Applitools Simple Test Runner (ASTR)

Applitools Simple Test Runner is an Applitools Eyes oriented, XML based, testing framework for using Applitools Visual Testing capabilities, as well as navigating the tested website with basic commands, without any knowledge of coding.

** TO REDUCE ISSUES PLEASE CHECK THE IMPORTANT COMMENTS SECTION BEFORE STARTING **

How does it work?

Requirements

- Java installed (version 1.8 or higher).
- When running on a local browser - make sure to have the proper browser driver.

Step 1 - Download the Applitools Simple Test Runner Jar file:

Download the latest version of the Applitools Simple Test Runner from HERE.

Step 2 - Create a project (or generate the example project):

a) Create an Applitools Configuration File, according to the following example:

applitoolsConf.xml

```
<settings>
	<apikey>YOUR_API_KEY</apikey>
</settings>
```

Note: see additional optional settings here.

b) Configure a job to execute, similar to the following example:

job.xml

```
<job>
	<batch name="my batch">
		<test appname="myapp" testname="mytest">
			<configuration>
				<applitools>applitoolsConf.xml</applitools>
				<driver>chrome</driver> <!-- Additional options: firefox, safari, ie, edge -->
				<viewport width="600" height="600"></viewport>
			</configuration>
			<execute>
				<action type="navigate">http://the-internet.herokuapp.com/login</action>
				<action type="sendkeys" by="css" selector="#username">john.smith@applitools.com</action>
				<action type="sendkeys" by="css" selector="#password">12345</action>
				<action type="click" by="css" selector="#login > button > i"></action>
				<action type="checkwindow">step name</action>
			</execute>
		</test>
	</batch>
</job>
```

Note: see additional execution commands here.

Step 3 - Run your created project:

Open the terminal and run the command:
Java -jar ApplitoolsSimpleTestRunner.jar job.xml

Step 4 - View your results:
Check your results in your Applitools dashboard!!! :)

Applitools Configuration Settings

When specifying the settings for Applitools Configurations (in the applitoolsConf.xml file), you can specify any of the following settings:

Setting	Description
`<apikey>YOUR_API_KEY</apikey>`	Specify the unique API KEY of your team. Note: this setting is mandatory.
`<viewKey>YOUR_VIEW_KEY</viewKey>`	Specify the unique VIEW KEY of your team. Relevant only for downloading baseline/current/diff images and the result JSON file.
`<MatchLevel>Layout</MatchLevel>`	Specify the wanted Applitools Match Level for your tests.
`<hostApp>hostApp</hostApp>`	Overriding the name of the host application (browser) that will be a part of the unique definition of the baseline that is related to your test.
`<hostOS>hostOS</hostOS>`	Overriding the name of the host operating system that will

	be a part of the unique definition of the baseline that is related to your test.
`<BaselineEnvName>baselineName</BaselineEnvName>`	Define a baseline name to enable cross-environment testing.

`<Proxy>https://proxy.url.com:8080</Proxy>`
 	
OR
 	
`<proxy username="JohnSmith" password="1234">https://proxy.url.com:8080</proxy>`
	Configure a Proxy to address issues such as connection refusal or connection timeout when using Applitools Eyes behind a proxy server.
	
	Optional attributes:
	• If there is a need for a username and password add: `username="JohnSmith" password="1234"`
	
	Please note the difference between a proxy configuration to instantiate a remote webdriver (in the driver configuration file) to this setting (in the Applitools configuration file). The two are independent of one another.

`<Server>https://eyes.applitools.com</Server>`	On-Premise and Private Cloud deployment modes allow customers to work with their own private Applitools Eyes server (as opposed to Applitools' public cloud). In order to use your private server when running tests, you need to explicitly declare which server you are working with. The default setting is to the public cloud: https://eyes.applitools.com
`<Fullpage>false</Fullpage>`	Most browsers do not give a full-page screenshot by default. In order to use Applitools' Eyes full-page screenshot capability, this flag is set to **true by default**. In order to override this behavior, set it to false.
`<StitchMode>scroll</StitchMode>`	The default stitching method is set to CSS Stitching to handle fixed elements that may appear on the page and will be duplicated when

OR `<StitchMode>css</StitchMode>`	stitching the different sections of the web-page for a full-page screenshot. In order to use regular scrolling, set the stitching method to “scroll”.
`<crop header="65" footer="20" left="10" right="10"></crop>`	Using the Applitools Image Cutter, to crop out a fixed border, while stitching the different sections of the web-page for a full-page screenshot. Any value that is specified will crop the relevant number of pixels out from the stitched image. Any value that is not specified will be set to zero by default.
`<Branch>child branch</Branch>`	When working with Multiple Branches, set the branch name in order to run your tests in a separate branch.
`<ParentBranch>parent branch</ParentBranch>`	When working with Multiple Branches, set the parent branch name in order to inherit existing baselines when running a test for the first time.
`<Matchtimeout>5000</Matchtimeout>`	For increased stability, define an implicit Match Timeout wait for your visual tests (the default is set to 2 seconds).
`<WaitBeforeScreenshot>3000</WaitBeforeScreenshot>`	If needed, specify an explicit wait before taking each screenshot, for increased stability of your visual tests.
`<SaveFailedTests>true</SaveFailedTests>`	In order to override the baselines automatically, with the result of the tests run, set this flag to true (the default setting is false). Important note: this is only needed in very unique edge cases. Be sure this is the desired behavior you are aiming for.
`<SaveNewTests>true</SaveNewTests>`	When running a new test, the result creates a new baseline by default. In order to change this behavior, set this flag to false.

`<hideScrollBars>false</hideScrollBars>`	In order to hide the scrollbars, before taking the screenshot, set this flag to true. The default value is true.
`<scaleRatio>2.0</scaleRatio>`	In rare cases, setting the scaling ratio is needed to get better visual results. If you get to a point where your visual testing results are pixelated or if an unintended section is captured when checking a specific region, please contact Applitools Support to consult about it.
`<rotateImageAngle>180</rotateImageAngle>`	Will rotate the captured screenshot by the specified angle.
`<stitchOverlap>150</stitchOverlap>`	Will increase the overlap between screenshot stitches (for the handling of bottom page fixed elements). The default value is 100px.
`<sendDom>true</sendDom>`	Will enable Dom sending for Root Cause Analysis feature (true by default, if stitch mode is configured to “CSS”, otherwise - false by default).
`<Accessibility level="AA" guidelinesVersion="WCAG_2_0" />`	Will run Applitools tests with the Applitools Accessibility Contrast Advisor enabled.

Supported Actions

The actions in this section can be inserted within any action supporting block (Execute/Foreach…).

All actions support the addition of the attribute “**onError**” with the following values:

- **continue** - will continue to the next action without throwing an error.

- **skipIteration** - if specified for an action within a foreach block, will terminate the current iteration and will skip to the next one without throwing an error. Specifying onError=”skipIteration” for an action within the main execution block will terminate the run.
- **skip:n** - will skip the n subsequent actions.

Example:

```
<action type="click" by="css" selector="<SELECTOR>" onError="continue"/>
```

Basic Execution Actions

Command	Syntax	Description
Navigate	`<action type="navigate">https://www.applitools.com</action>` Additional options: To perform browser back: `<action type="navigate">back</action>` To perform browser forward: `<action type="navigate">forward</action>` To perform browser refresh: `<action type="navigate">refresh</action>`	Navigate to a specified URL
Send Keys	`<action type="sendkeys" by="css" selector="<SELECTOR>">John Smith</action>`	Type in an input box. “By” options: Css / xpath / id / tagName / linkText / className / partialLinkText / name Optional attributes: • `implicit_wait = "5"` (value is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds

		was reached). ● clickReturn="true" (will click the “RETURN” key after sending the keys to the element.
Click	<action type="click" by="css" selector="<SELECTOR>" />	Click an element. “By” options: Css / xpath / id / tagName / linkText / className / partialLinkText / name Optional attributes: ● implicit_wait = "5" (value is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds was reached).
Drag&Drop	<action type="DragAndDrop"> <source by="css" selector="<SELECTOR>" /> <target by="css" selector="<SELECTOR>" /> </action>	“By” options: Css / xpath / id / tagName / linkText / className / partialLinkText / name Optional attributes: ● implicit_wait = "5" (value is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds was reached).

