Screen Scraping

Screen Scraping Defintions (http://www.wikipedia.org/)

Originally, it referred to the practice of reading text data from a computer display terminal's screen. This was generally done by reading the terminal's memory through its auxiliary port, or by connecting the terminal output port of one computer system to an input port on another.

As a concrete example of a classic screen scraper, consider a hypothetical legacy system dating from the 1960s — the dawn of computerized data processing. Computer to user interfaces from that era were often simply text-based dumb terminals which were not much more than virtual teleprinters (such systems are still in use today for various reasons).

The desire to interface such a system to more modern systems is common. A robust solution will often require things no longer available, such as source code, system documentation, APIs, and/or programmers with experience in a 50-year-old computer system.

In such cases, the only feasible solution may be to write a screen scraper which "pretends" to be a user at a terminal. The screen scraper might connect to the legacy system via Telnet, emulate the keystrokes needed to navigate the old user interface, process the resulting display output, extract the desired data, and pass it on to the modern system.

More modern screen scraping techniques include capturing the bitmap data from the screen and running it through an OCR engine, or in the case of GUI applications, querying the graphical controls by programmatically obtaining references to their underlying programming objects.

Web Scraping (http://www.wikipedia.org/)

From Screen Scraping we get the idea of the new technology Web Scraping (also called Web Harvesting or Web Data Extraction) which is a computer software technique of extracting information from websites.

Usually, such software programs simulate human exploration of the Web by either implementing HTTP, or embedding certain full-fledged Web browsers, such as the Internet Explorer or the Mozilla Web browser.

Web scraping is closely related to Web indexing, which indexes Web content using a bot and is a universal technique adopted by most search engines. In contrast, Web scraping focuses more on the transformation of unstructured Web content, typically in HTML format, into structured data that can be stored and analyzed in a central local database or spreadsheet.

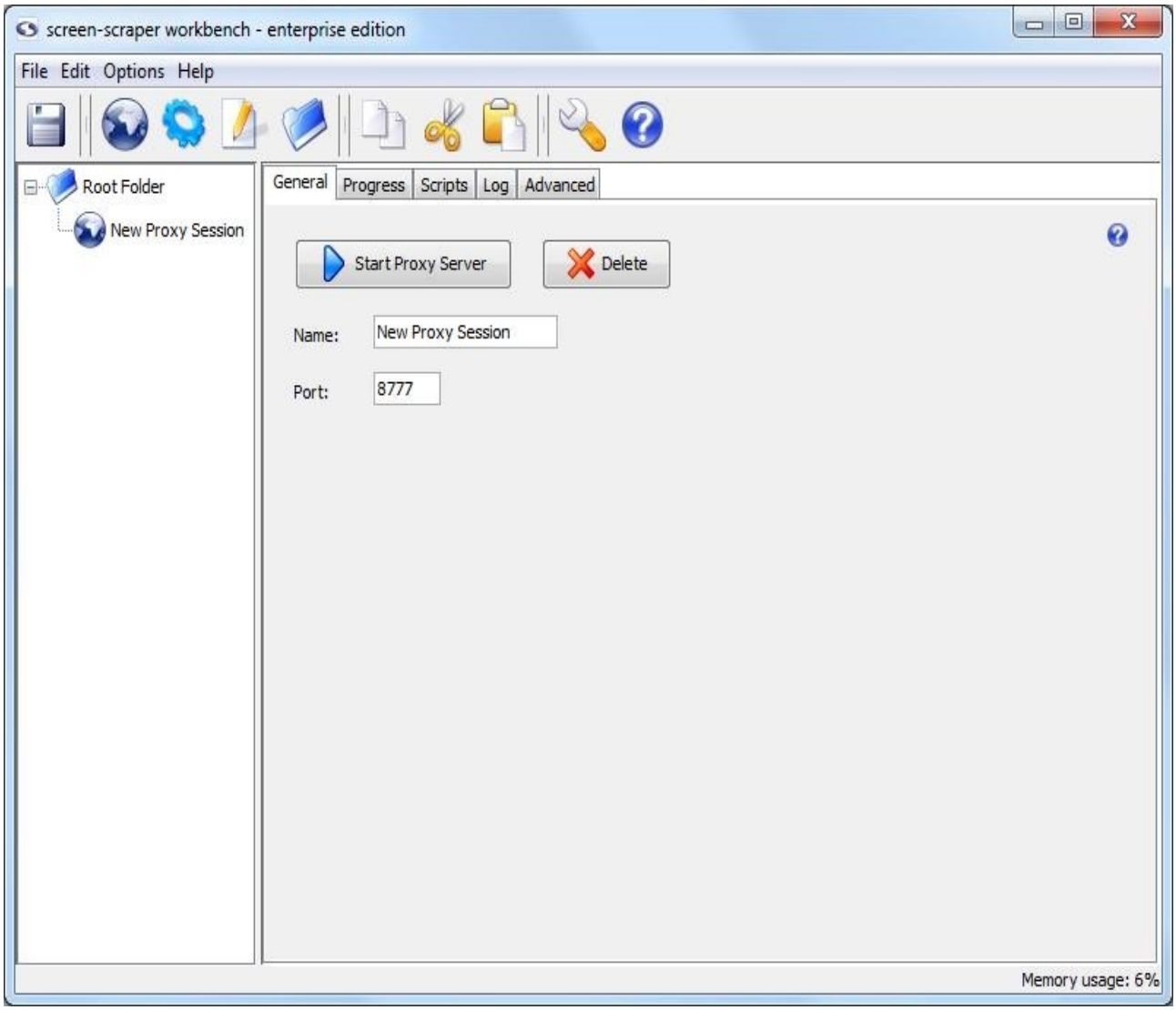
Web scraping is also related to Web automation, which simulates human Web browsing using computer software. Uses of Web scraping include online price comparison, weather data monitoring, website change detection, Web research, Web content mashup and Web data integration.

