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.

![Simplifier Log](image)

**9.3.10 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 software.

(a) 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.

### 9.3.11 References

Eclipse Workbench Documentation
9.4 Best Practices

9.4.1 Version Control

Although Touchstone does detect changes to uploaded test scripts and its dependent 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 Touchstone 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:
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 absolute path. The rule is used across many test groups in Touchstone.

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

**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 three 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 Validation 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:

```
Test Definitions
```

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:

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

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

To exclude “.sch” and “.xslt” files in Devices sub group from getting uploaded to Touchstone:
"excludeFromUpload": [ ]
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 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)
or

```json
{
   "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": [ ]
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:

9.5. Exclusions
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.
**Test Definitions - /FHIRSandbox/Initech/FHIR3-0-1-Basic**

**Name:** bad

[Create Test Setup]

[Resource ▼]

/FHIRSandbox/Initech/FHIR3-0-1-Basic/D-H/Device/_referenc.../resources/bad_format.xml

---

**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:
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:

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

```json
{
  "excludeFromValidation": [
    ".*/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 controlling 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

![Test Definitions](image)

- FHIREandbox
  - AEGIS
  - Initech
    - Allergy/intolerance
    - Device
      - Client Assigned Id
      - Server Assigned Id
    - Patient

2. Click on Delete

![Test Definitions UI](image)

3. Confirm deletion.

![Delete Test Group](image)
9.7 Rule Authoring

9.7.1 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.

9.7.1.1 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:

```xml
<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)."/>
    <direction value="response"/>
  </assert>
</action>
```

(continues on next page)
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.
9.7.1.2 Declaration

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

```xml
<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:

```xml
<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>
```

9.7.1.3 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:

```xml
<assert>
  <extension url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-assert-rule">
    <extension url="ruleId">
      <valueId value="assertETag"/>
    </extension>
    <warningOnly value="false"/>
  </extension>
</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.
9.7.1.4 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:

```groovy
/*
  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:

```groovy
assert false: "Could not find 'ETag' header in response"
```

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:

```groovy
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:
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:

```groovy
response/assertHeaderNotEmpty('ETag')
```

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

### 9.7.1.5 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

<table>
<thead>
<tr>
<th>Binding</th>
<th>Alternative bindings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exchange</td>
<td></td>
<td>The exchange taking place between a client (Touchstone or Test system) and a server Test System in Touchstone</td>
</tr>
<tr>
<td>request</td>
<td>exchange.request</td>
<td>The request message sent by a client (either Touchstone or another Test System in Touchstone).</td>
</tr>
<tr>
<td>response</td>
<td>exchange.response</td>
<td>The response received from the server Test System.</td>
</tr>
<tr>
<td>requestHeaders</td>
<td>request.headers</td>
<td>The headers of the request.</td>
</tr>
<tr>
<td>responseHeaders</td>
<td>response.headers</td>
<td>The headers of the response.</td>
</tr>
<tr>
<td>requestBody</td>
<td>request.body</td>
<td>The payload of the request.</td>
</tr>
<tr>
<td>responseBody</td>
<td>response.body</td>
<td>The payload of the response.</td>
</tr>
<tr>
<td>responseCode</td>
<td>statusCode, response.statusCode</td>
<td>e.g. 200. See RFC Status Codes and Touchstone ResponseCode rule assertions.</td>
</tr>
<tr>
<td>requestResource</td>
<td>request.resource</td>
<td>e.g. ‘Patient’ if the request message payload is present and the resource within it is a Patient resource. See FHIR Resource List.</td>
</tr>
<tr>
<td>responseResource</td>
<td>response.resource</td>
<td>e.g. ‘Bundle’ if the response message payload is present and the FHIR resource within it is a Bundle resource. See Touchstone Resource rule assertions.</td>
</tr>
<tr>
<td>clientCapStmt</td>
<td>originCapStmt, originConfStmt, clientConfStmt</td>
<td>See Touchstone Capability rule assertions.</td>
</tr>
</tbody>
</table>
9.7.2 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:

```groovy
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.

### 9.7.2.1 Definition

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

```groovy
/*
  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}')</pre>
```

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.

### 9.7.2.2 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:
2. Within the rule-assertion in a test:

```xml
<assert url="http://touchstone.aegis.net/touchstone/fhir/testing/StructureDefinition/testscript-assert-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>
</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:

```xml
<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:

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`:

```xml
<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-assertion to the test script after all the assertions:

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

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

That’s because we used parameters in the summary and description:
9.7.2.3 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:

```groovy
/*
   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:

```groovy
/*
   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
*/
```

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:

```xml
<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>
```

(continues on next page)
Re-upload and re-execute. You should see the results below:

<table>
<thead>
<tr>
<th>Assert</th>
<th>Response code is one of 200, 201</th>
<th>✔️ 0.005s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Description: Confirm that the returned HTTP status is 200(OK) or 201(Created).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Definition:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assert</th>
<th>Response 'ETag' header is present.</th>
<th>✔️ 0.454s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rule:</td>
<td>Confirm that 'ETag' header is present.</td>
</tr>
<tr>
<td></td>
<td>Definition:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assert</th>
<th>Response 'Last-Modified' header is present.</th>
<th>✔️ 0.318s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rule:</td>
<td>Confirm that 'Last-Modified' header is present.</td>
</tr>
<tr>
<td></td>
<td>Definition:</td>
<td></td>
</tr>
</tbody>
</table>

You can refer to [Header Rule API](#) for details on header assertions.
9.7.2.4 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.description=Confirm that response status code ${param.responseCodeOperator} $ → {param.responseCode}.
   rule.param.responseCode.required=true
   rule.param.responseCodeOperator.required=true
*/

```

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>
         <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>
</assert>
```

(continues on next page)
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:

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

### 9.7.3 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.

#### 9.7.3.1 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:
9.7. Rule Authoring
Copy the following contents into the newly created `RuleSet-VersioningHeaders.xml`:

```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.

### 9.7.3.2 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:

```xml
<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="ruleId">
      <valueId value="assertETag"/>
    </extension>
    <extension url="param">
      <extension url="name">
        <valueString value="header"/>
      </extension>
      <extension url="value">
        <valueString value="ETag"/>
      </extension>
    </extension>
  </extension>
</extension>
```

(continues on next page)
Notice that rule declarations have been replaced with the ruleset declaration in the test script:
9.7.3.3 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:
Re-upload and re-execute. You should see the following results:

![assertion-results-image]

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.

