## Windows Application Driver

¿Qué es Windows Application Driver?

Windows Application Driver (WinAppDriver) es una herramienta de uso gratuito que permite pruebas de automatización de interfaz gráfica tipo Selenium para windows, además, puede ejecutarse individualmente y como complemento para Appium. Esta herramienta solo permite pruebas en aplicaciones de Universal Windows Platform (UWP), Windows Forms (WinForms), Windows Presentation Foundation (WPF) y en Classic Windows (Win32) y sólo funciona en sistemas operativos con Windows 10.

Pros al utilizar WinAppDriver

* Es desarrollado por Microsoft, por tanto tiene mucho sentido probar aplicaciones de Windows con una herramienta de la misma compañía, además esta herramienta se actualiza por medio de un repositorio en GitHub.
* Utiliza el protocolo WebDriver, lo que significa que si se tienen conocimientos previos en automatización web / móvil y conoce como desarrollar usando el framework WebDriver, entonces la transición a WinAppDriver será fácil, rápida y clara.
* Se puede usar XPath y otras propiedades para localizar elementos de la interfaz de usuario de Windows.
* Los tests se pueden ejecutar con los lenguajes C#, Java, JavaScript, Python y Ruby.
* Se pueden utilizar los patrones de diseño POM y Screenplay.

Contras al utilizar WinAppDriver

* Aunque WinAppDriver sea una herramienta de uso gratuito, ésta no es de código abierto, pues en el repositorio WinAppDriver sólo han sido publicadas muestras y pruebas que sí son de código abierto.
* Ya que WinAppDriver sólo funciona con aplicaciones para Windows las cuales son de tipo UWP, WinForms, WPF y Win32, significa que excluye muchas otras aplicaciones que son de otras plataformas para Windows.
* Solo puede ser ejecutado en el sistema operativo Windows 10.
* No trae incorporada alguna herramienta para buscar las propiedades de la interfaz en una aplicación, para buscar las propiedades se deben usar herramientas como inspect.exe.
* WinAppDriver tiene métodos muy limitados en cuanto a los lenguajes de programación a utilizar para hacer las pruebas automatizadas, el lenguaje en el que se encuentra un mayor repertorio de métodos y funciones para realizar las automatizaciones es **C#**.

Requerimientos básicos para esta sección

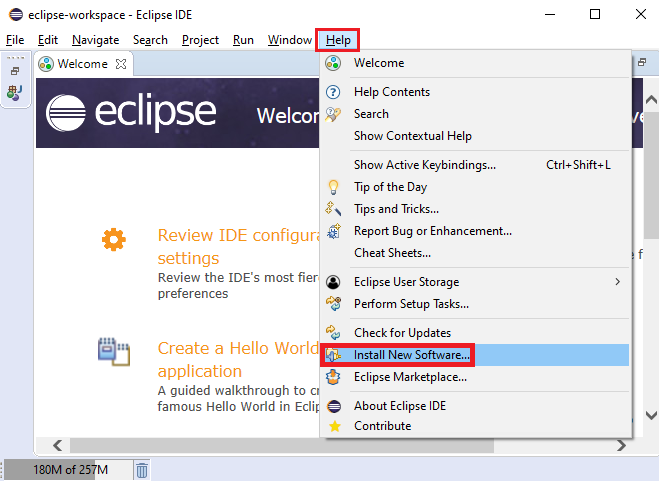
* Eclipse
* Windows 10
* JavaSE 1.8 o posterior
* [Maven](https://maven.apache.org/download.cgi) -
* Permisos de administrador del sistema.
* Modo desarrollador de Windows activado.

* [Windows Kits](https://developer.microsoft.com/es-es/windows/downloads/windows-10-sdk)
* [WinAppDriver](https://github.com/Microsoft/WinAppDriver/releases)

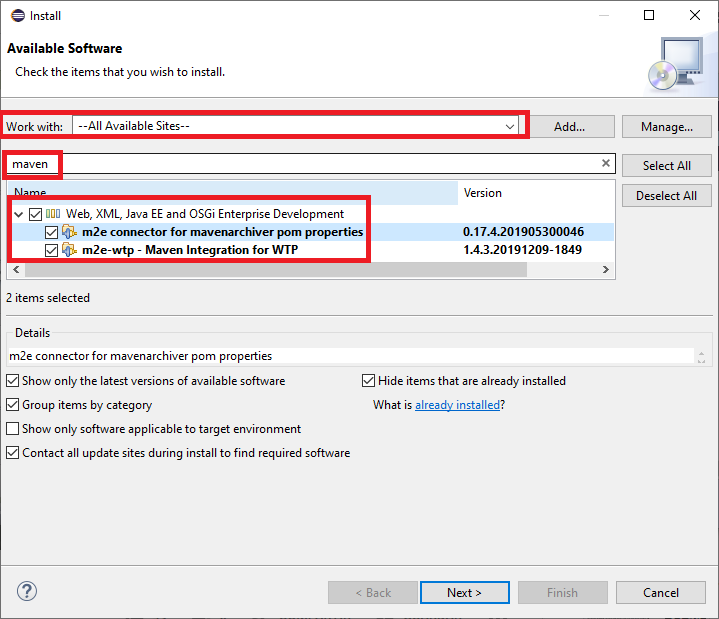
Configuraciones previas a la instalación de WinAppDriver

* Instalar Maven

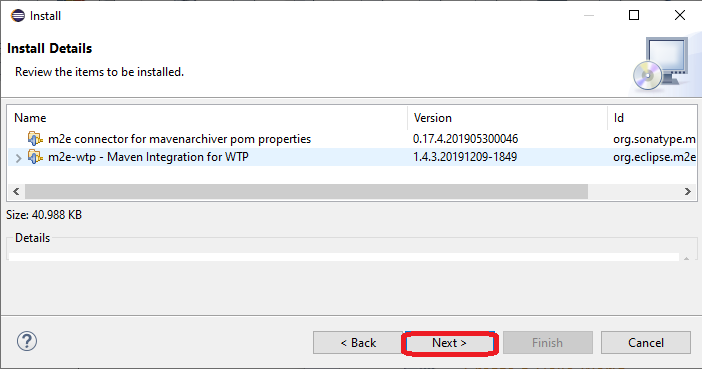
Como en este proyecto se va a trabajar con librerías provenientes de distintos repositorios, se precisa de maven. Para instalar maven hay que entrar a eclipse, e instalar dicha dependencia, ingresando a esta como se muestra en la siguiente imagen.