MouseOver	<action type="mouseOver" by="css" selector="<SELECTOR>" />	Click an element. “By” options: Css / xpath / id / tagName / linkText / className / partialLinkText / name Optional attributes: ● implicit_wait = "5" (value is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds was reached).
Execute JavaScript Command	<action type="executejs">JS_COMMAND</action> * Note: if the returned value is a **webelement** it can be used in any element action by using the attribute **elementVariable=”<variable_name>”**. For example, clicking the returned variable named “**variable_name**” would be as follows: **<action type="click" elementVariable="variable_name" />**	Execute a javascript command on the current web page. Optional attributes: ● OutputVariable="returned Value" (will store the returned string value as a variable called ‘returnedValue’).
Explicit Wait	<action type="sleep">5000</action>	Explicit wait for the number of specified milliseconds (5 seconds in the example).
Implicit Wait	<action type="implicitWait" by="css" selector="<selector>" implicit_wait="10" onError="continue"/>	Explicit wait for the number of specified seconds.

		“By” options: Css / xpath / id / tagName / linkText / className / partialLinkText / name implicit_wait = "5" (value is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds was reached).
Force Page Load	`<action type="forcepageload" />`	When using CSS stitching, elements that are triggered to load by a regular scroll will not load. Therefor you can use this action to initially scroll through the page to make sure all the elements were loaded (this will not do anything unless the scrolling method is css). Optional attributes: ● suspension_time (will suspend each scroll for the specified amount of milliseconds - the default value is 1 second). ● `returnToTop="false"` will not scroll back to the top of the page at the end of the page load.

Go to Manual Navigation State	`<action type="goManual" />`	Use this command to pause the automated test and perform actions manually or check information about the current state of your test.

Switch Context	`<action type="switchContext" destination="default" />`
	
	OR
	
	`<action type="switchContext" destination="iFrame" by="css" selector="<SELECTOR>" />`
	
	OR
	
	`<action type="switchContext" destination="window" index="1" />`
	
	OR
	
	`<action type="switchContext" destination="alert" action="accept" />`
	
	OR
	
	`<action type="switchContext" destination="alert" action="dismiss" />`
		Will switch the context of the driver to the specified Iframe/window(or tab) or back to the default context.
		
		“By” options:
		Css / xpath / id / tagName / linkText / className / partialLinkText / name
		
		Optional attributes:
		● implicit_wait = "5" (value is in seconds. The default is 5. Will keep looking for the element, when an element is specified, until found or the number of seconds was reached).
		● closeCurrent="true" - when switching to a window/tab destination adding this attribute will close the current

		tab/window when switching.
Switch Context (Mobile Context)	`<action type="switchContext" destination="mobileContext" context="NATIVE_APP"/>`	Will switch the context of the driver to the first mobile context which contains the **sub-string** of the specified context (Only for Mobile Appium drivers). The available mobile contexts can be found by using the "goManual" action and checking the "-info" parameter.
Add Cookie	`<action type="addCookie" cookieName = "name" cookieValue = "value" />`	Will add the specified cookie to the current site.
Delete All Cookies	`<action type="deleteAllCookies" />`	Will delete all the predefined cookies.
Execute Command	`<action type="executeCommand" command="java -jar ApplitoolsSimpleTestRunner.jar nextFile.xml -narrate" />`	Will execute the given OS command.

Collect Links	`<action type="CollectLinks" collectionName="urls">`
	`<element by="css" selector="<selector>"/>`
	`<logFileInfo>`
	`<path>/Downloads/collectedUrls.csv</path>`
	`<js title="Origin">return window.location.href</js>`
	`<js title="text">return arguments[0].innerText;</js>`
		Will grab all the links URLs within the given element container, and will append them into a collection called "urls" (in the example) without duplicates.
		This collection can be later on

	`</logFileInfo>` `</action>`	crawled with the collection crawler action. Optional: ● LogFileInfo tag will generate a CSV file with the links that have been collected. Path - specifies the absolute path to where the file will be saved. Js - will execute the specified JS code and will save the result under the CSV column with the given title attribute. Arguments[0] will be applied on the <a> tag of the collected link.

Select	`<action type="select" by="css" selector="<SELECTOR>" selectBy="selectByVisibleText" text="some text"/>`
	
	OR
	
	`<action type="select" by="css" selector="<SELECTOR>" selectBy="deselectByVisibleText" text="some text"/>`
	
	OR
	
	`<action type="select" by="css" selector="<SELECTOR>"`
		The Select capability provides the implementation of the HTML SELECT tag. A Select tag provides the helper methods with select and deselect options.
		
		Overloads:
		
		● **selectByVisibleText:** This method is used to select one of the options

	selectBy="selectByIndex" index="1"/> OR <action type="select" by="css" selector="<SELECTOR>" selectBy="deselectByIndex" index="1"/> OR <action type="select" by="css" selector="<SELECTOR>" selectBy="selectByValue" value="some value"/> OR <action type="select" by="css" selector="<SELECTOR>" selectBy="deselectByValue" value="some value"/>	in a drop-down box or an option among multiple selection boxes. • **selectByIndex:** This method is similar to 'selectByVisibleText', but the difference here is that the user has to provide the index number for the option rather than text. • **selectByValue:** This method asks for the value of the desired option rather than the option text or an index.
Hide Elements	<action type="hidElements" by="css" selector="<selector>"/>	This action will make all the elements on the page with the given selector invisible.

Applitools Eyes Actions

All check actions support the attribute createSeparateTest="true", which will generate a separate test in the Applitools dashboard. If this attribute is used, an additional attribute testName must be set. This will be used as the identifier for the baseline.

For Example,

```
<action type="checkWindow" createSeparateTest="true" testName="newTestName"></action>
```

If a check command has the seperate test attribute specified, the following additional attributes are supported as well:

- `overrideBaseline = "true" (this will override the baseline for this test).`
- `deleteOn="finish" (this will delete this test from the dashboard when it finishes running.`
- `matchLevel="layout/strict/content/exact" (this will override the global match level for this test).`

Command	Syntax	Description
Check Window	<action type="checkwindow">step name</action>	Applitools Check Window (takes a screenshot and sends it to Applitools Server).
Check Region	<action type="checkregion" by="css" selector="<SELECTOR>">step name</action>	Applitools Check Region “By” options: Css / xpath / id / tagName / linkText / className / partialLinkText / name Optional attributes: ● implicit_wait = "5" (value is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds was reached). ● stitchContent = "false" (will prevent content stitching if set to false).

Check Region By given coordinates	<action type="checkregion" by="region" left="100" top="50" width="500" height="300">step name</action>	Applitools Check Region by a given region.
Check Frame	<action type="checkFrame" id="<iframeID>">step name</action> OR <action type="checkFrame" name="<iframeName>">step name</action>	Applitools Check Frame (takes a screenshot of the specified IFrame and sends it to Applitools Server).
Check Image	<action type="checkimage" testname="test_name" createSeparateTest="true" path="PATH_TO_IMAGE/file.png" />	Optional attributes: ● scaleToViewportWidth="true" (will scale the image to the fit the viewport width of the test).
Check	<action type="check" target = "region" by="css" selector="<selector>" stepName="..."> <add type="scrollRootElement" by="css" selector="<selector>" /> <add type="ignoreRegion" by="css" selector="<selector1>" /> <add type="ignoreRegion" by="css" selector="<selector2>" /> <add type="layoutRegion" by="css" selector="<selector1>" /> <add type="ignoreRegions" by="css" selector="<selector1>" /> <add type="ignoreRegion" by="css" selector="<selector2>" /> <add type="layoutRegions" by="css" selector="<selector1>" /> <add type="gridLayoutBreakpoints" viewportWidthValue="750" /> <add type="gridLayoutBreakpoints" viewportWidthValue="950" />	Optional attributes: ● Option for target are: **region**, **window**, **frame** (only by selector, not id/name). ● fullPage="true" (optional, in the "action" tag) for full page per check. ● useDom="true" (optional, in action tag) to use the DOM for Layout