### 9.7.3.4 Overriding rules

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

```xml
<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"/>
        </extension>
        <extension url="value">
          <valueString value="201"/>
        </extension>
      </extension>
    </extension>
  </extension>
</assert>
```

(continues on next page)
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:

### 9.7.4 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 available in FHIR TestScript. The low-level APIs though can be combined to form more complicated rule logic than can be accomplished using TestScript assertions without rules.

#### 9.7.4.1 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:

    ```java
    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"
9.7.4.2 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 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 into Touchstone.

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

The advantage of assertSupportViaFhirPath assertions over assertSupportViaNonFhirPath, assertSupportViaXPath, and assertSupportViaJsonPath 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 into Touchstone. The disadvantage is that it will run significantly slower than XPath and JSONPath evaluation.

**Warning:** Avoid using assertSupportViaXPath, and assertSupportViaJsonPath assertions as they make assumptions about the content type of the capability statement downloaded into Touchstone. Use either assertSupportViaFhirPath or assertSupportViaNonFhirPath 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:
    ```java
    serverCapStmt.assertSupportViaFhirPath(
        "rest.where(mode.value='server').resource.where(type.value='Patient versioning').versioning",
        "versioned,versioned-update", "in", "Patient versioning")
    ```
  - Equivalent to these:
    ```java
    assertSupportViaFhirPath(
        "rest.where(mode.value='server').resource.where(type.value='Patient versioning').versioning",
        "versioned,versioned-update", "in", "Patient versioning", serverCapStmt)
    ```

    ```java
    boolean supports = serverCapStmt.supportsViaFhirPath(
        "rest.where(mode.value='server').resource.where(type.value='Patient versioning').versioning",
    (continues on next page)
• 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 expression to describe the capability.

  If the capability statement downloaded 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:

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

    - Equivalent to these:

      ```python
      assertSupportViaNonFhirPath(
        "rest\[mode/@value='server']/resource\[type/@value='Patient']/\n        →versioning",
        "rest[?(@.mode=='server')].resource[?(@.type=='Patient')].\n        →versioning",
        "versioned,versioned-update", "in", "Patient versioning")
      ```

      ```python
      boolean supports = serverCapStmt.supportsViaNonFhirPath(
        "rest\[mode/@value='server']/resource\[type/@value='Patient']/\n        →versioning",
        "rest[?(@.mode=='server')].resource[?(@.type=='Patient')].\n        →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)

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

- `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"

9.7.4.3 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.contentTypeContains("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.contentTypeNotContains("xml")
    ```
  – Equivalent to these:
    ```
    assertContentTypeNotContains("xml", response)
    ```
  ```
  // expectedValue and operator can be passed as parameters from test script.
  assertContentType("xml", "notContains", request)
  ```

9.7. Rule Authoring
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:

    ```java
    response.contentTypeEquals("application/fhir+json;charset=UTF-8")
    ```
  
  – Equivalent to these:

    ```java
    assertContentTypeEquals("application/fhir+json;charset=UTF-8", response)
    ```

    ```java
    // expectedValue and operator can be passed as parameters from test script.
    assertContentType("application/fhir+json;charset=UTF-8", "equals", request)
    ```

• assertContentTypeNotEquals(expectedValue)
  
  – Asserts that the value of header ‘Content-Type’ does not match the provided value
  
  – Example:

    ```java
    response.contentTypeNotEquals("application/fhir+xml")
    ```
  
  – Equivalent to these:

    ```java
    assertContentTypeNotEquals("application/fhir+xml", response)
    ```

    ```java
    // expectedValue and operator can be passed as parameters from test script.
    assertContentType("application/fhir+xml", "notEquals", request)
    ```
9.7.4.4 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)
    ```

- **assertHeaderNotContains(headerName, expectedValue)**
  - Asserts that the provided header `headerName` does not contain the provided `expectedValue`.
  - Example:
    ```
    request.header('Accept').contains('xml'): "The actual value "+
    request.header('Accept')?.value+" did not contain the expected
    'xml' for 'Accept' in request"
    ```
    ```
    response.header('Content-Type').contains('xml'): "The actual
    value "+
    response.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 "+
    response.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)
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:
    ```java
    response.assertHeaderEquals("Content-Type","text/html")
    ```
  - Equivalent to these:
    ```java
    assertHeaderEquals("Content-Type","text/html", response)
    ```

- **assertHeaderNotEquals**(headerName, expectedValue)
  - Asserts that the provided header headerName does not match the provided expectedValue
  - Example:
    ```java
    response.assertHeaderNotEquals("Content-Type","text/html")
    ```
  - Equivalent to these:
    ```java
    assertHeaderNotEquals("Content-Type","text/html", 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:
    ```javascript
    response.assertHeaderGreaterThan("Content-Length", 0)
    ```
  – Equivalent to each of these:
    * assertHeaderGreaterThan("Content-Length", 0, response)

    ```javascript
    // header, expectedValue, and operator can be passed as parameters from test script.
def assertHeader(headerName, expectedValue, operator, response)
    ```

    ```javascript
    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:
    ```javascript
    response.assertHeaderLessThan("Content-Length", 100)
    ```
  – Equivalent to each of these:
    * assertHeaderLessThan("Content-Length", 100, response)

    ```javascript
    // header, expectedValue, and operator can be passed as parameters from test script.
def assertHeader(headerName, expectedValue, operator, response)
    ```

    ```javascript
    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:
9.7.4.5 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(minimumId)**
  - 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:

```xml
<action>
  <assert>
    <description value="Confirm that the returned resource contains the expected retained elements and values.
    <minimumId value="patient-create-PeterChalmers-min"/>
    <warningonly value="true"/>
  </assert>
</action>
```
9.7.4.6 Path

Path assertions involve extracting element (or attribute) values and comparing them to an expected value.

9.7.4.6.1 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")
  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."
    ```

(continues on next page)
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

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:

  ```python
  response.assertFhirPathEmpty("entry[0].resource.photo.title")
  ```

  – Equivalent to these:

  ```python
  assertFhirPathEmpty("entry[0].resource.photo.title", response)
  response.fhirPath("entry[0].resource.photo.title").empty();
  ```

  ```python
  def title = response.getFhirPathValue("entry[0].resource.photo.title")
  assert !title: "Expected title to be absent but was present in response"
  ```

  ```python
  def title = response.fhirPathValue("entry[0].resource.photo.title")
  assert title==null: "Expected title to be absent but was present in response"
  ```

  ```python
  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:

  ```python
  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()

(continues on next page)
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 expected-Value.

