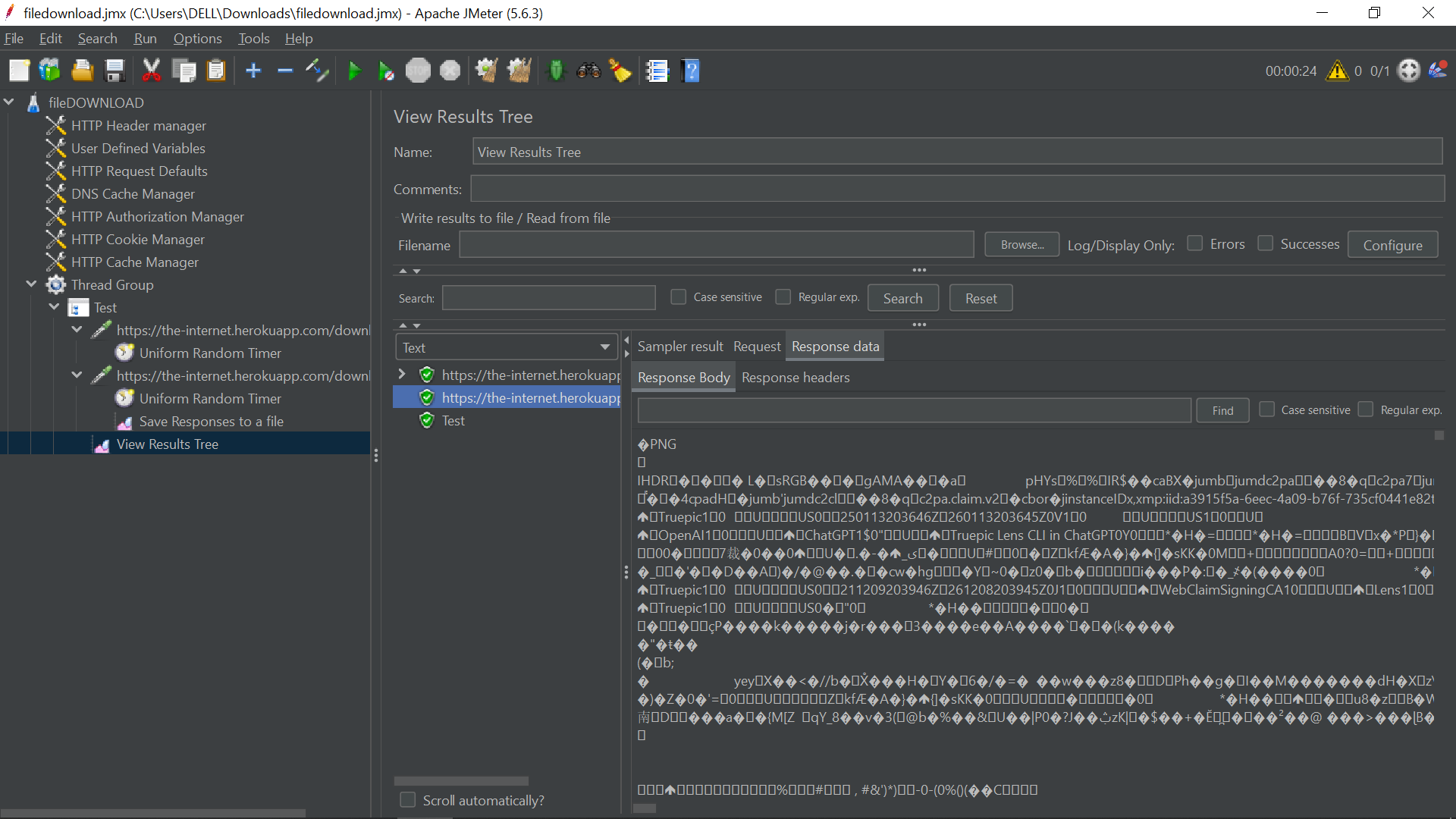
🔽 File Download





Apache JMeter is a powerful open-source tool for performance testing. It operates at the protocol level, making it ideal for simulating and measuring file downloads without rendering web pages or executing JavaScript like a browser.