	<add type="gridLayoutBreakpoints" viewportWidthValue="all" /> <!-- will add a layout breakpoint for each viewport width → <add type="visualGridOption" key="polyfillAdoptedStyleSheets" value="true" /> </action> **Notes**: • notice the difference between “ignoreRegion” and “ignoreRegions” or the difference between “layoutRegion” and “layoutRegions”. The singular form will only apply the region to the first element on the page with the given css selector. The plural form would apply the region to *all* element on the page that the given css selector applies to. • If the UFG gridLayoutBreakpoints is set to “**all**” once, the rest of the layout breakpoints are ignored.	calculations. • enablePatterns=”true” (optional, in action tag) to analyze complex patterns for Layout calculations (requires DOM). • ignoreDisplacementDiffs= ”true” • Multiple Coded Ignore regions can be added • Match level can be specified for the specific check (optional). Example: matchLevel="Layout" • beforeRenderScreenshot Hook="..." (allows executing JS code on the UFG before rendering the screenshot). This is only relevant when running on the UFG. • variationGroupId="<id>" (optional). Sets a variation group ID for auto-maintenance purposes with A/B experiment tests.

Iterative Actions

Command	Syntax	Description
Crawl a sitemap.xml file	`<action type="crawl" from="sitemap.xml file" createSeparateTests="true" path="path_to_sitemap.xml" />` **Important Note:** `from="sitemap.xml file"` should be hard coded and never changed. This value specifies that the crawl is on a file of type sitemap.xml and not the path to the file of URLs.	Crawl through a sitemap.xml file (See Appendix A for example) and does a window check for each URL. Optional attributes: ● select="indexes:1,5,7-9" (will only crawl through the urls with the following indexes: 1,5,7,8,9 in the sitemap.xml file). ● select="domains:applitools.com,google.com" (will only crawl through the urls with the specified comma separated domains). ● createSeparateTests="true" (will create separate tests, within the same batch, for each url in the sitemap.xml file).
Crawl URLs in collection	`<action type="crawl" from="collection" createSeparateTests="true" collectionName="urls"/>`	Will crawl the URLS in the collection called “urls” (in the example) and grab a screenshot of each one. You can override the crawl

		behavior with the <foreach> tag.
Crawl through all links within given Elements	<action type="crawl" from="links in elements" createSeparateTests="true"> <element by="css" selector="<SELECTOR 1>" /> <element by="css" selector="<SELECTOR 2>" /> </action>	Crawl through all link-destinations within the specified elements and does a window check for each url. "By" options: Css / xpath / id / tagName / linkText / className / partialLinkText / name Optional attributes (added to the action tag): • select="indexes:1,5,7-9" (will only crawl through the urls with the following indexes: 1,5,7,8,9 in the sitemap.xml file). • select="domains:applitools.com,google.com" (will only crawl through the urls with the specified comma separated domains). • createSeparateTests="true" (will create separate tests, within the same batch, for each link within the element). Optional attributes (added to the element tag): • implicit_wait = "5" (value

		is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds was reached).
Iterate Elements	<action type="iterateElements" tag="a"> <withinContainer by="css" selector="<SELECTOR>" /> <foreach> <action type="click"/> <action type="checkWindow">step name</action> </foreach> </action> Note: actions that require an element (such as click, send keys, drag&drop…) will use the iterated element when the “by” and “selector” attributes are not specified.	Iterates all the elements with the HTML tag specified by the ‘**tag**’ attribute, within the specified element container and performs the actions within the **foreach** block for each element. “By” options (for the “withinContainer” tag): Css / xpath / id / tagName / linkText / className / partialLinkText / name Optional attributes (added to the “withinContainer” tag): ● implicit_wait = "5" (value is in seconds. The default is 5. Will keep looking for the element until found or the number of seconds was reached).

Overriding default sequence

In order to override the default sequence for each url, when crawling, which is:

1) Navigate to the URL.
2) Do a window check.

You can add a <foreach></foreach> block with the actions/functions that you would like to perform. Within that block a local variable called “url” will be accessible with the value of the URL that is being handled.

Example #1:

```
<action type="crawl" from="sitemap.xml file" select="domains:applitools.com" path="sitemap.xml" createSeparateTests="true">
	<foreach>
		<action type="function-addPrefix" prefix="https://" to="exp[%url%]" OutputVariable="url" />
		<action type="function-addQueryParams" to="exp[%url%]" OutputVariable="url">
			<param key="key1" value="value1" />
			<param key="key2" value="value2" />
		</action>
		<action type="navigate">exp[%url%]</action>
		<action type="checkWindow">exp[%url%]</action>
	</foreach>
</action>
```

Example #2 (Comparison between environments):

```
<action type="crawl" from="sitemap.xml file" createSeparateTests="true" path="sitemap.xml">
	<foreach>
		<action type="function-addPrefix" prefix="https://firstPrefix" to="exp[%url%]" OutputVariable="urlBaseline" />
		<action type="function-addPrefix" prefix="https://secondPrefix" to="exp[%url%]" OutputVariable="urlCurrent" />
		<action type="navigate">exp[%urlBaseline%]</action>
		<action type="checkWindow" createSeparateTest="true" overrideBaseline = "true"
			 deleteOn="finish" testName="exp[%url%]">exp[%urlBaseline%]</action>
		<action type="navigate">exp[%urlCurrent%]</action>
		<action type="checkWindow" createSeparateTest="true" testName="exp[%url%]">exp[%urlCurrent%]</action>
	</foreach>
</action>
```

Example #3:

```
<action type="crawl" from="links in elements" createSeparateTests="true">
	<element by="css" selector="<SIDE_NAVIGATION_MENU_SELECTOR>" />
	<foreach>
		<action type="checkFrame" name="<FRAME_NAME>">step name</action>
	</foreach>
</action>
```

SmartSplit URL Crawling to Different Tests

In many cases splitting the crawled sitemap or collection of URLs to different url groups and crawling each one with a separate driver can be beneficial (e.g., when running concurrently, in order to save time).
In order to accomplish that, simply add a SmartSplit tag to your test as follows:

```
<job concurrent_runs="10">
	<load>configurations.xml</load>
	<batch name="BatchName">
		<test appname="AppName" testname="TestName">
			<configurations groups="locally" />
			<SmartSplit to="sitemap url groups" max="50" path="sitemap.xml" maxAttempts="5"/>
			<execute>
				<login /> <!-- authentication process before crawling sitemap -->
				<action type="crawl" from="sitemap.xml file" createSeparateTests="true" path="sitemap.xml"/>
			</execute>
		</test>
	</batch>
</job>
```

In this example, for a sitemap.xml file with 150 urls, 3 browsers will start, **each one** will **go through the login process** and crawl a **different** set of 50 urls from the sitemap.xml files - indexes: 1-50, 51-100, 101-150.

Attributes:

- max - specifies the maximum number of urls in a single group.
- path - specifies the path to the sitemap.xml file that will be used to split the test.

- maxAttempts (optional) - in case of intermittent errors (exceptions) that cause the run of the test to crash, while crawling a sitemap (e.g., timeouts, memory limits, etc.) a self-healing mechanism will attempt to spawn the crashed driver again and rerun the entire test, completing the **remaining** URLs that have not finished successfully. In order to prevent an infinite loop a "maxAttempts" attribute must be defined.

Note: The total number of configurations and viewport sizes across the different URL groups will not exceed the specified concurrency in the **Job** tag.