  – Example:

```java
// 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:

```java
* 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)
```

```java
* def family = response.getFhirPathValue("entry[0].resource.name.family")
  
assert !family.equals("Chalmers"): "The actual value " + family + " did not 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:

    ```java
    // 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:

    ```java
    // 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:

```java
// 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:

```java
* 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."
```

### 9.7.4.6.2 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:

```java
// 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:

```java
* assertJsonPathContains("entry[1].resource.name.family", "Chalmers", response)
```
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)
    - assertJsonPath("entry[1].resource.name.family", "Chalmers", "notContains", response)

- assertJsonPathEmpty(jsonPath)

9.7. Rule Authoring 227
– Asserts that the evaluated value of the provided `jsonpath` expression is absent or empty.

– Example:

```java
response/assertJsonPathEmpty("entry[0].resource.photo.title")
```

– Equivalent to these:

```java
assertJsonPathEmpty("entry[0].resource.photo.title", response)
response(jsonPath("entry[0].resource.photo.title").empty());
```

```java
def title = response/getJsonPathValue("entry[0].resource.photo.title")
assert !title: "Expected title to be absent but was present in response"
```

```java
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:

```java
response/assertJsonPathNotEmpty("entry[0].resource.birthDate")
```

– Equivalent to these:

```java
assertJsonPathNotEmpty("entry[0].resource.birthDate", response)
response(jsonPath("entry[0].resource.birthDate").notEmpty());
```

```java
def title = response/getJsonPathValue("entry[0].resource.birthDate")
assert title: "Expected birthDate to be absent but was present in response"
```

```java
def title = response(jsonPath("entry[0].resource.birthDate").value
assertNotEmpty(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:

```java
// 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:

  ```python
  // 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:

  ```python
  assertJsonPathEquals("entry[0].resource.name.family", "Chalmers", response)
  ```

  ```python
  // jsonPath, expectedValue, and operator can be passed as parameters from test script.
  assertJsonPath("entry[0].resource.name.family", "Chalmers", "equals", response)
  ```

  ```python
  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."
  ```

9.7. Rule Authoring 229
• **assertJsonPathNotEquals(jsonPath, expectedValue)**
  
  - Asserts that the evaluated value of the provided `jsonpath` expression does not match the provided `expectedValue`.
  
  - Example:

    ```python
    // 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:

      ```python
      * 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+" 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+" matched the expected value "+"Chalmers" for "entry[0].resource.name.family" 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" in response."
      ```
• **assertJsonPathGreaterThan(jsonpath, expectedValue)**

  - Asserts that the evaluated value of the provided `jsonpath` expression is greater than the provided `expectedValue`.

  - Example:

    ```python
    // 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:

      ```python
      # assertJsonPathGreaterThan("entry[1].resource.id", 1100, response)
      # 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:

    ```python
    // 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:

      ```python
      # 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.jsonPathNotIn("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."

9.7.4.6.3 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)
assertNonFhirPath("entry[2]/resource/Patient/name/family", "entry[1].resource.name.family", "Chalmers", "contains", response)

```python
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."
```

```python
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:

    ```python
    // 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:

    ```python
    // xpath, 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)
    ```

    ```python
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\"" with operator 'notContains' in response."
```

```python
def family = response.nonFhirPath("entry[2]/resource/Patient/name/family", "entry[1].resource.name.family").value
```

(continues on next page)
• **assertNonFhirPathEmpty**(jsonpath)

  - Asserts that the evaluated value of the provided xpath or jsonpath expression is absent or empty.

  - Example:

    ```python
    response.assertNonFhirPathEmpty("entry[1]/resource/Patient/photo/title", "entry[0].resource.photo.title")
    ```

  - Equivalent to these:

    ```python
    assertNonFhirPathEmpty("entry[1]/resource/Patient/photo/title", "entry[0].resource.photo.title", response)
    ```

    ```python
    response.jsonPath("entry[1]/resource/Patient/photo/title", "entry[0].resource.photo.title").empty();
    ```

    ```python
    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"
    ```

    ```python
    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.

  - Example:

    ```python
    response.assertNonFhirPathNotEmpty("entry[1]/resource/Patient/birthDate", "entry[0].resource.birthDate")
    ```

  - Equivalent to these:

    ```python
    assertNonFhirPathNotEmpty("entry[1]/resource/Patient/birthDate", "entry[0].resource.birthDate", response)
    ```

    ```python
    response.nonFhirPath("entry[1]/resource/Patient/birthDate", "entry[0].resource.birthDate").notEmpty();
    ```

    ```python
    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"
    ```

    ```python
    def title = response.jsonPath("entry[1]/resource/Patient/birthDate", "entry[0].resource.birthDate").value
    assert !empty(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:

```java
// 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:

```java
* 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:

```java
// 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:

```java
* assertNonFhirPathNotEquals("entry[1]/resource/Patient/name/family", "entry[0].resource.name.family", "Chalmers", response)
```
assertNonFhirPath("entry[1]/resource/Patient/name/family", "entry[0].resource.name.family", "Chalmers", "notEquals", response)

```python
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."
```

```python
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."
```

```python
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:

    ```python
    // 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:

    ```python
    assertNonFhirPathGreaterThan("entry[2]/resource/Patient/id", "entry[1].resource.id", 1100, response)
    ```

    ```python
    // 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)
    ```

    ```python
    def id = response.getNonFhirPathValue("entry[2]/resource/Patient/id", "entry[1].resource.id")
    ```

(continues on next page)
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").getValue()
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:

```java
// 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.
  assertNonFhirPathNot("entry[2]/resource/Patient/id", "entry[1].resource.id", "1100,1101,1102", "in", response)
```
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."

