# Writing Custom Annotations

In this topic, we will see how to create a custom annotation and hookup with TestNG configuration methods.

WE'LL COVER THE FOLLOWING


- Need for creating custom annotations
- Example of custom annotation


## Need for creating custom annotations

We can create custom annotations and use them using TestNG configurations methods to make annotations more customizable per your needs.

## Example of custom annotation

As a demonstration, we will create a single `@DataProvider` method that reads from different files with a file to read given in a test method.

An annotation that is applicable to methods that pass on the file path to read data from:

```
@Retention(java.lang.annotation.RetentionPolicy.RUNTIME)
@Target(java.lang.annotation.ElementType.METHOD)
public @interface DataProviderFile {

    String file() default "";
}
```

`DataProviderFactory` class containing the `@DataProvider` methods:

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
```

```
import java.util.List;

public class DataProviderFactory {

    @DataProvider(name = "data")
    public static Iterator<Object[]> getData(Method method) throws IOExcep
tion {
        DataProviderFile file = method.getAnnotation(DataProviderFile.clas
s);
        if (file != null) {
            List<Object[]> list = new ArrayList<>();
            for (String line : Files.readAllLines(Paths.get(file.file())))
 {
                list.add(new Object[] { line });
            }
            return list.iterator();
        }
        return Collections.emptyIterator();
    }
}
```

In the above code snippet, we are getting the file path that is passed to the `@DataProvider` method using `@DataProviderFile` annotation and transforming the lines in the file to `Object[]` and returning as `Iterator<Object[]>` as expected from `@DataProvider` method.

Test Class passing different files to read from:

```
public class DataProviderExample {

    @DataProviderFile(file = "sampleA.txt")
    @Test(dataProvider = "data", dataProviderClass = DataProviderFactory.c
lass)
    public void testA(String line) {
        assertNotNull(line);
    }

    @DataProviderFile(file = "sampleB.txt")
    @Test(dataProvider = "data", dataProviderClass = DataProviderFactory.c
lass)
    public void testB(String line) {
        assertNotNull(line);
    }

}
```

In the above test class, both the test methods are using the same `@DataProvider` method but reading from different files that are passed using `@DataProviderFile`. Thus, we are reusing the same `@DataProvider` method instead of writing two different ones.

---

That is it for the details of the TestNG framework. You now know how the different functionalities provide support for effective testing of a UI. There is a quiz in the next lesson for you to solve.