Another option for a SmartSplit is when crawling a URL collection. Notice that in this case, it is important to define a predecessor test where the URLs will be collected first, otherwise the list of URLs will be empty.

```
<test appname="APP"  testname="URLS" predecessors="URLCollection">
            <configurations names="Ultrafastgrid" />
            <SmartSplit to="collection url groups" max="1" collectionName="Urls" maxAttempts="5"/>
            <execute>
                        <action type="crawl" from="collection" createSeparateTests="true" collectionName="Urls">
                                    <foreach>
                                                <action type="navigate">exp[%url%]</action>
                                                <action type="sleep">1000</action>
                                                <action type="forcepageload" returnToTop="false"/>
                                                <action type="checkWindow">exp[%url%]</action>
                                    </foreach>
                        </action>
            </execute>
</test>
```

Supported Functions

Function	Syntax	Description
Add Prefix	`<action type="function-addPrefix" prefix="https://" to="www.applitools.com" OutputVariable="fullUrl" />`	Creates a new local variable (or overrides it if already exists) named “fullUrl” with the values of the “prefix” and the “to” attributes concatenated. I.e., in the syntax example the result would be: “https://www.applitools.com”

Add Query Parameter	`<action type="function-addQueryParams" to="https://www.applitools.com" OutputVariable="url">`
	`  <param key="key1" value="value1" />`
	`  <param key="key2" value="value2" />`
	`</action>`
		Creates a new local variable (or overrides it if already exists) named “url” with the value of the URL specified by the “to” attribute and the specified query params added to it.
		
		I.e., in the syntax example the results will be:
		
		https://www.applitools.com?key1=value1&key2=value2

Cleaner & Modular Coding

Using Environment Variables

Within any block or attribute value it is possible to get a value from an environment variable by specifying its name within Env[...].

For example, in order to get your Applitools API Key from an environment variable called **APPLITOOLS_API_KEY** add the following line to your **applitoolsConf.xml** file:

```
<apikey>Env[APPLITOOLS_API_KEY]</apikey>
```

Using Predefined variables/Configurations/Configuration Groups

1) Defining Variables

Within any block or attribute value it is possible to get a the value of a predefined variable in an external definitions file by specifying its name with **var[...]** or within an expression surrounded by an asterisk (e.g., variable called "**key**" can be read either by **var[key]** or by **exp[*key*].**

For example,

Create a definition file as follows:

definitions.xml

```
<definitions>
	<var key="User Name">#username</var>
	<var key="Password">#password</var>
	<var key="Login Button">#login > button > i</var>
</definitions>
```

Then **load** the definitions.xml file in the job.xml file and **use the variable key** in your code as follows:

job.xml

```
<job>
	<load>definitions.xml</load>
	<batch name="my batch">
		<test appname="myapp" testname="mytest">
			<configuration>
				<applitools>applitoolsConf.xml</applitools>
```

```
                        <driver>chrome</driver> <!-- Additional options: firefox, safari, ie, edge -->
                        <viewport width="600" height="600"></viewport>
                    </configuration>
                    <execute>
                        <action type="navigate">http://the-internet.herokuapp.com/login</action>
                        <action type="sendkeys" by="css" selector="var[User Name]">john.smith@applitools.com</action>
                        <action type="sendkeys" by="css" selector="var[Password]">12345</action>
                        <action type="click" by="css" selector="var[Login Button]"></action>
                        <action type="checkwindow">step name</action>
                    </execute>
                </test>
        </batch>
</job>
```

2) Defining External Configurations and Configuration Groups

In order to create an external configuration block, simply add that block as you would create it within a test block and add a name parameter to it, as follows:

```
<definitions>
	<configuration name="chromeLocally">
		<applitools>applitoolsConf.xml</applitools>
		<driver>chrome</driver>
		<viewport width="600" height="600"></viewport>
	</configuration>
	<configuration name="firefoxLocally">
		<applitools>applitoolsConf.xml</applitools>
		<driver>firefox</driver>
		<viewport width="600" height="600"></viewport>
	</configuration>
</definitions>
```

Then include it in the test block as follows (you can add as many configurations as you want, comma separated):

```
<configurations names="chromeLocally, firefoxLocally" />
```

In addition, you can also define a configuration group in the definition.xml file as follows:

```
<configurationGroup groupName="locally" configurationsNames="chromeLocally, firefoxLocally" />
```

And then include it in your test block as follows (you can add as many groups as you want, comma separated):

```
<configurations groups="locally" />
```

NOTE: notice the difference between a “Configuration” block for specifying local configurations and a “Configuration**s**” block for importing external configurations.

Execution Variables

In some cases, there is a need to have variables with different values for each execution. In that case you can define execution variable while executing the job run as follows:

```
Java -jar ApplitoolsSimpleTestRunner.jar job.xml “-var[key1,value1]” “-var[key2,value2]” ...
```

Then, in your code extract the variable values with **exeVar[...]** or within an expression surrounded by a caret (e.g., variable called “**key”** can be read either by **exeVar[key]** or by **exp[^key^].**

Example:

If you execute the following command:

```
Java -jar ApplitoolsSimpleTestRunner.jar job.xml “-var[website,https://www.applitools.com]”
```

Then the results of the following action:

```
<action type="navigate">exeVar[website]</action>
```

Will be is if it were:

```
<action type="navigate">https://www.applitools.com</action>
```

Local Variables

Local variables are only accessible within their relevant scope.

A value of a local variable can only be read within an expression (marked as “exp[...]”) by attaching “%” to both ends of the variable name.

For example, assuming within the relevant scope a variable called “url” is defined, it can be read with the following code: `exp[%url%]`
Navigating to that url for example can be done with the following action:

```
<action type="navigate">exp[%url%]</action>
```

Action Blocks

In order to avoid multiplication of code, it is recommended to define **Action Blocks** in a separate file.
An action block is defined in an external definition file, as can be seen in the following example:

actionBlocks.xml

```
1 <definitions>
2       <login userName="tomsmith">
3               <action type="sendkeys" by="css" selector="#username">exp[%userName%]</action>
4               <action type="sendkeys" by="css" selector="#password">exp[%password%]</action>
5               <action type="click" by="css" selector="#login > button > i"></action>
6       </login>
7 </definitions>
```

In the example above, an Action Block with the name “login” is defined in line #2.
As can be seen in lines #3 and #4 the local variables **userName** and **password** are used within an expression. The value for these variables will be set when calling this action block.
A default value can be set (but not required) to a variable within the defining tag of the block - As can be seen in line #2 the default value “**tomsmith**” is set to the variable “**userName**”.

In order to use action blocks, the actionBlocks.xml file needs to be loaded, as highlighted in yellow below:

```
<job>
	<load>actionBlocks.xml</load>
	<batch name="BatchName">
		<test appname="AppName" testname="TestName">
			<configurations groups="locally" />
			<execute>
				<action type="navigate">http://the-internet.herokuapp.com/login</action>
				<login userName="john.smith@applitools.com" password="12345" />
				<action type="checkWindow">step</action>
			</execute>
		</test>
	</batch>
</job>
```

The call for the **login** block is highlighted in green, in the example above. Variables that are passed in this call will override any default values that were set during the definition of the action block.

Advanced Uses

Running on Applitools Ultrafast Grid

Applitools allow the rendering and visually validating of your current DOM across many environments during a single run through Applitools Ultrafast Grid.

In order to run on your Ultrafast Grid, in your configuration file, simply add the Visual Grid block. Applitools will do the rest!
You can find supported browser types in this link as well as supported device emulators in this link.

```
<definitions>
	<ChromeOptions name="headless">
		<argument>--headless</argument>
```

```
        </ChromeOptions>
        <configuration name="grid">
                <applitools>applitoolsConf.xml</applitools>
                <driver useChromeOptions="headless">chrome</driver>
                <viewport width = "1000" height = "600" ></viewport>
                <visualGrid> <!-- optional attribute: disableBrowserFetching="false" will attempt to download resources through the
browser directly -->

                        <browser type="chrome" width="1300" height="600" />
                        <browser type="chrome" width="1100" height="600" />
                        <browser type="chrome" width="980" height="600" />
                        <browser type="chrome" width="690" height="600" />

                        <browser type="firefox" width="1300" height="600" />
                        <browser type="firefox" width="1100" height="600" />
                        <browser type="firefox" width="980" height="600" />
                        <browser type="firefox" width="690" height="600" />

                        <browser type="IE_11" width="1300" height="600" />
                        <browser type="IE_11" width="1100" height="600" />
                        <browser type="IE_11" width="980" height="600" />
                        <browser type="IE_11" width="690" height="600" />

                        <chromeDeviceEmulation name="iPhone_X" />
                        <chromeDeviceEmulation name="iPhone_X"  orientation="LANDSCAPE"/>
                        <iosDevice name="iPad_7" />
                        <iosDevice name="iPad_7" orientation="LANDSCAPE" />

                </visualGrid>
        </configuration>
</definitions>
```

Running on a remote selenium server

1) Create a settings file as follows:

driverConf.xml

`<settings>`	
`<url>http://userName:pwd@remotelocation.com:80/wd/hub</url>`	The URL of the remote selenium server.