9.7.4.6.4 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:

```python
response.assertXPathEmpty("entry[1]/resource/Patient/photo/title")
```

- Equivalent to these:

```python
assertXPathEmpty("entry[1]/resource/Patient/photo/title", response)
response.fhirPath("entry[1]/resource/Patient/photo/title").empty();
```

```python
def title = response.getXPathValue("entry[1]/resource/Patient/photo/→title")
assert !title: "Expected title to be absent but was present in response →"
```

```python
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:

```python
response.assertXPathNotEmpty("entry[1]/resource/Patient/birthDate")
```

- Equivalent to these:

```python
assertXPathNotEmpty("entry[1]/resource/Patient/birthDate", response)
response.xPath("entry[1]/resource/Patient/birthDate").notEmpty();
```

```python
def title = response.getXPathValue("entry[1]/resource/Patient/birthDate/→")
assert title: "Expected birthDate to be absent but was present in response →"
```

```python
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:

```python
// Asserts that the value of the family element of the first entry is →'Chalmers'
response.assertXPathEquals("entry[1]/resource/Patient/name/family", →"Chalmers")
```

9.7. Rule Authoring 243
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 \\
  "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 \\
  "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 \\
  "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 expected-Value.
  - 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)"
• `assertXPathGreaterThan(xpath, expectedValue)`
  - Asserts that the evaluated value of the provided `xpath` expression is greater than the provided `expectedValue`.
  - Example:
    ```java
    // 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:
    ```java
    * assertXPathGreaterThan("entry[2]/resource/Patient/id", 1100, response)
    ```
* `def id = response.xpath("entry[2]/resource/Patient/id").value`

```java
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:

```java
// 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:

```java
assertXPathLessThan("entry[1]/resource/Patient/id", 1100, response)
assertXPath("entry[1]/resource/Patient/id", "1100", "lessThan", response)
```

* `def id = response.getXPathValue("entry[1]/resource/Patient/id")`

```java
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()`

```java
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`

```java
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:

```java
// 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:

```java
assertXPathIn("entry[2]/resource/Patient/id", ["1100", "1101", "1102"], response)
```
• `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")`

  ```python
  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`

  ```python
  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:

    ```python
    // 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:

    ```python
    assertXPathNotIn("entry[2]/resource/Patient/id", "1100,1101,1102", response)
    ```

    ```python
    // xPath, expectedValue, and operator can be passed as parameters from test script.
    assertXPath("entry[2]/resource/Patient/id", "1100,1101,1102", "notIn", response)
    ```

    ```python
    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."  
    ```

    ```python
    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."  
    ```

9.7. Rule Authoring
9.7.4.7 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:

```xml
<action>
  <assert>
    <description value="Validate that the returned resource conforms to the corresponding FHIR bundle profile."/>
    <validateProfileId value="bundle-profile"/>
    <assert/>
  </assert>
</action>
```

- Profile definition in TestScript:

```xml
<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:

```python
response.assertValidWithProfileId("bundle-profile")
```

  - Equivalent to these:

```python
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:

```python
response.assertValidWithProfile("http://hl7.org/fhir/StructureDefinition/Bundle")
```

  - Equivalent to these:

```python
response.validateWithProfile("http://hl7.org/fhir/StructureDefinition/Bundle")
assertValidWithProfile("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:
    ```java
    response.assertValidXmlExtractWithProfileId("Bundle/entry[1]/resource/Patient", "bundle-profile")
    ```
  – Equivalent to these:
    ```java
    response.extractXmlAndValidateWithProfileId("Bundle/entry[1]/resource/Patient", "bundle-profile")
    response.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:
    ```java
    response.assertValidXmlExtractWithProfile("Bundle/entry[1]/resource/Patient", "http://hl7.org/fhir/StructureDefinition/Bundle")
    ```
  – Equivalent to these:
    ```java
    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:
    ```java
    ```
  – Equivalent to these:
    ```java
    ```

9.7. Rule Authoring
9.7.4.8 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:
    ```java
    request.assertRequestMethodEquals("GET")
    ```
  - Equivalent to these:
    ```java
    assertRequestMethodEquals("GET", request)
    ```
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)
• assertRequestMethodNotIn(expectedValues)
  – Asserts that the method of the request is none of the provided `expectedValues` where each value is separated by a comma.
  – Example:

```java
request.assertRequestMethodNotIn("POST, PUT")
```

– Equivalent to these:

```java
assertRequestMethodNotIn("POST, PUT", request)
```

// expected method and operator can be passed as parameters from testscript.
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"

```java
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"

```java
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"
```

### 9.7.4.9 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:

```java
request.assertRequestURLEquals("http://touchstone.aegis.net:57084/fhir3-0-1/Patient?name=Connectathon15")
```