Screen-Scraper Software (http://www.screen-scraper.com/)

Here we will use one of the Web Scraping Software called screen-scraper which extracte information from web sites with four main steps:-

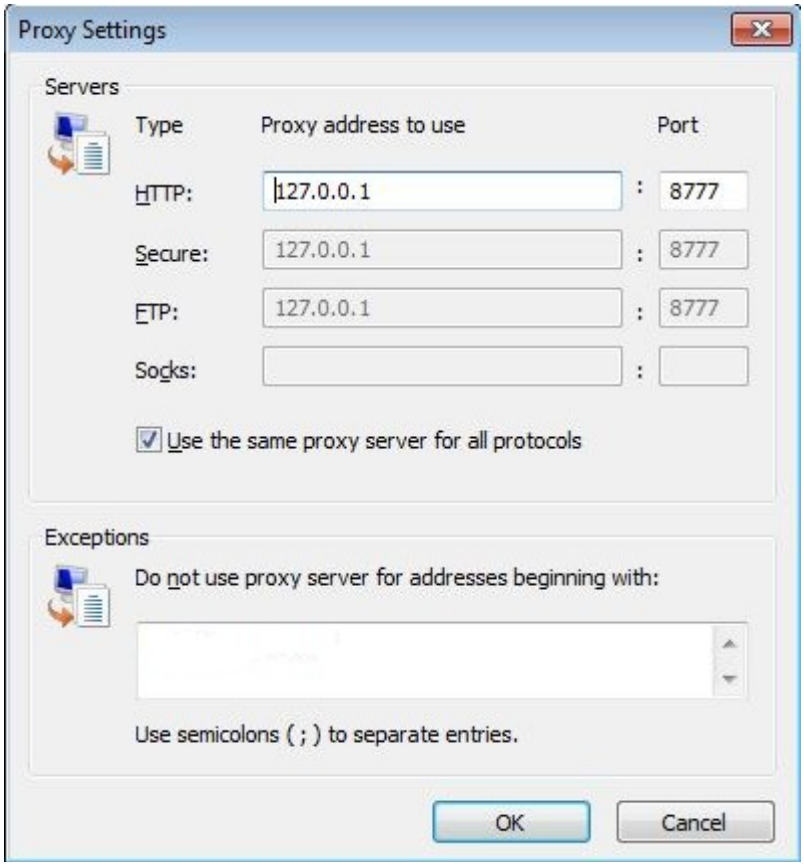
1. Use the **proxy server** to determine the exact files that need to be requested in order to get the information you're after.
2. Create a **scraping session** with **scrapeable files** that define the sequence of pages screen-scraper will request.
3. Generate **extractor patterns** to define the exact information you need screen-scraper to grab from each page.
4. Write small **scripts** or programming code to invoke screen-scraper and/or work with the data it extracts. If you don't do much programming, don't worry. Generally the scripts you'll need to write to work with screen-scraper are small and simple, and you can often just modify the example scripts we provide.