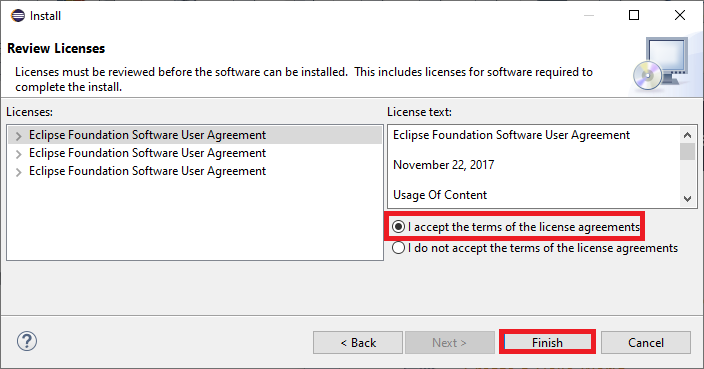
Posterior a esto, se abrirá la ventana para hacer instalaciones, entonces en la caja que dice “Work with”, se selecciona la opción All Available Sites.., y hecho esto, inmediatamebte abajo hay un cuadro de texto en blanco, en el cual se deberá de escribir “maven” para buscar esta herramienta. Una vez encontrada la herramienta aparecerá abajo del cuadro de texto donde se podrá apreciar la el nombre del módulo de maven y sus versiones al lado del nombre, esto significa que el módulo fue encontrado y procedemos a seleccionar todas sus dependencias. Hecho esto, hacemos clic a next.



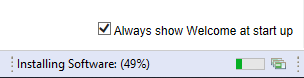
Después de esto, se nos dirige a una pantalla donde notifica los de talles, de la instalación que se va a hacer.



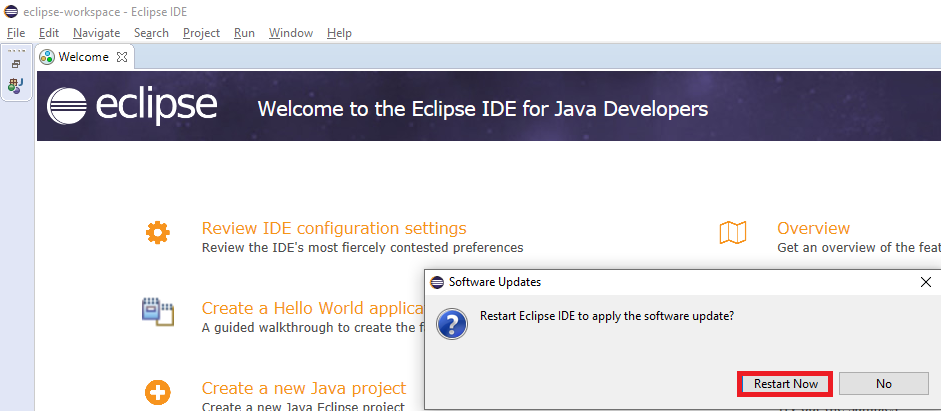
Para empezar la instalación, se aceptan términos y condiciones y oprimimos finalizar.



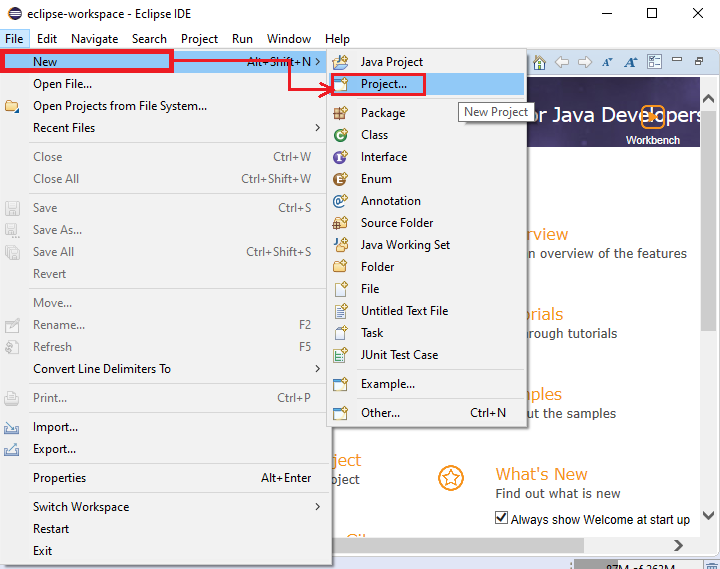
Esperamos a que la barra de carga del lado inferior derecho de eclipse cargue, lo que nos indicara que maven fue instalado



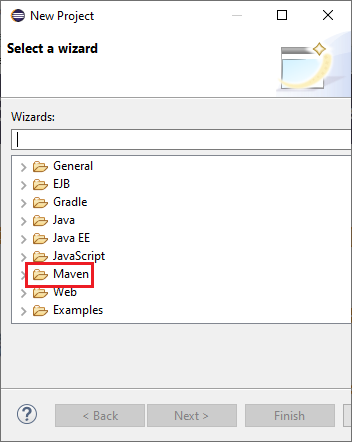
Una vez instalado maven, eclipse pedirá reiniciado, a lo que responderemos oprimiendo el botón restart now.