– Equivalent to these:

```java
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 operator 'notEquals' in request"
• **assertRequestURLContains(expectedValue)**
  
  – Asserts that the URL of the request contains the provided `expectedValue`
  
  – Example:

    ```java
    request.assertRequestURLContains("fhir3-0-1/Patient?name=Connectathon15")
    ```

  – Equivalent to these:

    ```java
    assertRequestURLContains("fhir3-0-1/Patient?name=Connectathon15", request)
    ```

    ```java
    // expected URL and operator can be passed as parameters from test
    assertRequestURL("fhir3-0-1/Patient?name=Connectathon15", "contains", request)
    ```

    ```java
    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"
    ```

    ```java
    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:

    ```java
    request.assertRequestURLNotContains("fhir3-0-1/Patient?name=Connectathon15")
    ```

  – Equivalent to these:

    ```java
    assertRequestURLNotContains("fhir3-0-1/Patient?name=Connectathon15", request)
    ```

    ```java
    // expected URL and operator can be passed as parameters from test
    assertRequestURL("fhir3-0-1/Patient?name=Connectathon15", "notContains", request)
    ```

    ```java
    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"
    ```

    ```java
    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""
    ```

(continues on next page)
9.7.4.9.1 Request URL Prefixes

The `expectedValue` in requestURL can be prefixed for fine-grained assertion of individual parts of the URL.
<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
<th>Operators</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemeHostPortPath:</td>
<td>Validates the scheme, host, port, and path.</td>
<td>equals, notEquals</td>
<td>“schemeHostPortPath: <a href="http://hostname:11111/fhirSys1%E2%80%9D">http://hostname:11111/fhirSys1”</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>contains, notContains</td>
<td>“schemeHostPortPath: <a href="http://hostname:11111/fhir%E2%80%9D">http://hostname:11111/fhir”</a></td>
</tr>
<tr>
<td>schemeHostPort:</td>
<td>Validates the scheme, host, and port.</td>
<td>equals, notEquals</td>
<td>“schemeHostPort: <a href="http://hostname:11111%E2%80%9D">http://hostname:11111”</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>contains, notContains</td>
<td>“schemeHostPort: <a href="http://hostname:111%E2%80%9D">http://hostname:111”</a></td>
</tr>
<tr>
<td>schemeHost:</td>
<td>Validates the scheme and host.</td>
<td>equals, notEquals</td>
<td>“schemeHost: <a href="http://hostname%E2%80%9D">http://hostname”</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>contains, notContains</td>
<td>“schemeHost: <a href="http://hos%E2%80%9D">http://hos”</a></td>
</tr>
<tr>
<td>scheme:</td>
<td>Validates the scheme.</td>
<td>equals, notEquals</td>
<td>“scheme: https”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>contains, notContains</td>
<td>“scheme: http”</td>
</tr>
<tr>
<td>hostPortPath:</td>
<td>Validates the host, port, and path.</td>
<td>equals, notEquals</td>
<td>“hostPortPath: hostname:11111/fhir”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>contains, notContains</td>
<td>“hostPortPath: hostname:111”</td>
</tr>
<tr>
<td>hostPort:</td>
<td>Validates the host and port.</td>
<td>equals, notEquals</td>
<td>“hostPort: hostname:11111”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>contains, notContains</td>
<td>“hostPort: hostname:1”</td>
</tr>
<tr>
<td>host:</td>
<td>Validates the host.</td>
<td>equals, notEquals</td>
<td>“host: hostname”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>contains, notContains</td>
<td>“host: hostna”</td>
</tr>
</tbody>
</table>
9.7.4.10 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:

    ```python
    response.assertResourceIn("Bundle, Patient")
    ```
  - Equivalent to these:

    ```python
    assertResourceIn("Bundle, Patient", response)
    ```
    
    ```python
    // expected resource type and operator can be passed as parameters
    assertResource("Bundle, Patient", "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:

    ```python
    response.assertResourceNotIn("Bundle, Patient")
    ```
  - Equivalent to these:

    ```python
    assertResourceNotIn("Bundle, Patient", response)
    ```
    
    ```python
    // expected resource type and operator can be passed as parameters
    assertResource("Bundle, Patient", "notIn", response)
    ```
9.7.4.11 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:
    ```java
    response.assertResponseCodeEquals(200)
    ```
  - Equivalent to these:
    ```java
    assertResponseCodeEquals(200, response)
    ```
    ```java
    assertResponseCode(200, "equals", response)
    ```
    ```java
    
    - did not match the expected value "200" for response code in
    
    - response"
    ```
    ```java
    
    - did not match the expected value "200" for response code in
    
    - response"
    ```
    ```java
    assert equals(200, responseCode): "The actual value "+responseCode+"\n
    - 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:
    ```java
    response.assertResponseCodeNotEquals(200)
    ```
  - Equivalent to these:
    ```java
    assertResponseCodeNotEquals(200, response)
    ```
    ```java
    assertResponseCode(200, "notEquals", response)
    ```
  →responseCodeInt+\"\" matched the expected value \"200\" for response code with operator
  →'notEquals' in response"

  →response.responseCode+\"\" matched the expected value \"200\" for response code with operator
  →'notEquals' in response"

  →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:

    ```java
    response/assertResponseCodeGreaterThan(399)
    ```
  – Equivalent to each of these:
    * **assertResponseCodeGreaterThan**(399, response)

    ```java
    // 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:

    ```java
    response/assertResponseCodeLessThan(300)
    ```
  – Equivalent to each of these:
    * **assertResponseCodeLessThan**(300, response)

    ```java
    // expected status code and operator can be passed as parameters from test script.
    assertResponseCode(300, "lessThan", 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:

    ```java
    // Asserts that the response code is either 200 or 201
    response.assertResponseCodeIn("200,201")
    ```

    - Equivalent to each of these:

      ```java
      * assertResponseCodeIn("200,201", response)
      ```

      ```java
      // expected status code and operator can be passed as parameters from
      // test script.
      assertResponseCode("200,201", "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:

    ```java
    // Asserts that the response code is neither 200 nor 201
    response.assertResponseCodeNotIn("200,201")
    ```

    - Equivalent to each of these:

      ```java
      * assertResponseCodeNotIn("200,201", response)
      ```

      ```java
      // expected status code and operator can be passed as parameters from
      // test script.
      assertResponseCode("200,201", "notIn", response)
      ```

      ```java
      * 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"
      ```
9.7.5 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.

9.7.5.1 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.

9.7.5.2 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).
9.7.5.3 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).
9.7.6 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.

9.7.6.1 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
-->
```
9.7.6.2 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:

9.7.6.2.1 Groovy

```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 "
    + responseHeaders.header['Content-Type']+'""
```

9.7.6.2.2 XSLT

```xml
<?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: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'
```
The right one will be supplied in place of headers.

```xml
<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))">
            <t:error>Expected Content-Type '<xsl:value-of select="$param/expectedContentType" />' did not match actual Content-Type</t:error>
          </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>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
    <t:error>The headers were not supplied.</t:error>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>

9.7.6.2.3 Schematron

```xml
<?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" url="http://hl7.org/fhir/>
  <sch:ns prefix="h" url="http://www.w3.org/1999/xhtml"/>
</sch:schema>
```
9.7.6.3 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:

9.7.6.3.1 Groovy

```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 + '""
```
9.7.6.3.2 XSLT

```xml
<?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:t="http://touchstone.aegis.net/" version="2.0">
    <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>
                                    <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>
```

9.7.6.3.3 Schematron

```xml
<?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" url="http://hl7.org/fhir/"/>
    <sch:ns prefix="h" url="http://www.w3.org/1999/xhtml"/>
</sch:schema>
```

(continues on next page)
9.8 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:

   ```xml
   <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 available here.