`<capabilities>`
`<browserName>chrome</browserName>`
`<platform>Windows 10</platform>`
`<version>46.0</version>`
`</capabilities>`
	Capabilities for the driver.

`<Proxy>`
`<proxyHost>localhost</proxyHost>`
`<proxyPort>8080</proxyPort>`
`<proxyUserDomain>proxyUserDomain</proxyUserDomain>`
`<proxyUser>proxyUser</proxyUser>`
`<proxyPassword>proxyPassword</proxyPassword>`
`</Proxy>`
	If a proxy is needed to instantiate the driver, add this block with the relevant values.

`</settings>`	

2) In the relevant configuration block replace:

```
<driver>chrome</driver>
```

With:

```
<driver>driverConf.xml</driver>
```

Executing the same test with multiple configurations

Add as many configuration blocks as you like and the test will execute in all of them!

For example, in order to run the same test on both **Chrome** and **Firefox** try running the following **job.xml** file:

```
<job>
	<batch name="my batch">
		<test appname="myapp" testname="mytest">
			<configuration>
				<applitools>applitoolsConf.xml</applitools>
				<driver>chrome</driver>
				<viewport width="600" height="600"></viewport>
			</configuration>
			<configuration>
				<applitools>applitoolsConf.xml</applitools>
				<driver>firefox</driver>
				<viewport width="600" height="600"></viewport>
			</configuration>
			<execute>
				<action type="navigate">http://the-internet.herokuapp.com/login</action>
				<action type="sendkeys" by="css" selector="#username">john.smith@applitools.com</action>
				<action type="sendkeys" by="css" selector="#password">12345</action>
				<action type="click" by="css" selector="#login > button > i"></action>
				<action type="checkwindow">step name</action>
			</execute>
		</test>
	</batch>
</job>
```

Also, try adding multiple viewport sizes to run on, for responsive design testing.
For example, the following test will run twice: once with a viewport of **600x600** and once with the viewport of **1024x600**.

```
<job>
	<batch name="my batch">
		<test appname="myapp" testname="mytest">
			<configuration>
				<applitools>applitoolsConf.xml</applitools>
				<driver>firefox</driver>
				<viewport width="600" height="600"></viewport>
				<viewport width="1024" height="600"></viewport>
			</configuration>
			<execute>
				<action type="navigate">http://the-internet.herokuapp.com/login</action>
				<action type="sendkeys" by="css" selector="#username">john.smith@applitools.com</action>
```

```
                        <action type="sendkeys" by="css" selector="#password">12345</action>
                        <action type="click" by="css" selector="#login > button > i"></action>
                        <action type="checkwindow">step name</action>
                    </execute>
                </test>
        </batch>
</job>
```

Grouping tests under the same batch

In order to group several tests under the same Applitools Batch - Simply add another “test” block to the batch, as follows:

```
<job>
        <batch name="my batch">
                <test appname="myapp" testname="mytest">
                        <configuration>
                                <applitools>applitoolsConf.xml</applitools>
                                <driver>chrome</driver>
                                <viewport width="600" height="600"></viewport>
                        </configuration>
                        <execute>
                                <action type="navigate">http://the-internet.herokuapp.com/login</action>
                                <action type="sendkeys" by="css" selector="#username">john.smith@applitools.com</action>
                                <action type="sendkeys" by="css" selector="#password">12345</action>
                                <action type="click" by="css" selector="#login > button > i"></action>
                                <action type="checkwindow">step name</action>
                        </execute>
                </test>
                <test appname="myapp" testname="mytest2">
                        <configuration>
                                <applitools>applitoolsConf.xml</applitools>
                                <driver>chrome</driver>
                                <viewport width="600" height="600"></viewport>
                        </configuration>
```

```
                    <execute>
                        <action type="navigate">http://the-internet.herokuapp.com/login</action>
                        <action type="sendkeys" by="css" selector="#username">john.smith@applitools.com</action>
                        <action type="sendkeys" by="css" selector="#password">1234567890</action>
                        <action type="click" by="css" selector="#login > button > i"></action>
                        <action type="checkwindow">step name</action>
                    </execute>
                </test>
        </batch>
</job>
```

Integrate with Applitools Jenkins Plugin

Step 1:
Follow the instructions here.

Step 2:
Execute the following command from Jenkins:
Java -jar ApplitoolsSimpleTestRunner.jar job.xml

(Yes! Exactly the same as before. The “Applitools Simple Test Runner” will recognize you are running from Jenkins with the Applitools Jenkins plugin enabled and will automatically integrate with it).

Note: when running from a CI, with an Applitools Plugin enabled, all the defined batches in the job will be merged together into a single batch.

Downloading Results

In order to download the test results add the following blocks to the configuration section of the test:

Command	Description
`<download type="baselines">/Users/name/Downloads</download>`	Download the Baseline images to the specified path
`<download type="currents">/Users/name/Downloads</download>`	Download the Current images to the specified path
`<download type="diffs">/Users/name/Downloads</download>`	Download the Current images, with the highlighted differences on them, to the specified path
`<download type="json report" name="chrome">/Users/name/Downloads/</download>`	Download a JSON file test results report

E.g, The following configuration block will download:

1) The Baseline images of the test to: "*/Users/name/Downloads"*
2) The Current images of the test to: "*/Users/name/Downloads"*
3) The diff images to both: "*/Users/name/Downloads"* AND "*/Users/name/Downloads2"*

```
<configuration>
       <applitools>applitoolsConf.xml</applitools>
       <driver>chrome</driver>
       <viewport width="600" height="600"></viewport>
       <download type="baselines">/Users/name/Downloads</download>
       <download type="currents">/Users/name/Downloads</download>
       <download type="diffs">/Users/name/Downloads</download>
       <download type="diffs">/Users/name/Downloads2</download>
</configuration>
```

Labels and "If" Statements

In some cases, there is a need to use "IF" statements in order to avoid multiplication of code.

For example, when running the same test on multiple viewport sizes, a different selector needs to be clicked to perform an action (Eg., a visible link, becomes hidden inside a menu, for a smaller viewport and requires an extra click to get to it).

For that, it is possible to “label” a configuration block (a driver, a viewport, an Applitools configuration or/and an entire configuration block) as follows:

```
<configuration labels="configuration1">
       <applitools>applitoolsConf.xml</applitools>
       <driver labels="driver1 chromeDriver myDriver">chrome</driver>
       <viewport width="600" height="600" labels="viewport1"></viewport>
       <viewport width="800" height="600" labels="viewport2"></viewport>
</configuration>
```

Note: You can add to a block as many labels as you like, separated by a space, as shown in the **driver** block above and highlighted in yellow.