Para probar que haya quedado instalado este programa ingresamos a la siguiente ruta…

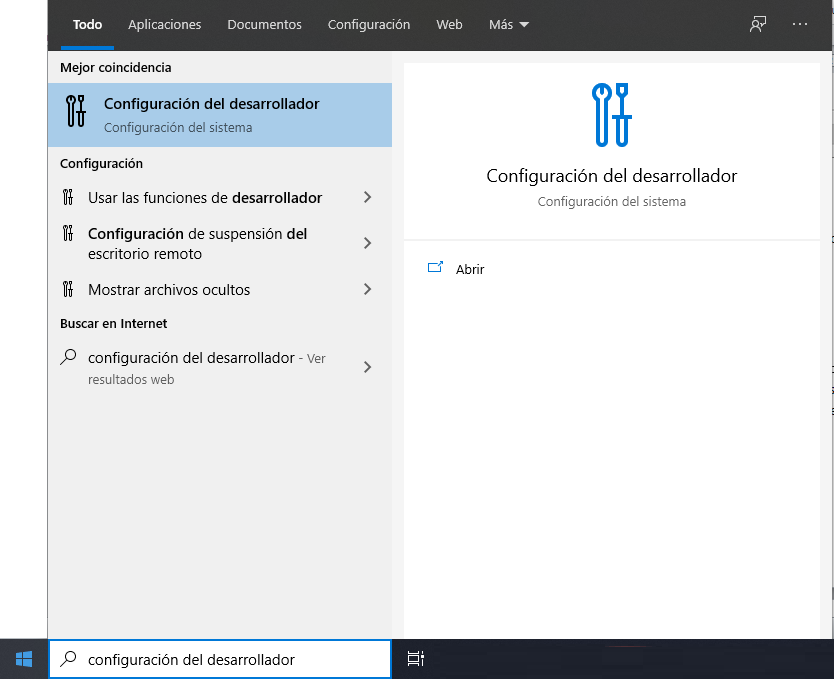


Como se puede apreciar en el siguiente cuadro, maven quedó instalada correctamente,

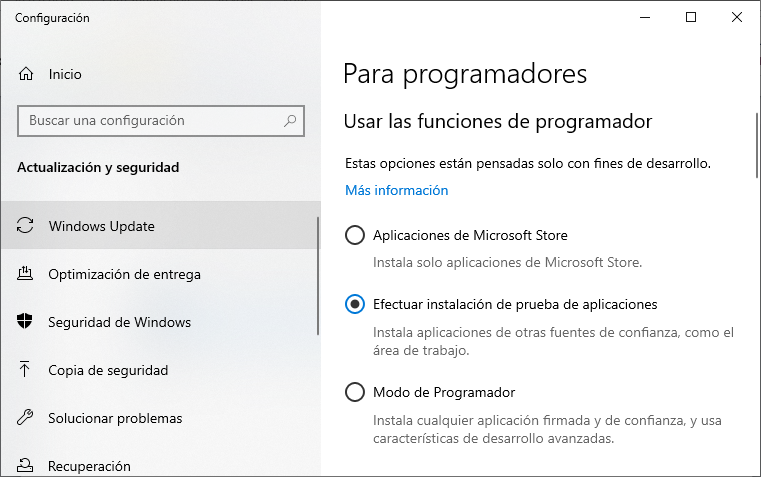


* Activar modo desarrollador en Windows 10

Para activar el modo desarrollador de Windows 10, escribimos en la barra de búsqueda de Windows 10 “Configuración del desarrollador”.



Una vez hagamos clic en esas configuraciones, se nos deberá abrir el siguiente panel…



A continuación, se selecciona la opción de Modo de Programador, lo cual abrirá una ventana donde mostrará una advertencia de lo que implica usar el modo programador, después de leer esto, oprimimos sí para activar esta opción.



Después de oprimir si, estas opciones de desarrollador quedarán habilitadas

* Instalar Windows Development Kits

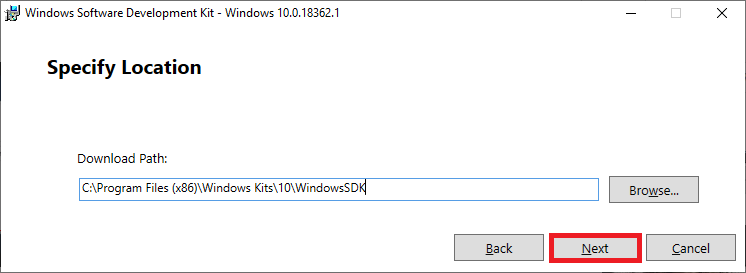
Como se necesita una herramienta para poder identificar los nombres de los identificadores que se necesitan para realizar el mapeo de los objetos para realizar la prueba, se necesita de un programa llamado inspect, el cual hace parte de este paquete de herramientas.

Para empezar, se debe entrar a la página de descarga de [Windows Development Kits](https://developer.microsoft.com/es-es/windows/downloads/windows-10-sdk) y descargar el instalador.