3. When launching Touchstone IDE I am getting an error about the Java version If you are seeing this error please configure TouchstoneIDE.ini as follows depending on your environment:

   **Microsoft Windows – add in the blue bolded lines to the TouchstoneIDE.ini.** Please check to see where the jdk 8 built and ensure your path is correct for your machine:
   ```ini
   -startup
   plugins/org.eclipse.equinox.launcher_1.4.0.v20161219-1356.jar
   -launcher.library
   plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.551.v20171108-1834
   -launcher.appendVmargs
   -vm
   C:Program FilesJavajdk1.8.0_221bin
   -vmargs
   -Dosgi.requiredJavaVersion=1.8
   ```
Mac OS X – add in the red bolded lines to TouchstoneIDE.ini. Please check to see where the jdk 8 built and ensure your path is correct for your machine:

- startup
  ../Eclipse/plugins/org.eclipse.equinox.launcher_1.4.0v20161219-1356.jar
  -launcher.library
  ../Eclipse/plugins/
  org.eclipse.equinox.launcher.cocoa.macosx.x86_64_1.1.551.v20171108-1834
  -launcher.appendVmargs
  -vm
  /Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home/bin
  -vmargs
  -Dorg.eclipse.swt.internal.carbon.smallFonts
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

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 Server 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:

\[\text{You can create Conformance Suite only with test groups owned by Initech. No test groups have been uploaded by Initech. You can upload one here if you have 'Test Editor' role in the system.}\]

**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:

<table>
<thead>
<tr>
<th>Test Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FHIRSandbox</td>
</tr>
<tr>
<td>AEGIS</td>
</tr>
<tr>
<td>Initech</td>
</tr>
<tr>
<td>FHIR3-0-2-S-Advanced</td>
</tr>
<tr>
<td>FHIR3-0-2-S-Basic</td>
</tr>
<tr>
<td>FHIR3-0-2-S-Conn</td>
</tr>
</tbody>
</table>

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:

   **Name**
   
   Initech-Basic

   **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:

   FHIR3-0-2-Initech-Basic-Server

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 Validator 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:

   ![Format dropdown](image)

   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:

   ![Publish checkbox](image)

   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. Removing this flag on a Conformance Suite with existing published results will also prohibit users from unpublishing their results:
Please refer to Publishing to learn more about conformance results publishing.

14. Select the access attributes of the suite:

**Can be viewed by**
- Me
- My organization
- Everyone

**Can be modified by**
- Me
- My organization
- Everyone

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:

Profiles Supported *

- [x] FHIR-Client
- [ ] FHIR-Server

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:

Name *

Initech-Basic

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:

FHIR3-0-2-Initech-Basic-Client

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 Validator 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:
Touchstone Documentation, Release 5.0.0

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. Removing this flag on a Conformance Suite with existing published results will also prohibit users from unpublishing their results:
14. Select the access attributes of the suite:

<table>
<thead>
<tr>
<th>Can be viewed by</th>
<th>Can be modified by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>Me</td>
</tr>
<tr>
<td>My organization</td>
<td>My organization</td>
</tr>
<tr>
<td>Everyone</td>
<td>Everyone</td>
</tr>
</tbody>
</table>

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:

   Categorization is needed when more than one test group is selected. You can upload a FHIR-Client Categorization file for FHIR 3.0.2 here, or reduce the number of test groups to 1.

   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:

Categorization is needed when more than one test group is selected. You can upload a FHIR-Client Categorization file for FHIR 3.0.2 here. Can reduce the number of test groups to 1.

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:

```
<category name="FHIR3-0-2-Standard-Client">
   <category name="CapabilityStatement" path="resource/CapabilityStatement"/>
   <category name="ImplementationGuide" path="resource/ImplementationGuide"/>
   <category name="SearchParameter" path="resource/SearchParameter"/>
   <category name="OperationDefinition" path="resource/OperationDefinition"/>
   <category name="StructureDefinition" path="resource/StructureDefinition"/>
   <category name="GraphDefinition" path="resource/GraphDefinition"/>
   <category name="Definition" path="resource/Definition"/>
   <category name="Language" path="resource/Language"/>
   <category name="ConceptMap" path="resource/ConceptMap"/>
   <category name="ValueSet" path="resource/ValueSet"/>
   <category name="NamingSystem" path="resource/NamingSystem"/>
   <category name="Reference" path="resource/Reference"/>
   <category name="Provenance" path="resource/Provenance"/>
   <category name="AuditEvent" path="resource/AuditEvent"/>
   <category name="EventDefinition" path="resource/EventDefinition"/>
</category>
```

We’ve saved the contents into a file called “InitechCert.xml”:
9. Change the description appropriately:

```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="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="CompartentDefinition" path="resource/CompartentDefinition" />
        <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>
    <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>
</xml>
```

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:

10.3. *Categorization*
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):
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:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<category spec="FHIR 3.0.2" type="FHIR-Client"
    description="Initech Cert Categorization for FHIR 3.0.2 Client">
    <category name="Resources">
        <category name="Foundation">
            <category name="Conformance">
                <category name="CapabilityStatement" path="resource/CapabilityStatement" />
                <category name="StructureDefinition" path="resource/StructureDefinition" />
                <category name="ImplementationGuide" path="resource/ImplementationGuide" />
                <category name="SearchParameter" path="resource/SearchParameter" />
                <category name="MessageDefinition" path="resource/MessageDefinition" />
                <category name="OperationDefinition" path="resource/OperationDefinition" />
                <category name="CompartmentDefinition" path="resource/CompartmentDefinition" />
                <category name="StructureMap" path="resource/StructureMap" />
                <category name="GraphDefinition" path="resource/GraphDefinition" />
                <category name="DataElement" path="resource/DataElement" />
            </category name="Terminology">
                <category name="CodeSystem" path="resource/CodeSystem" />
                <category name="ValueSet" path="resource/ValueSet" />
                <category name="ConceptMap" path="resource/ConceptMap" />
        </category name="Terminology">
    </category name="Resources">
</category spec="FHIR 3.0.2" type="FHIR-Client"
    description="Initech Cert Categorization for FHIR 3.0.2 Client">
```

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:

```xml
<catag>
  <catag name="Management">
    <catag name="encounter" path="resource/Encounter" />
    <catag name="EpisodeOfCare" path="resource/EpisodeOfCare" />
    <catag name="Flag" path="resource/Flag" />
    <catag name="List" path="resource/List" />
    <catag name="Library" path="resource/Library" />
  </catag>
</catag>
<catag name="Clinical">
  <catag name="AllergyIntolerance" path="resource/AllergyIntolerance" />
  <catag name="AdverseEvent" path="resource/AdverseEvent" />
  <catag name="Condition" path="resource/Condition" />
  <catag name="Procedure" path="resource/Procedure" />
  <catag name="FamilyMemberHistory" path="resource/FamilyMemberHistory" />
  <catag name="ClinicalImpression" path="resource/ClinicalImpression" />
  <catag name="DetectedIssue" path="resource/DetectedIssue" />
</catag>
<catag name="Diagnostics">
  <catag name="Observation" path="resource/Observation" />
  <catag name="DiagnosticReport" path="resource/DiagnosticReport" />
  <catag name="Specimen" path="resource/Specimen" />
  <catag name="BodySite" path="resource/BodySite" />
  <catag name="ImagingStudy" path="resource/ImagingStudy" />
  <catag name="ImagingManifest" path="resource/ImagingManifest" />
  <catag name="QuestionnaireResponse" path="resource/QuestionnaireResponse" />
  <catag name="Sequence" path="resource/Sequence" />
</catag>
```

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:
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):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<category spec="FHIR 3.0.2" type="FHIR-Client"

description="Intech Cert Categorization for FHIR 3.0.2 Client">
<category name="Resources">
  <category name="Foundation">
    <category name="conformance">
      <category name="CapabilityStatement" path="resource/CapabilityStatement" />
      <category name="StructureDefinition" path="resource/StructureDefinition" />
      <category name="ImplementationGuide" path="resource/ImplementationGuide" />
      <category name="SearchParameter" path="resource/SearchParameter" />
      <category name="MessageDefinition" path="resource/MessageDefinition" />
      <category name="OperationDefinition" path="resource/OperationDefinition" />
      <category name="CompartmentDefinition" path="resource/CompartmentDefinition" />
      <category name="StructureMap" path="resource/StructureMap" />
      <category name="GraphDefinition" path="resource/GraphDefinition" />
      <category name="DataElement" path="resource/DataElement" />
    </category>
    <category name="Terminology">
      <category name="CodeSystem" path="resource/CodeSystem" />
      <category name="ValueSet" path="resource/ValueSet" />
      <category name="ConceptMap" path="resource/ConceptMap" />
      <category name="ExpansionProfile" path="resource/ExpansionProfile" />
      <category name="NamingSystem" path="resource/NamingSystem" />
    </category>
    <category name="Security">
      <category name="Provenance" path="resource/Provenance" />
      <category name="AuditEvent" path="resource/AuditEvent" />
      <category name="Consent" path="resource/Consent" />
    </category>
    <category name="Documents">
      <category name="Composition" path="resource/Composition" />
      <category name="DocumentManifest" path="resource/DocumentManifest" />
      <category name="DocumentReference" path="resource/DocumentReference" />
    </category>
    <category name="Other">
      <category name="Basic" path="resource/Basic" />
      <category name="Binary" path="resource/Binary" />
      <category name="Bundle" path="resource/Bundle" />
      <category name="Linkage" path="resource/Linkage" />
      <category name="MessageHeader" path="resource/MessageHeader" />
      <category name="OperationOutcome" path="resource/OperationOutcome" />
      <category name="Parameters" path="resource/Parameters" />
      <category name="Subscription" path="resource/Subscription" />
    </category>
  </category>
</category>
</category>
<category name="Base">
  <category name="Individuials">
    <category name="Patient" path="resource/Patient" />
    <category name="Practitioner" path="resource/Practitioner" />
    <category name="PractitionerRole" path="resource/PractitionerRole" />
    <category name="PractitionerRoleCode" path="resource/PractitionerRoleCode" />
  </category>
</category>
```

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:

```xml
  <categ name="Foundation"/>
  <categ name="Base"/>
  <categ name="Clinical">
    <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>
</categ>
```

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:

```
<table>
<thead>
<tr>
<th>Interactions</th>
<th>0% passed</th>
<th>Pass</th>
<th>Fail</th>
<th>Other</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td></td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Procedure</td>
<td></td>
<td>0</td>
<td>0</td>
<td>56</td>
<td>56</td>
</tr>
</tbody>
</table>
```

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:

```
<catg name="Foundation"/>
<catg name="Base">
  <catg name="Individuals">
    <catg name="Patient" path="resource/Patient"/>
    <catg name="Practitioner" path="resource/Practitioner"/>
    <catg name="PractitionerRole" path="resource/PractitionerRole"/>
    <catg name="RelatedPerson" path="resource/RelatedPerson"/>
    <catg name="Person" path="resource/Person"/>
    <catg name="Group" path="resource/Group"/>
  </catg>
  <catg name="Entities"></catg>
  <catg name="Management"></catg>
</catg>
<catg name="Clinical"></catg>
<catg name="Financial"></catg>
<catg name="Specialized"></catg>
</catg>
<catg name="Operations">
  <catg name="$apply" path="operation/apply"/>
  <catg name="$close" path="operation/closure"/>
</catg>
```

### 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.

```
<catg name="Individuals"/>
  <catg name="Patient" path="resource/Patient"/>
  <catg name="Practitioner" path="resource/Practitioner"/>
  <catg name="PractitionerRole" path="resource/PractitionerRole"/>
  <catg name="RelatedPerson" path="resource/RelatedPerson"/>
  <catg name="Person" path="resource/Person"/>
  <catg 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.

```xml
<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"/>
    <categ name="$find" path="operation/find"/>
</categ>
```

• 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:

<table>
<thead>
<tr>
<th>Whole System Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>capabilities</td>
</tr>
<tr>
<td>batch/transaction</td>
</tr>
<tr>
<td>history</td>
</tr>
<tr>
<td>search</td>
</tr>
</tbody>
</table>

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:
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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initech-Multiple</td>
<td>FHIR-Client</td>
<td>Initech-FHIR 3</td>
</tr>
</tbody>
</table>

Validator

Test Group

/FHIRSandbox/Initech/FHIR3-0-2-C-Basic

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:

1. The content of one or more test definitions (test scripts, fixtures, or rules) within the Test Group changes.
2. 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 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:

```
<catg name="Foundation"></catg>
<catg name="Base"></catg>
<catg name="Clinical">
<catg name="Summary">
<catg name="AllergyIntolerance" path="resource/AllergyIntolerance" />
<catg name="AdverseEvent" path="resource/AdverseEvent" />
<catg name="Condition" path="resource/Condition" />
<catg name="Procedure" path="resource/Procedure" />
<catg name="FamilyMemberHistory" path="resource/FamilyMemberHistory" />
<catg 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:

- Allow filtering of conformance results by supported interactions in the capability statement
- Allow filtering of conformance results by format (All / JSON / XML)
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.

![Name field](image)

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.

![Description field](image)

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.

![Publishable attribute](image)

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.

![Access attributes](image)

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 last 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-BasicCert”, 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-BasicDev” 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-BasicDev” 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-BasicDev” test group as “FHIR4-0-1-Initech-BasicCert” 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.