Once you labeled some configurations, use the “**if_labeled**” and “**if_not_labeled**” attributes on commands in the execution block. For example:

```
<execute>
       <action type="navigate">http://the-internet.herokuapp.com/login</action>
       <action type="sendkeys" by="css" selector="#username">john.smith@applitools.com</action>
       <action type="sendkeys" by="css" selector="#password">12345</action>
       <action type="click" by="css" selector="#login > button > i"></action>
       <action type="checkwindow" if_labeled="viewport1">Step Name When Viewport1</action>
       <action type="checkwindow" if_labeled="viewport2" if_not_labeled="chromeDriver">Step Name When Viewport2 & Not Chrome</action>
</execute>
```

In cases where an “If” statement is needed in order to decide whether an action/set of actions should be performed during the test, an “if” block should be used as follows:

```
<if value="some_value" equals="some_other_value">
```

```
        <action.....>
        <action.....>
</if>
```

```
<if value="some_value" not_equals="some_other_value">
        <action.....>
        <action.....>
</if>
```

```
<if value="some_value" contains="some_other_value">
        <action.....>
        <action.....>
</if>
```

Test Tagging

When having a large test suite, it's sometimes useful to be able to run only a subset of it.
For that purpose, you can tag your tests, in the job.xml file as follows - you can add as many tags as you like, separated by a **space** character:

```
<test appname="Applitools" testname="Home Page" tags = "desktop mobile">
```

Then, when running your execution command, specify the tags you would like to execute and only these will run, as follows:

```
Java -jar ApplitoolsSimpleTestRunner.jar Job.xml -runTags[mobile,desktop]
```

Notice that you can specify as many tags as needed to run, separated by a **comma.**

If no tags are specified, the entire test suite will run.

Enforcing Tests Execution Order

In some situations a subset of tests must finish as a prerequisite for other tests to start.

For that purpose, you can define the predecessors of a test, in the job.xml file as follows:

```
<test appname="Applitools" testname="Home Page" predecessors = "tag1 tag2">
```

The above configuration will make sure this test will not start until **all** tests with tags “tag1” and “tag2” have finished. See more about test tagging in the previous section.

Concurrent Runs

In order to run concurrently, pass a “concurrent_runs” argument to the “Job” block opener.
You can also pass “ufgConcurrency” attribute to configure the concurrency on Applitools Ultrafast Grid.
For example, in order to have 3 concurrent runs, set it as follows:

```
<job concurrent_runs="3" ufgConcurrency="3"> <!--ufg concurrency is only relevant if running with Applitools UFG-->
```

For **this** example, all the tests (in all configurations and viewport sizes) in this job will run concurrently - no more than **3** at a time.

Batch Completion & Notification

In order to receive batch notifications on completion, when configuration your batch, add the notifyOnCompletion = "true" attribute, as follows:

```
<batch name="My Batch" notifyOnCompletion = "true">
```

At the end of your test suite, the batch instances that were used during the run will be closed. That means no more tests could be sent to those instances, and an attempt to do so will create a new separate batch instance in your dashboard. Sometimes, when executing several jobs separately, if those are intended to be sent to the same batch instance, you may want to prevent the Applitools batch from closing.
In such situations you may add the `dontClose = "true"` attribute to the definition of the batch, as follows:

```
<batch name="My Batch" dontClose = "true">
```

Applitools Customized Properties

Customized properties provide you the ability to set custom labels for each one of your tests. The Benefit of this feature is to provide a convenient way to group test results on the Applitools dashboard based on a label or multiple labels. For example, you might have tests tagged as Critical or Smoke. You might want to set a custom label to group these particular labels on the dashboard for convenient display.

In order to add a property, simply add it to the relevant test block in your job.xml file, as follows:

```
<test appname="..." testname="...">
        <property name="propName1" value="propValue1" />
        <property name="propName2" value="propValue2" />
        <configurations groups="..." />
        <execute>
                <action type="..." />
        </execute>
</test>
```

Locator Groups

In some situations, when searching for an element there are several alternatives for the identifying selector and those are not always known in advance. For that reason, you have the ability to add a locator group. When looking for an element, simply define the “by” option as “locatorGroup” and ApplitoolsSImpleTestRunner will iterate over all the selectors in the group until a valid one is found.

In order to define a locator group, create a definitions file, for example definitions.xml with the following content will create two different locator groups: groupName1 and groupName2 with two locators each:

```
<definitions>
	<locatorGroup name="groupName1">
		<locator by="css" selector="<Selector1>" />
		<locator by="xpath" selector="<Selector2>" />
	</locatorGroup>
	<locatorGroup name="groupName2">
		<locator by="css" selector="<Selector3>"/>
		<locator by="css" selector="<Selector4>"/>
	</locatorGroup>
</definitions>
```

Then, in order to use a group, for example to click on a button, in the job file insert the following command:

```
<action type="click" by="locatorGroup" groupName="groupName1" />
```

Using Chrome and Firefox Options

In order to apply ChromeOptions or Firefox Options to your driver, first define the relevant block in your definitions file.

For ChromeOptions define the following:

<definitions> <ChromeOptions name="<Chrome Options Name>"> <argument>...</argument> </ChromeOptions> </definitions>	Supported Options: <argument>...</argument> <encodedExtension>...</encodedExtension> <extension>...</extension>

For FirefoxOptions define the following:

<definitions> <FirefoxOptions name="<Firefox Options Name>"> <argument>...</argument> </FirefoxOptions> </definitions>	Supported Options: <argument>...</argument>

Then, make sure to use the relevant chrome/firefox options name when you configure your driver.

For Chrome:

```
<driver useChromeOptions="<Chrome Options Name>">chrome</driver>
```

For Firefox:

```
<driver useFirefoxOptions="<Firefox Options Name>">firefox</driver>
```

Important Comments

- You can generate an example project by opening the terminal and running the command:

Java -jar ApplitoolsSimpleTestRunner.jar

Note: The project will be generated in the location of the ApplitoolsSimpleTestRunner.Jar file, so it is recommended to have it located in an empty folder.

- If your ChromeDriver/GeckoDriver is not in your PATH, simply add a location in the “driver” block as follows:

```
<driver location="PATH_TO_DRIVER">chrome</driver>
```

- Some special characters are not allowed in XML files (such as <, >, etc) and require a conversion before being inserted to the code. There are many converters on the web - this is a good one.

  For example, the following css selector:
  body > div.position-relative.js-header-wrapper > header > div > button > svg > path

  Would need to be converted to:

  body > div.position-relative.js-header-wrapper > header > div > button > svg > path

- Adding “**-narrate**” to the execution command will print out the actions that are executed.
  For exampe: **Java -jar ApplitoolsSimpleTestRunner.jar job.xml -narrate**

- Adding “**-applitoolsLogs**” to the execution command will print out the logs for the Applitools commands that are executed.
  For exampe: **Java -jar ApplitoolsSimpleTestRunner.jar job.xml -applitoolsLogs**

- Adding “-time” to the execution command will print out the time it took for each action to be performed.
  For exampe: **Java -jar ApplitoolsSimpleTestRunner.jar job.xml -time**

- In order to specify a specific Applitools Batch Id in the batch tab of your job file:

```
<batch name = "batch name" id="batchid12345">
```

- In recent versions of chrome a notification bar is added for automated tests saying: "**Chrome is being controlled by automated test software**". This can sometimes affect visual tests.
  In order to overcome this, simply add the following to your definition.xml file:

```
<definitions>
	<ChromeOptions name="NotificationDisabler">
		<argument>disable-infobars</argument>
	</ChromeOptions>
</definitions>
```

  Then, attach the chrome options element to your Chrome driver in your **job.xml** file, as follows:

```
<driver useChromeOptions="NotificationDisabler">chrome</driver>
```

- In order to execute your test on a Chrome mobile emulator, use the following configuration for the driver (remember to **not** set a viewportsize). The names for the supported devices can be taken from the Chrome Emulator:

```
<driver deviceName="iPhone X">chrome</driver>
```

- Please note the difference between a proxy configuration to instantiate a remote webdriver (in the driver configuration file) to the proxy settings for running Applitools visual tests (in the Applitools configuration file). The two are independent of one another.
- Anywhere that a file is specified (e.g., applitoolsConf.xml, configurations.xml, sitemap.xml…), if the file is not located in the current directory, ASTR will search for it in subdirectories. This allows flexibility with moving files around without having to change full paths in the actual files.
- The default driver type that is created in the background is an **AppiumDriver**. In case a **RemoteWebDriver** is needed, add the **forceRemoteDriver=”true”** attribute to the driver tag as follows:

```
<driver forceRemoteDriver="true">chrome</driver>
```

- When running with a local chromeDriver, it is possible to configure a fixed port by adding a chromedriverPort attribute as follows:

```
<driver chromedriverPort="<PORT_NUMBER>">chrome</driver>
```

Versions

* In order to download the version from the jfrog artifactory navigate to “Artifacts”->”Applitools-Simple-Test-Runner” and download the required version.