**🎯 Why Test File Downloads?**

Testing file downloads helps:

* Measure server response time under load.
* Check server throughput for static or dynamic file delivery.
* Verify file integrity (if necessary).

**✅ Step-by-Step Guide to File Download Testing**

**1. Identify the File Download URL**

Use browser developer tools (Network tab) or a proxy tool like **Fiddler**, **Charles Proxy**, or **Burp Suite** to capture the actual download request URL.  
Look for a direct GET or possibly POST request returning a file (e.g., .zip, .pdf, .exe).

**2. Create a Basic JMeter Test Plan**

**Add a Thread Group:**

* Right-click **Test Plan** → Add → **Threads (Users)** → Thread Group.
* Set **Number of Threads**, **Ramp-Up Period**, and **Loop Count**.

**Add an HTTP Request Sampler:**

* Right-click **Thread Group** → Add → **Sampler** → HTTP Request.
* Configure as follows:
  + **Protocol**: http or https
  + **Server Name or IP**: example.com
  + **Path**: /downloads/file.zip
  + **Method**: GET (or POST if required)

**3. (Optional) Save the Downloaded File**

To verify file content or integrity:

**Add a Listener:**

* Right-click **HTTP Request** → Add → **Listener** → Save Responses to a file.

**Configure:**

* **Filename prefix**: e.g., downloaded\_file\_
* Use ${\_\_threadNum} or ${\_\_time} to avoid file overwrite in multi-threaded tests.
* Decide whether to:
  + Add numeric suffixes
  + Override file extension
  + Save only successful responses

**4. Handle Dynamic Download URLs (if needed)**

Some download links are dynamically generated (e.g., pre-signed URLs, JWT tokens).

**Workflow:**

1. **Send request** to the page/API generating the download link.
2. **Extract link** using a Post-Processor:
   * **Regular Expression Extractor**
   * **JSON Extractor**
   * **CSS/JQuery Extractor**
3. **Store value** in a JMeter variable (e.g., ${download\_url}).
4. **Use it** in your file download request:
   * Method: GET
   * Path: ${download\_url} (you may need to split host and path)

**5. Execute and Analyze Your Test**

**Add Listeners:**

* **View Results Tree** (for debugging, not recommended for load tests)
* **Summary Report** / **Aggregate Report** (for metrics)

**Run the Test:**

* Use **GUI mode** for test design and debugging.
* Use **Non-GUI mode** (jmeter -n -t test.jmx -l results.jtl) for performance testing to reduce resource usage.

**📌 Helpful Tips**

* **JMeter ≠ Browser**: It doesn’t execute JavaScript or follow UI logic. Mimic network requests manually.
* **File Overwrite**: Use ${\_\_threadNum} or ${\_\_time} in the prefix to avoid conflicts during multi-user testing.
* **HTTP Cache Manager**: Add it to simulate browser cache or disable caching.
* **Content-Type Handling**: Use it to derive file extensions or override manually.

**🧪 Example Use Case**

Suppose you’re testing https://example.com/downloads/report.pdf. Your HTTP Request Sampler would be:

* **Protocol**: https
* **Server Name**: example.com
* **Path**: /downloads/report.pdf
* **Method**: GET

Add "Save Responses to a file" listener with prefix report\_${\_\_threadNum}\_.

**🧩 Bonus: Verify File Content (Optional)**

You can write a **BeanShell / JSR223 Post-Processor** to:

* Read downloaded file content.
* Compare it with a checksum or expected data.
* Log verification status.

Let me know if you’d like:

* A downloadable .jmx template.
* A reusable Post-Processor snippet for validating file content.
* Help with dynamic download scenarios (e.g., pre-signed URLs).