- Allow publishing of conformance results

### 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.
## Conformance Suites

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>History</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FHIR4-0-1-Standard-Server</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 be 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.
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 recommended 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.

<table>
<thead>
<tr>
<th>URL</th>
<th><a href="https://touchstone.aegis.net/touchstone/api/authenticate">https://touchstone.aegis.net/touchstone/api/authenticate</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>POST</td>
</tr>
<tr>
<td>Examples</td>
<td>See goodBasic-authenticateRequest-xxx and goodBasic-authenticateResponse-xxx files within authenticate folder in touchstone-api.zip</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>URL</th>
<th><a href="https://touchstone.aegis.net/touchstone/api/testExecution">https://touchstone.aegis.net/touchstone/api/testExecution</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>POST</td>
</tr>
<tr>
<td>Examples</td>
<td>See good-executeTestRequest-xxx and good-executeTestResponse-xxx files within executeTest folder in touchstone-api.zip</td>
</tr>
</tbody>
</table>

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.

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.

<table>
<thead>
<tr>
<th>URL</th>
<th><a href="https://touchstone.aegis.net/touchstone/api/testExecution/%5BtestExecId">https://touchstone.aegis.net/touchstone/api/testExecution/[testExecId</a>]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>GET</td>
</tr>
<tr>
<td>Example URL</td>
<td><a href="https://touchstone.aegis.net/touchstone/api/testExecution/20160516101258248">https://touchstone.aegis.net/touchstone/api/testExecution/20160516101258248</a></td>
</tr>
<tr>
<td>Examples</td>
<td>See good-testExecStatusRequest-xxx and good-testExecStatusResponse-xxx files within testExecStatus folder in touchstone-api.zip</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>URL</th>
<th><a href="https://touchstone.aegis.net/touchstone/api/testExecDetail/%5BtestExecId">https://touchstone.aegis.net/touchstone/api/testExecDetail/[testExecId</a>]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>GET</td>
</tr>
<tr>
<td>Example URL</td>
<td><a href="https://touchstone.aegis.net/touchstone/api/testExecDetail/20160516101258248">https://touchstone.aegis.net/touchstone/api/testExecDetail/20160516101258248</a></td>
</tr>
<tr>
<td>Examples</td>
<td>See good-testExecDetailRequest-xxx and good-testExecDetailResponse-xxx files within testExecDetail folder in touchstone-api.zip</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>URL</th>
<th><a href="https://touchstone.aegis.net/touchstone/api/scriptExecDetail/%5BtestExecId%5D?testscript=%5Bvalue">https://touchstone.aegis.net/touchstone/api/scriptExecDetail/[testExecId]?testscript=[value</a>]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>GET</td>
</tr>
<tr>
<td>Example URL</td>
<td><a href="https://touchstone.aegis.net/touchstone/api/scriptExecDetail/20180507121755387?testscript=FHIR3-2-0-Connectathon17/Patient-02-Formal/FHIRServer/03-PatientRead/connectathon-17-patient-fhirserver-03-read-client-id-json">https://touchstone.aegis.net/touchstone/api/scriptExecDetail/20180507121755387?testscript=FHIR3-2-0-Connectathon17/Patient-02-Formal/FHIRServer/03-PatientRead/connectathon-17-patient-fhirserver-03-read-client-id-json</a></td>
</tr>
<tr>
<td>Examples</td>
<td>See good-scriptExecDetailRequest-xxx and good-scriptExecDetailResponse-xxx files within scriptExecDetail folder in touchstone-api.zip</td>
</tr>
</tbody>
</table>

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:

```java
// 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

// Retrieve the test execution status in a loop until completion.
TestExecStatusResponse execStatusResponse = RestClient.useRelaxedHTTPSValidation()
    .headers("API-Key:"+
        authenticateResponse.getApiKey(), "Accept: "application/json")
    .get("https://touchstone.aegis.net/touchstone/api/testExecution"")
```

(continues on next page)
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.

<table>
<thead>
<tr>
<th>URL</th>
<th><a href="https://touchstone.aegis.net/touchstone/api/testReport/%5BtestExecId%5D?testscript=%5Bvalue">https://touchstone.aegis.net/touchstone/api/testReport/[testExecId]?testscript=[value</a>]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>GET</td>
</tr>
<tr>
<td>Examples</td>
<td>See good-testReport-xxx and good-testReport-xxx files within testReport folder in touchstone-api.zip</td>
</tr>
</tbody>
</table>

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.
11.2 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.

```groovy
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.
Touchstone Documentation, Release 5.0.0

// 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) {
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}"
(continues on next page)

322

Chapter 11. Continuous Integration


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) {
    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:

![Add post-build step menu](image)

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.

![Post Steps](image)

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-loggin-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-groovysdoc-2.4.4.jar
• $APPS_HOME/groovy-2.4.4/lib/groovy-groovyssh-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/httpcore-4.3.3.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

11.2. Jenkins Integration Example
• $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
TestScript Editor (Touchstone IDE)
Comprehensive suite of development tools for creating, managing and publishing TestScript resources.

<table>
<thead>
<tr>
<th>Version</th>
<th>Release Date</th>
<th>Linux</th>
<th>Mac</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.0</td>
<td>June 26, 2020</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Touchstone-API zip
Contains schemas and sample messages for using Touchstone API in Continuous Integration.

<table>
<thead>
<tr>
<th>Version</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8.0+</td>
<td>touchstone-api.zip</td>
</tr>
</tbody>
</table>
13.1 Touchstone

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.
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.
Touchstone Documentation, Release 5.0.0

- 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**

- 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.
- 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.

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
• 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
• 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 <request-header> element in TestScript operation definition and not the usual <accept> and <content-type> 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.
• 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


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


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

• Misc 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.

Bug fix
• Analytics/Conformance screen throws error when number of test scripts exceeds 256 for a given specification.

Enhancement
• Support for SSL-cert communication between Touchstone and Test Systems.

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.

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.

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.

Bug fixes
• System errors when user attempts to update the access token on Test Setup screen (instead of Test System screen).

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 TestSets.

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.

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).
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.

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.

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.
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 1.4.0 - June 26, 2020

**Enhancements**

- Support for Multi-Profile Testing and Conformance Testing

IDE 1.3.0 - Jan 9, 2019

**Enhancement**


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.