Date	Version Number	Additions	Applitools SDK Version	Selenium Version	Download Link
Mar 23 2022	3.58.1	• Bug fix: edge case when tests not batched correctly. • Generate Log file of collected URLs.			Download
Jan 24 2022	3.57.19	• Updated Eyes SDK Version	Eyes-selenium-java3 (3.211.0)	3.141.59	Download
Jan 5 2022	3.57.18	• Added ability to hide elements on the page.	Eyes-selenium-java3 (3.211.0-beta)	3.141.59	Download
Jan 3 2022	3.57.17	• Added ability to enforce test execution order. • Added support for SmartSplitting the crawling of a URL collection.	Eyes-selenium-java3 (3.211.0-beta)	3.141.59	Download
Dec 13 2021	3.57.16	• Added ability to close current window/tab when switching window context.	Eyes-selenium-java3 (3.210.4)	3.141.59	Download
Dec 2 2021	3.57.14	• Bug fix: “If” statement not working within function blocks is resolved. • Added ability to switch context to “alert” and perform	Eyes-selenium-java3 (3.208.2)	3.141.59	Download

		accept/dismiss. • Added support for selenium "Select".			
Oct 22 2021	3.57.12	• Added ability to not return to the top of the page at the end of a force page load action.	Eyes-selenium-java3 (3.208.2)	3.141.59	Download
Oct 14 2021	3.57.11	• Bug fix: issue with clicking an element resolved.	Eyes-selenium-java3 (3.208.2)	3.141.59	Download
Oct 12 2021	3.57.10	• Added support for locator groups.	Eyes-selenium-java3 (3.208.2)	3.141.59	Download
Sep 29 2021	3.57.9	• Added ability to grab links URLS into a collection and then iterating over them to perform actions.	Eyes-selenium-java3 (3.208.1)	3.141.59	Download
Sep 3 2021	3.57.8	• Bug fix - fixed issue with LayoutBreakpoints resizing not working.	Eyes-selenium-java3 (3.208.1)	3.141.59	Download
Aug 10 2021	3.57.7	• Bug fix - Issue with reading environment variables resolved.	Eyes-selenium-java3 (3.207.0)	3.141.59	Download
Jul 13 2021	3.57.6	• Added ability to implicitly wait for elements. • Added ability to delete all cookies. • Added execution command to get action performance time.	Eyes-selenium-java3 (3.206.0)	3.141.59	Download
Jun 27 2021	3.57.5	• Updated Eyes SDK version	Eyes-selenium-java3 (3.204.1)	3.141.59	Download

Jun 16 2021	3.57.4	• Added support for UFG disableBrowserFetching configuration • Added support for UFG Concurrency	Eyes-selenium-java3 (3.202.1)	3.141.59	Download
Jun 9 2021	3.57.2	• Added support for A/B testing, auto-maintenance (Variation Group Id).	Eyes-selenium-java3 (3.202.1)	3.141.59	Download
Apr 6 2021	3.57.1	• Bug fix regarding attribute Expression parsing. • Added support for specifying the ScrollRootElement in the check action.	Eyes-selenium-java3 (3.198.0)	3.141.59	Download
Mar 2 2021	3.56.0	• Added support for Applitools Accessibility Contrast Advisor	Eyes-selenium-java3 (3.194.0)	3.141.59	Download
Feb 5 2021	3.55.10	• Updated Eyes SDK version	Eyes-selenium-java3 (3.192.0)	3.141.59	Download
Jan 29 2021	3.55.9	• Added support for adding UFG option in the check action.	Eyes-selenium-java3 (3.191.4)	3.141.59	Download
Jan 28 2021	3.55.8	• Added support for FirefoxOptions • Fixed issue with automatically deleting UFG tests when they finish.	Eyes-selenium-java3 (3.191.4)	3.141.59	Download
Jan 26 2021	3.55.6	• Added support for beforeRenderScreenshotHook in the “check” action. • forcePageLoad not scrolls back	Eyes-selenium-java3 (3.191.1)	3.141.59	Download

		to the top after loading the page.			
Jan 19 2021	3.55.5	• Added support for additional UFG devices.	Eyes-selenium-java3 (3.191.0)	3.141.59	Download

Jan 18 2021	3.55.4	• Bug fixes:
		  ○ Fixed batch closure of classic (none-UFG) runs.
		  ○ Fixed running multiple tests with UFG.
			Eyes-selenium-java3 (3.190.5)	3.141.59	Download

Jan 4 2021	3.55.3	• Bug fix in the UFG layout breakpoint support.	Eyes-selenium-java3 (3.190.3)	3.141.59	Download
Jan 3 2021	3.55.0	• Added support for UFG layout breakpoints.	Eyes-selenium-java3 (3.190.2)	3.141.59	Download
Nov 23 2020	3.54.3	• Updated Eyes SDK version.	Eyes-selenium-java3 (3.185.0)	3.141.59	Download
Oct 13 2020	3.54.2	• Bug fix: memory leak when using Applitools Ultrafast Grid was fixed. • Added dontClose attribute to prevent Applitools batches from closing at the end of the test.	Eyes-selenium-java3 (3.180.0)	3.141.59	Download
Aug 23 2020	3.53.0	• Added support for IOS devices on Applitools' UltraFast Grid. • Added support for landscape orientation for devices on Applitools' UltraFast Grid.	Eyes-selenium-java3 (3.174.0)	3.141.59	Download
Jul 31 2020	3.52.0	• Added support for batch notification on completion.	Eyes-selenium-java3 (3.171.1)	3.141.59	Download

Jul 31 2020	3.51.0	• Check action now supports setting a Match Level. • Tests can now be tagged and executed by tag.	Eyes-selenium-java3 (3.171.1)	3.141.59	Download
Jul 16 2020	3.50.1	• Bug fix when adding ignore/layout regions to check method.	Eyes-selenium-java3 (3.169.0)	3.141.59	Download
Jul 9 2020	3.50.0	• Added action to execute OS commands. • Added ability to apply multiple coded regions with "check" command.	Eyes-selenium-java3 (3.169.0)	3.141.59	Download
Jul 6 2020	3.49.0	• Bug fixes regarding DOM capturing.	Eyes-selenium-java3 (3.169.0)	3.141.59	Download
Jun 24 2020	3.48.0	• Added support for cookie addition.	Eyes-selenium-java3 (3.167.0)	3.141.59	Download
Jun 15 2020	3.47.5	• Added support for useDom and enablePatterns Layout Flags in Check action.	Eyes-selenium-java3 (3.165.0)	3.141.59	Download
May 26 2020	3.47.4	• Eyes SDK fixes.	Eyes-selenium-java3 (3.164.0)	3.141.59	Download
May 14 2020	3.47.3	• Bug fix: allowing recursive directory structure for remote driver files.	Eyes-selenium-java3 (3.162.0)	3.141.59	Download
May 7 2020	3.47.2	• Bug fix: Using remote webdriver with a proxy.	Eyes-selenium-java3 (3.160.3)	3.141.59	Download

Apr 15 2020	3.47.1	• Minor improvements.	Eyes-selenium-java3 (3.160.3)	3.141.59	Download
Mar 21 2020	3.47.0	• Bug fixes - solved “Empty Test” issue when running on the Ultrafast Grid.	Eyes-selenium-java3 (3.160.2)	3.141.59	Download
Feb 27 2020	3.46.0	• Added support for coded Layout Region • Added ability to run tests with the Ignore Displacement Diffs cleanup algorithm.	Eyes-selenium-java3 (3.160.2)	3.141.59	Download
Jan 31 2020	3.45.0	• Support for Eyes server 10.9 - any version below this version will not be supported on Applitools version 10.9 and above.	Eyes-selenium-java3 (3.160.0)	3.141.59	Download
Jan 28 2020	3.44.3	• Visual Grid options added	Eyes-selenium-java3 (3.159.0)	3.141.59	Download
Nov 22 2019	3.44.2	• Bug Fixes with Dom Snapshot	Eyes-selenium-java3 (3.158.8)	3.141.59	
Nov 12 2019	3.44.1	• Bug Fixes (fixed hanging status at the end of the run).	Eyes-selenium-java3 (3.158.5)	3.141.59	
Oct 21 2019	3.44.0	• Bug Fixes	Eyes-selenium-java3 (3.158.3)	3.141.59	
Aug 27 2019	3.43.0	• Bug Fixes	Eyes-selenium-java3 (3.157.7)	3.141.59	
Jun 7	3.41.0	• Added support for fixed	Eyes-selenium-java3 (3.152.6)	3.141.59	