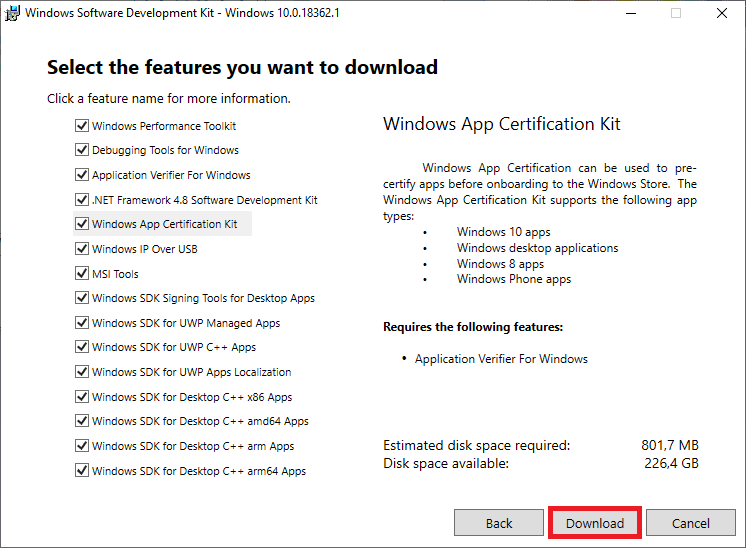


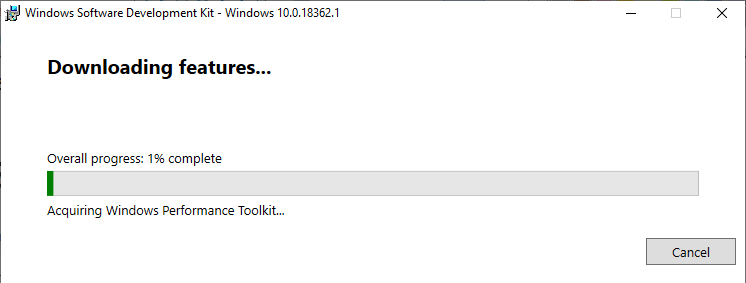
Una vez descargado el instalador, cliqueamos el archivo descargado para empezar la instalación. Cuando se abra el recuadro del instalador, oprimimos continuar, y después se elige el directorio en el cual se desea almacenar el kit de herramientas, que en este caso se dejará el directorio predeterminado.





Hecho esto, procedemos a descargar las herramientas de Windows Development Kit.



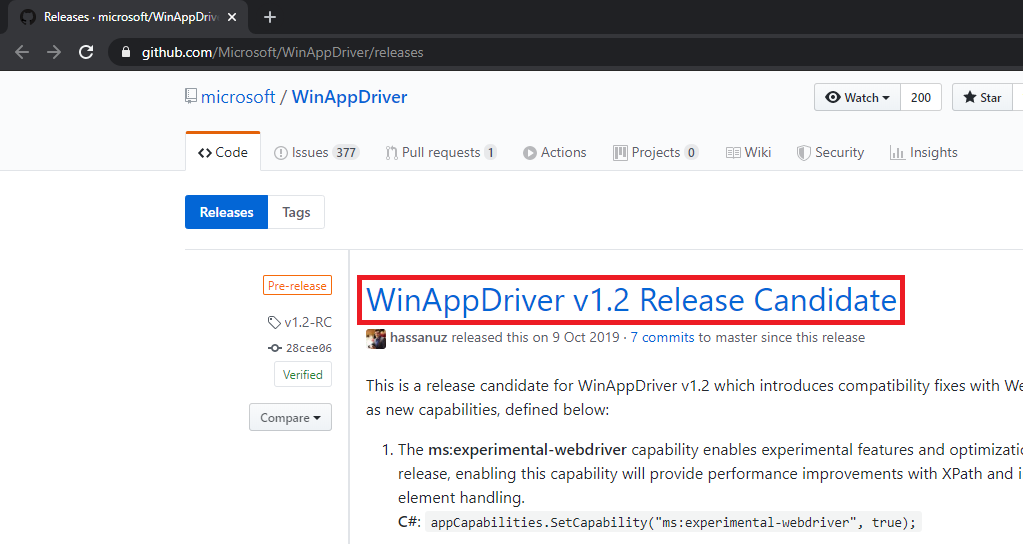
.

Ya finalizada la instalación, tendríamos instalado Windows Development Kit.

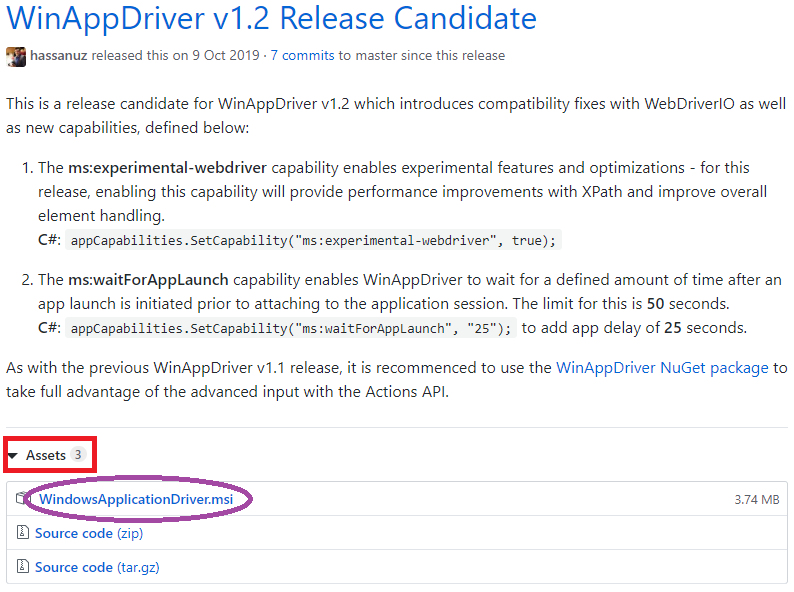


Como descargar e instalar WinAppDriver

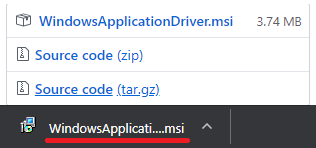
Como WinAppDriver es un servicio, inicialmente debe de ser descargado el servidor, por tanto se procede a buscar el repositorio de los instaladores de [WinAppDriver](https://github.com/Microsoft/WinAppDriver/releases), donde se seleccionará la versión más reciente como se puede apreciar en la siguiente imagen.