Create a proxy session now by clicking the Add a new proxy session icon. Once you have added the proxy session, you screen should look similar to the following screenshot.



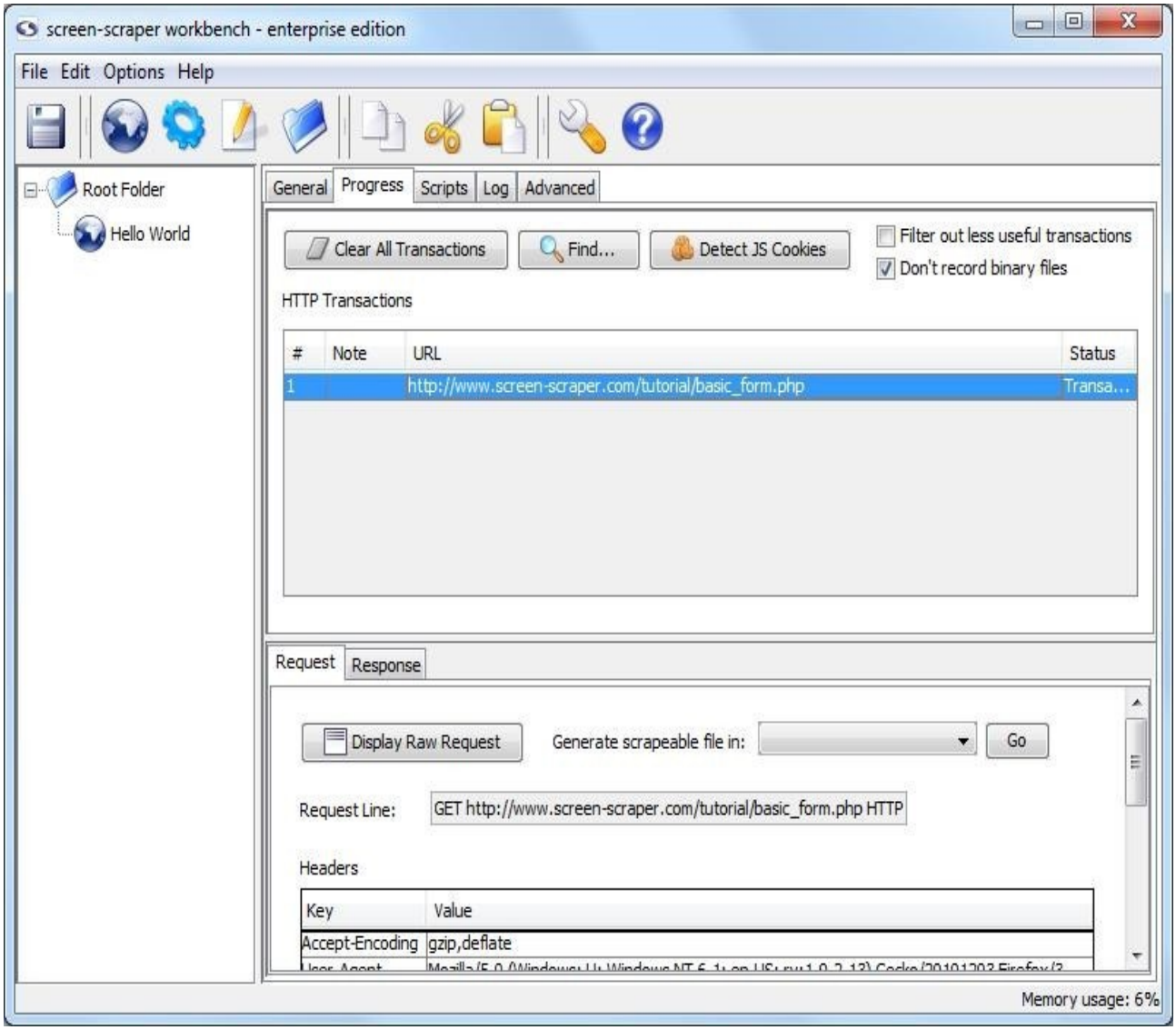
Give the proxy session the name `Hello World` by typing it into the Name field. The Port field determines the port number that your web browser will use when communicating with screen-scraper's proxy server. Leave the port at `8777`.

we need to set up our web browser so that it will use screen-scraper as a proxy server



At this point we can get the proxy server running. We can do this by clicking on the Start Proxy Server button in the proxy session. After this, we click on the Progress tab, which will display all of the requests and responses recorded by the proxy server.