2019		chromeDriver port.			
April 29 2019	3.38.0	• Bug fixes. • Visual Grid Support.	Eyes-selenium-java3 (3.151.2)	4.0.0-alpha	
	3.36.0	• Support for Chrome mobile emulators	Eyes-selenium-java3 (3.150.1)	3.141.59	
	3.35.0	• Bug fixes. • Support for RCA feature.	Eyes-selenium-java3 (3.141.2)	3.141.59	
	3.34.0	• Improved experience regarding Image upload handling. • Added "Check" action with fluent interface.	Eyes-selenium-java3 (3.35)	3.11.0	
	3.33.2	• Support for Java 10	Eyes-selenium-java3 (3.35)	3.11.0	
	3.33	• Bug Fixes	Eyes-selenium-java3 (3.34)	3.11.0	
	3.32	• Bug fixes - Json report is now created for each test.	Eyes-selenium-java3 (3.32.1)	3.11.0	
	3.31	• Added ability to override Applitools StitchOverlap. • Bug fixes.	Eyes-selenium-java3 (3.32.1)	3.11.0	
	3.30	• Added MouseOver Action	Eyes-selenium-java3 (3.32.1)	3.11.0	
	3.29	• Added ability to define separate tests for check commands (included deletion of test runs at finish, overriding baselines and change of match level).	Eyes-selenium-java3 (3.32.1)	3.11.0	

	3.28.2	• Added support for running on Edge browser locally.	Eyes-selenium-java3 (3.31)	3.11.0	
	3.28.1	• Added support for running on Internet Explorer browser locally. • Added ability for "If" Statements.	Eyes-selenium-java3 (3.31)	3.11.0	
	3.26	• Bug fixes.	Eyes-selenium-java3 (3.25)	3.9.1	
	3.25	• Added mechanism for self-healing during sitemap crawling, in case of intermittent errors.	Eyes-selenium-java3 (3.25)	3.9.1	
	3.24	• Added ability to upload a image from local storage. • Added support for flexible directory structure (see important comments section).	Eyes-selenium-java3 (3.25)	3.9.1	
	3.23	• Added ability to perform actions on returned Elements from JS execution.	Eyes-selenium-java3 (3.25)	3.8.1	
	3.22	• Reverted version of Applitools Eyes SDK, to avoid a bug with image crop.`	Eyes-selenium-java3 (3.25)	3.8.1	
	3.21	• Bug Fixes.	Eyes-selenium-java3 (3.29)	3.8.1	
	3.20	• Ability to set screenshot rotation, if needed. • Added ability to click "RETURN" at the end of a sendKeys action.	Eyes-selenium-java3 (3.29)	3.8.1	

	3.19	• Added ability to switch between browser tabs/windows.	Eyes-selenium-java3 (3.28)	3.7.1	
	3.18	• Improved Project Generators	Eyes-selenium-java3 (3.28)	3.7.1	
	3.17.1	• Added ability to prevent stitch content while checking a region. • Hide scroll bars is now the default.	Eyes-selenium-java3 (3.28)	3.7.1	
	3.17	• Added ability to define reusable action blocks.	Eyes-selenium-java3 (3.26)	3.6.0	
	3.16.1	• Bug fixes.	Eyes-selenium-java3 (3.26)	3.6.0	
	3.16	• Added the ability to store a returned string value, from a Javascript command execution, as a local variable.	Eyes-selenium-java3 (3.26)	3.6.0	
	3.15	• SmartSplit test when crawling a sitemap.xml file.	Eyes-selenium-java3 (3.26)	3.6.0	
	3.14.2	• Added ability to switch context to a mobile context.	Eyes-selenium-java3 (3.24)	3.6.0	
	3.14.1	• Upgraded Selenium and Eyes versions.	Eyes-selenium-java3 (3.23)	3.6.0	
	3.14	• Improved abilities to read variables. • Bug fixes.	Eyes-selenium-java3 (3.19)	3.5.3	
	3.13	• Added ability to switch context.	Eyes-selenium-java3 (3.19)	3.5.3	

	3.12	• Bug fixes.	Eyes-selenium-java3 (3.19)	3.5.3	
	3.11	• Added "onError" handling. • Added options for browser back, forward & refresh. • Added option for execution variables. • Added ability to iterate elements.	Eyes-selenium-java3 (3.20)	3.5.3	
	3.10	• Additional Crawling capabilities. • Added Functions. • Bug fixes.	Eyes-selenium-java3 (3.20)	3.5.3	
	3.9	• Added ability to check an IFrame.	Eyes-selenium-java3 (3.20)	3.5.3	
	3.8	• Added support for Applitools Test Properties. • Bug fixes.	Eyes-selenium-java3 (3.18)	3.5.2	
	3.73	• Added support Proxy settings for instantiating a remote selenium server.	Eyes-selenium-java3 (3.18)	3.5.2	
	3.72	• Bug fixes.	Eyes-selenium-java3 (3.16)	3.4.0	
	3.7	• Added ability to crawl by domains. • Added ability to add configurations and configuration groups in an external definition file. Note: A known bug is if no checkWindow/checkRegion	Eyes-selenium-java3 (3.11)	3.4.0	

		actions are a part of the execution block, **GoManual** and **Crawl** actions will not work. This is fixed in version 3.72.			
	3.6	• Crawling in separate tests. • Additional options for generating projects and files. • Added options for logs. • Added an option to configure Chrome options for a Chromedriver. Note: A known bug is if no checkWindow/checkRegion actions are a part of the execution block, **GoManual** and **Crawl** actions will not work. This is fixed in version 3.72.	Eyes-selenium-java3 (3.13)	3.4.0	
	3.5	• Bug Fixes	Eyes-selenium-java3 (3.11)	3.4.0	
	3.4	• Added an action for drag&drop. • Improved options for generating projects and files.	Eyes-selenium-java3 (3.9)	3.1.0	
	3.3	• Added the ability to download a test results JSON file.	Eyes-selenium-java3 (3.9)	3.1.0	
	3.2	• Added the ability to crawl through all links in given elements.	Eyes-selenium-java3 (3.9)	3.1.0	
	3.1	• Added the ability to go to manual navigating state.	Eyes-selenium-java3 (3.9)	3.1.0	

	3	• Bug fixes. • Added the ability to generate an example project.	Eyes-selenium-java3 (3.9)	3.1.0	
	2.6	• Downgraded Selenium version to avoid a bug in version 3.3.1.	Eyes-selenium-java3 (3.9)	3.1.0	
	2.5	• Added: action for forcing a page to load.	Eyes-selenium-java3 (3.9)	3.3.1	
	2.4	• Added: action for explicit wait. • Added: Support for predefined variables.	Eyes-selenium-java3 (3.9)	3.3.1	
	2.3	• Added: ability to crawl through selected indexes in a Sitemap.xml file.	Eyes-selenium-java3 (3.9)	3.3.1	
	2.2	• Added: ability for Concurrent Runs • Bug fixes.	Eyes-selenium-java3 (3.9)	3.3.1	
	2.1	• Added: ability for reading from Environment Variables. • Added: ability for "IF" statements according to labels.	Eyes-selenium-java3 (3.9)	3.3.1	

Appendices

Appendix A - Example for a sitemap.xml file

sitemap.xml

```
<sitemap>
     <loc>https://applitools.com/</loc>
     <loc>https://applitools.com/features/</loc>
     <loc>https://applitools.com/pricing/</loc>
     <loc>https://applitools.com/contact-us/</loc>
</sitemap>
```