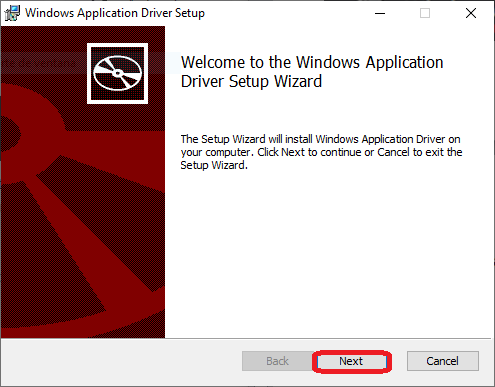
Después de encontrar la versión más reciente de WinAppDriver, lo que sigue es descender un poco en la página hasta encontrar un ítem que dice “Assets”, el cual se debe hacer clic para desplegar una lista con unos archivos asociados, hecho de esto, para descargar el instalador de WinAppDriver se hace clic el archivo que dice WindowsApplicationDriver.msi



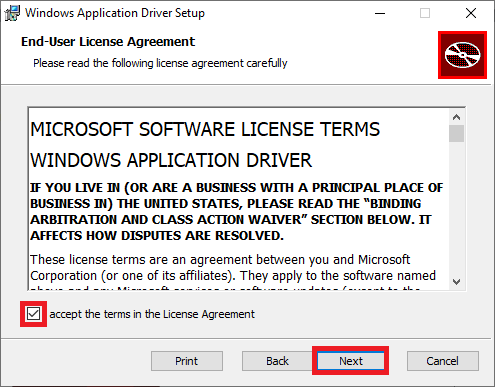
Hecho esto, se descargará el instalador.



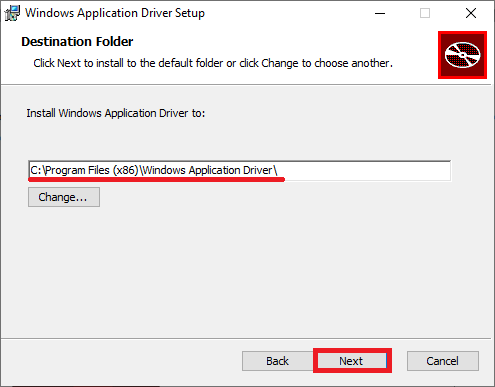
Cuando el instalador sea ejecutado, se le oprime a next para iniciar la instalación.



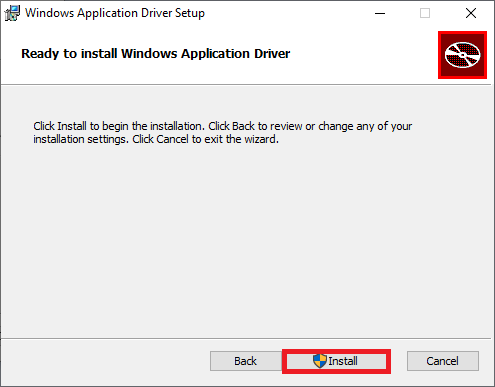
Luego se aceptan términos y condiciones para poder clicar el siguiente next.



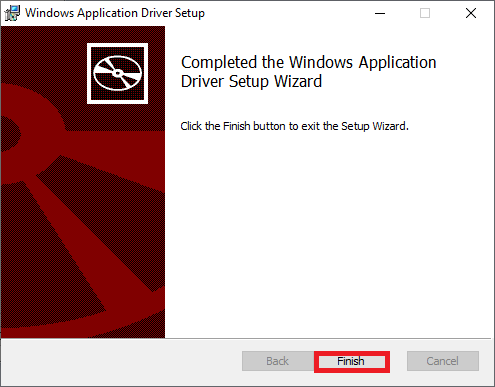
Para continuar se escoge la ruta de instalación del programa (que para este ejemplo se utilizará la que trae predeterminada)



Finalmente se selecciona instalar.



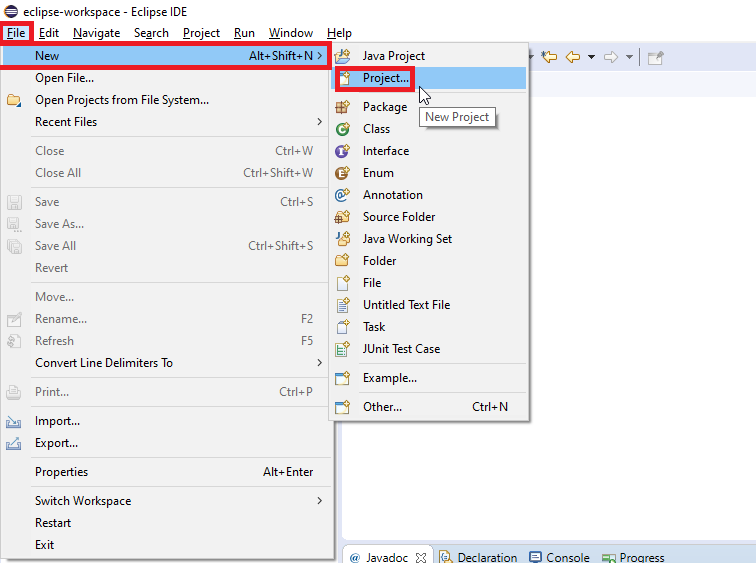
Y ya en esta pantalla se cierra el menú de instalación con el botón finish.