Now we visit
If you take a look at screen-scraper, you'll notice that it recorded this transaction in the HTTP Transactions table.
We can click on any row in the table to load the information related to the browser's request and response in the lower pane.

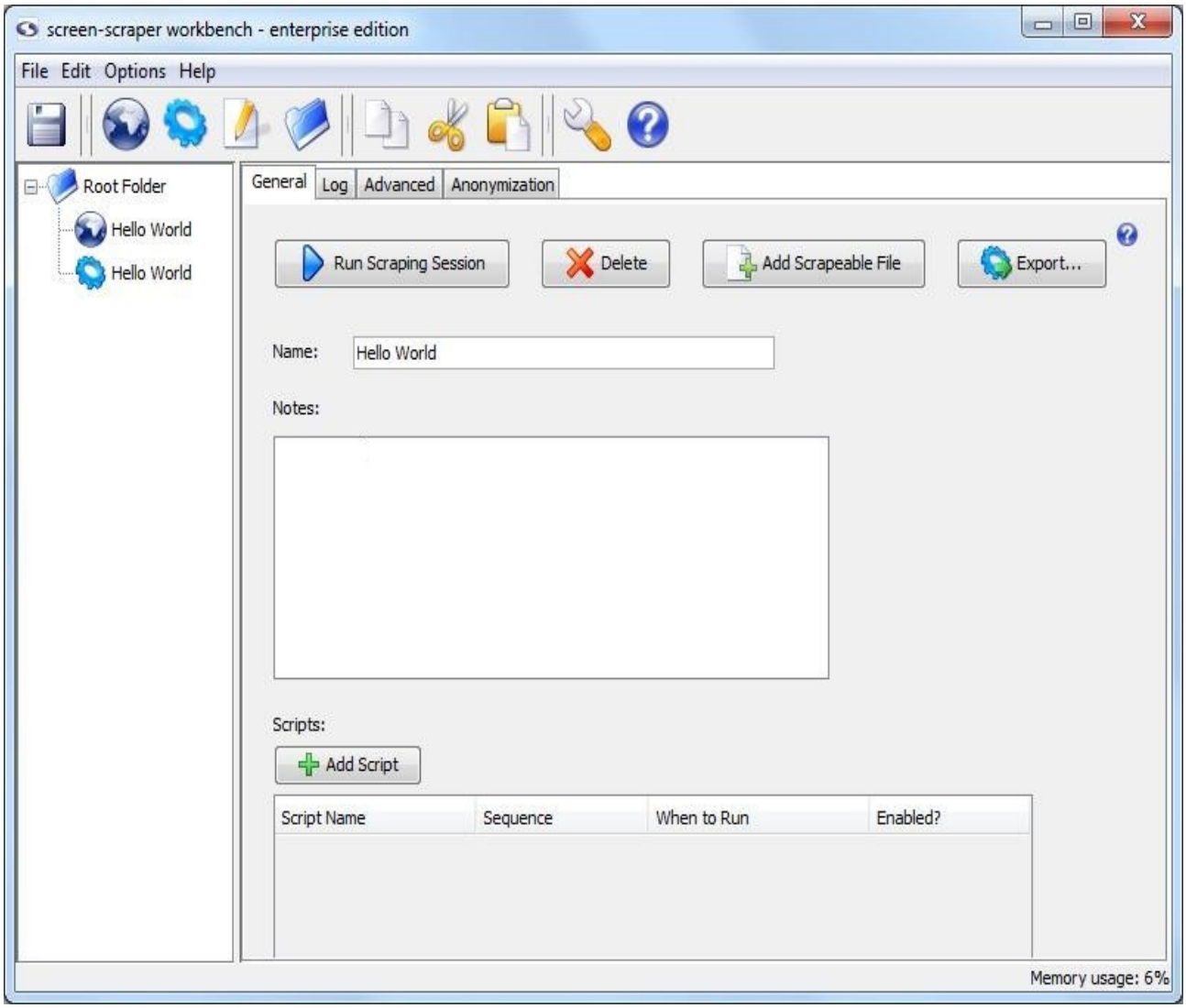


The lower pane shows the details of the HTTP request the browser made, any HTTP headers (including cookies), as well as POST data (if any was sent). We can view the corresponding response from the server by clicking on the Response tab.