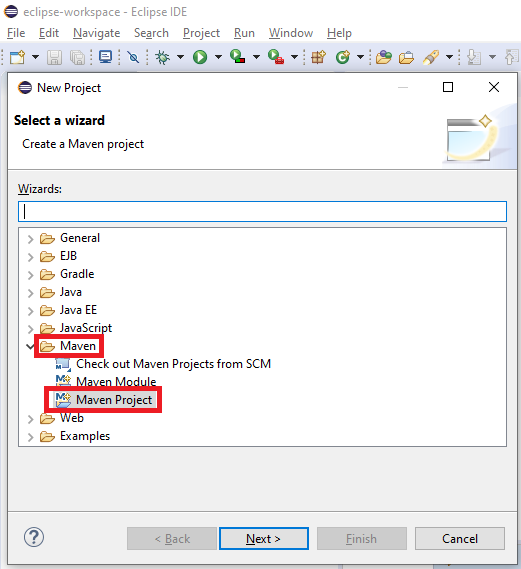


Hecho esto, tenemos el ejecutable del programa en la ruta establecida.

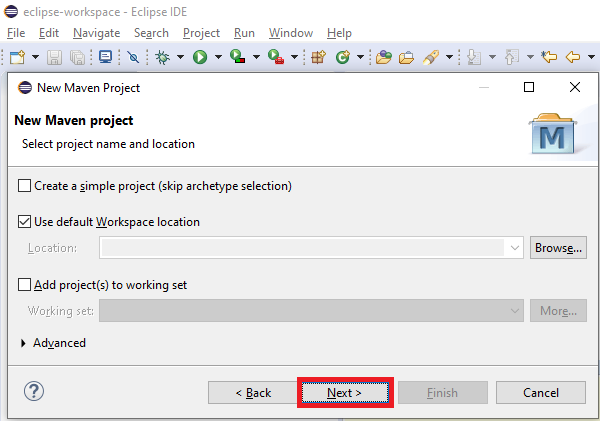
Configuración de un proyecto

Para poder hacer un proyecto y hacer pruebas con WinAppDriver se deben tener ciertas librerías y dependencias en este, para eso seguiremos los siguientes pasos.

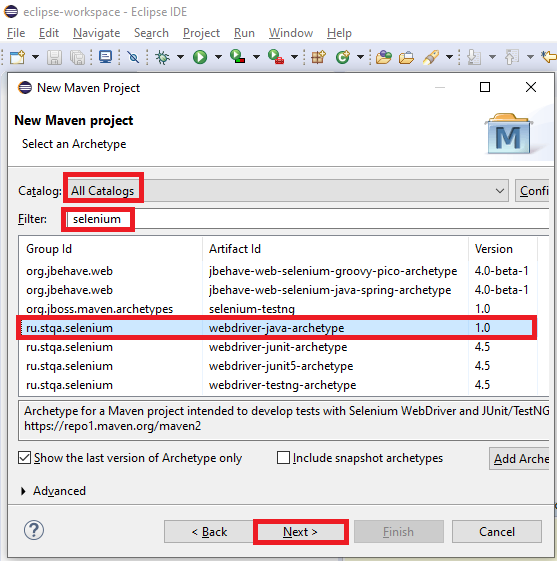
1. Estar en la ventana principal de eclipse y crear un proyecto nuevo desde File>New>Project
2. Hecho lo anterior, se abrirá una ventana para seleccionar el tipo de proyecto a trabajar, estando en esta ventana, seleccionamos la carpeta que dice Maven, hacemos click en el tipo de proyecto Maven Project, y una vez seleccionado esto hacemos click en next.



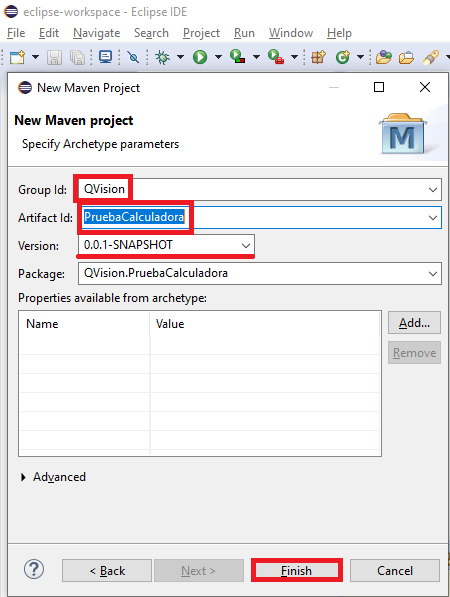
1. Después de hacer click en next aparece la ventana para configurar el directorio de trabajo donde se desea poner el proyecto, en este caso se dejará en el directorio predeterminado y se oprimirá next.



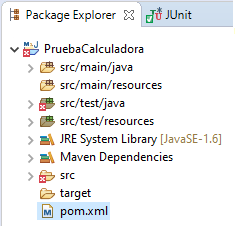
1. En esta parte, se escoge el arquetipo del proyecto, hay que asegurarse de que en la lista de la etiqueta catalog esté seleccionada la opción de All Catalogs, y en filter escribimos selenium para filtrar el arquetipo, una vez escrito eso esperamos a que se filtren todos los arquetipos, buscamos el que dice webdriver-java-archetype y seleccionamos la versión más reciente. Hecho esto oprimimos next.



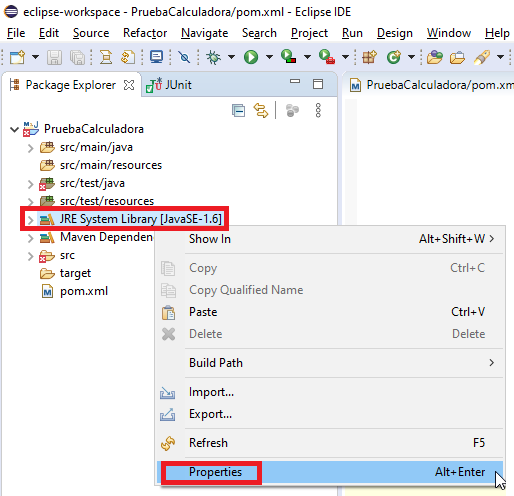
1. Hecho esto, ponemos el Grupo al que deseamos que pertenezca el proyecto en Group Id, y el nombre del proyecto en Artifact Id y la versión la dejamos en el valor 0.0.1-SNAPSHOT que es la versión predeterminada que trae este proyecto. Para terminar y crear el proyecto oprimimos finish.



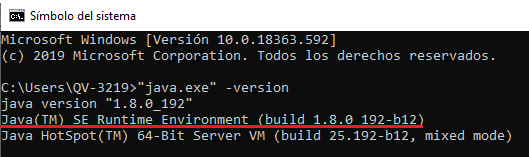
1. Si todos los pasos fueron hechos correctamente, el proyecto en el visualizador de paquetes debería verse así.

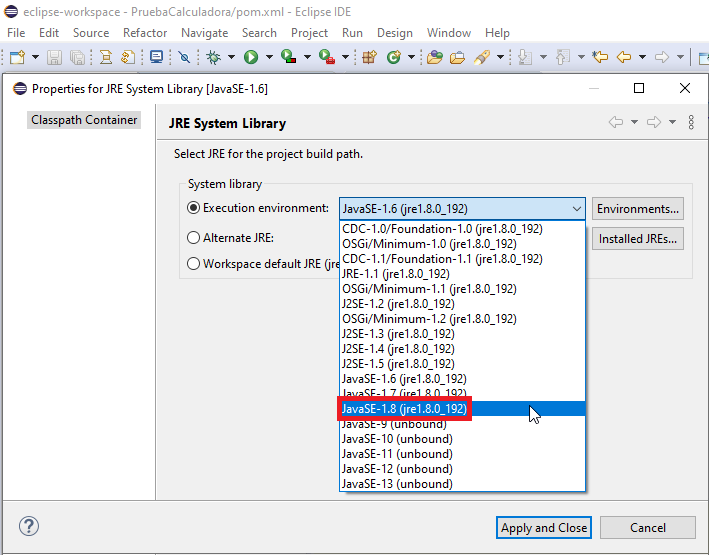


1. Ahora configuraremos las librerías del JRE que tengamos, para esto haremos click derecho en JRE System Library y seleccionamos la opción Properties.

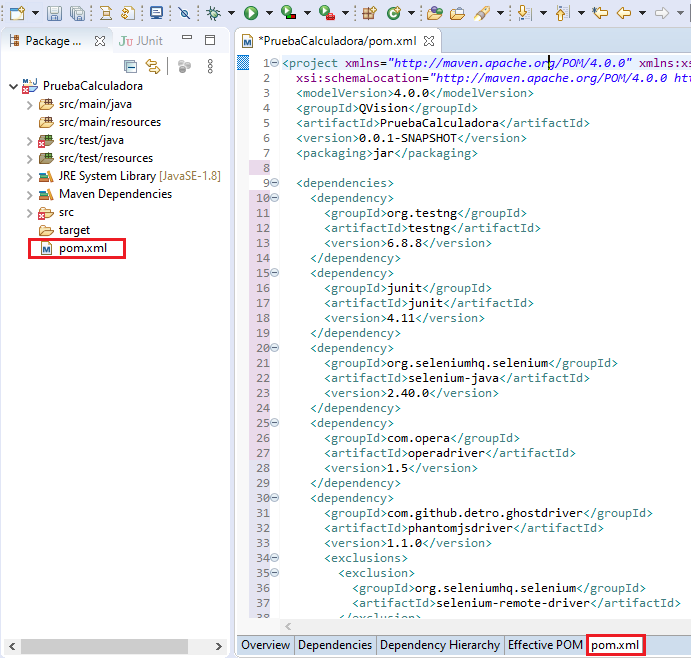


1. Cuando se abra esta ventana, escogemos la versión del JavaSE que tengamos, recordemos que para que podamos hacer las pruebas se debe tener una versión igual o superior a la JavaSE 1.8 (para saber la versión abrimos la consola de comandos y escribimos: “java.exe” –version), después de ver la versión, en el panel de propiedades buscamos y seleccionamos dicha version y finalmente oprimimos Apply and Close.





1. Lo siguiente es poner las dependencias que se van a utilizar y quitar las que no, para esto, abrimos en los archivos del proyecto un archivo que se llama pom.xml donde después en la parte inferior de la ventana de ese archivo hacemos clic en donde dice pom.xml, hecho esto, se abrirá una ventana que contiene el XML del proyecto.



1. Ahora vamos a reemplazar el código que tiene este pom.xml como se puede ver en el siguiente gif con el código fuente que se encuentra debajo de este.

(Gif\*)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>CalculatorTest</groupId>

<artifactId>CalculatorTest</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.3.1</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

</dependency>

<dependency>

<groupId>io.appium</groupId>

<artifactId>java-client</artifactId>

<version>5.0.0-BETA6</version>

</dependency>

</dependencies>

</project>

1. adsasdasd

Localizadores para mapear objetos

|  |  |  |  |
| --- | --- | --- | --- |
| API | Localizador | Nombre del atributo en inspect.exe | Ejemplo |
| FindElementByAccessibilityId | Identificador único para elementos hermanos en un mismo contenedor | AutomationId | AppNameTitle |
| FindElementByClassName | Nombre de la clase de un elemento | ClassName | TextBlock |
| FindElementById | Identificador único del elemento | RuntimeId (decimal) | 42.333896.3.1 |
| FindElementByName | Nombre | Name | Calculator |
| FindElementByTagName | Nombre de etiqueta | LocalizedControlType (upper camel case) | Text |
| FindElementByXPath | xpath | Any | //Button[0] |

Capacidades

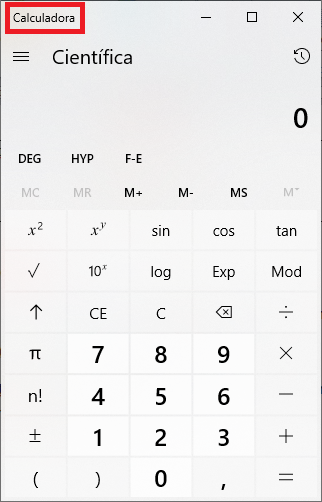
|  |  |  |
| --- | --- | --- |
| **Capacidad** | **Descripción** | **Ejemplo** |
| app | Identificador de la aplicación o directorio del ejecutable | Microsoft.MicrosoftEdge\_8wekyb3d8bbwe!MicrosoftEdge |
| appArguments | Argumentos para iniciar la aplicación | https://github.com/Microsoft/WinAppDriver |
| appTopLevelWindow | Argumento para crear una nueva sesión para una aplicación que tiene otra sesión existente | 0xB822E2 |
| appWorkingDir | Directorio donde la aplicación va a trabajar (Solo aplicaciones clasicas) | C:\Temp |
| platformName | Nombre de la platadorma | Windows |
| platformVersion | Version de la plataforma | 1.0 |

Métodos para el manejo de ventana y sesiones

quit(): Método para cerrar y terminar una sesión con todas sus ventadas asociadas.

close(): Método para cerrar una sesión.

getTitle(): Obtiene el título de la ventana de la sesión (el título que está en la parte superior de toda ventana)



manage(): interfaz de opciones para usar métodos para controlar la ventana de un programa.

* window(): interfaz para controlar métodos de la ventana actual.
  + maximize(): método para maximizar una ventana.
  + size(): sin argumentos, retorna las dimensiones de una ventana, con argumentos (ancho, alto) cambia el tamaño de una ventana
  + getPosition(): obtiene la posición inmediata de la ventana de un programa
  + setPosition(): recibe como argumento un objeto tipo Point, el cual define una posición en pantalla para ubicar la interfaz gráfica del programa.
* Timeouts(): Interfaz que controla los métodos de espera
  + implicitlyWait(): recibe como parámetro un entero y una unidad de tiempo dada por TimeUnit, este método sirve para que el driver espere una cierta cantidad de tiempo antes de determinar que no se encuentra un elemento.
  + pageLoadTimeout();recibe como parámetro un entero y una unidad de tiempo dada por TimeUnit, este método sirve para que el driver espere un tiempo determinado a que cargue la página de un programa antes de determinar de que no se encuentra.

getWindowHandle(): Obtiene un identificador único de la ventana del programa que está abierto

getWindowHandles(): Obtiene los identificadores de ventanas de un mismo programa que estén abiertas.

getOrientation():Obtiene la orientación de la pantalla.

getSessionId(): Retorna el Id de una sesión instanciada

Métodos de entrada

getKeyboard(): Es una interfaz de métodos para el teclado

* sendKeys(): Recibe como argumento una secuencia de caracteres (letras, números o caracteres especiales) y los ejecuta sobre la sesión.
* pressKey(): Recibe un objeto tipo Keys (ej: Keys.ENTER) y ejecuta esta secuencia sobre la sesión.

Métodos de interacción con elementos

Estos métodos aplican siempre y cuando se utilicen localizadores de elementos.

Ej: *CalculatorSession*.**findElementByName**("Uno").click();

* click(): método para hacer click en un elemento.
* sendKeys(): Método que recibe como argumento una secuencia de caracteres (letras, números o caracteres especiales) y los ejecuta sobre un elemento.
* clear(): limpia un campo de entrada si no está vacío
* idDisplayed(): Retorna true si el elemento seleccionado se encuentra en pantalla (hay que tener cuidado con el uso de este método, porque en Windows si un elemento no se está mostrando en pantalla, normalmente no se puede mapear y generará un error)
* isEnabled(): Retorna true si un elemento seleccionado se encuentra habilitado, en caso contrario retorna false.
* isSelected(): Método que retorna verdadero si un elemento esta seleccionado, en caso contrario retorna false, normalmente se usa en Checkboxes, Drop Down List y Radio Buttons.
* element(): se puede usar cualquier localizador para mapear elementos si un elemento tiene hijos, pueden mapearse uno o más.
* getText(): Retorna el nombre de un elemento.
* getScreenshotAs(): Toma una captura de pantalla a un elemento seleccionado (también se puede usar este método con las sesiones) y recibe como argumento un objeto tipo OutputType, donde normalmente se usa el OutputType.File, usando este tipo de dato se guarda como un archivo JPG y se guarda temporalmente en la carpeta C:\Users\NombreEquipo\AppData\Local\Temp
* equals(): Retorna true si el elemento seleccionado es igual a un elemento objetivo, en caso contrario retorna false.
* getLocation(): Retorna las coordenadas de localización del elemento.
* getSize(): Retorna las medidas de un elemento.
* getTagName(): Retona el tipo de nombre de controlador de un objeto, por ejemplo: *CalculatorSession*.findElementByAccessibilityId("num7Button").getTagName() retornará: ControlType.Button