We start creating a scraping session. A scraping session is simply a container for all of the files and other objects that will allow us to extract data from a given web site.

To create a scraping session, click on the Add a new scraping session button.

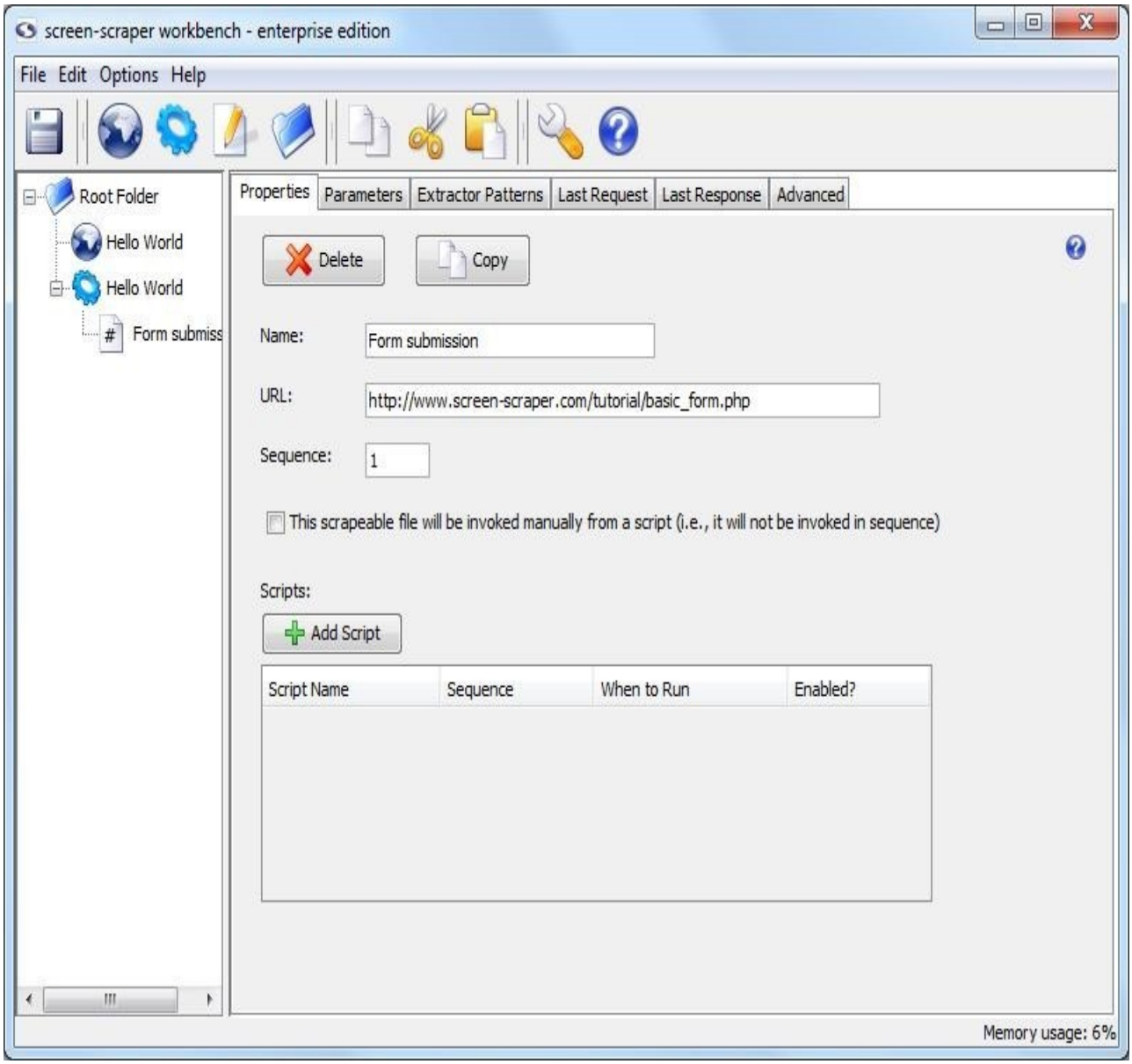
When scraping session appears rename it to `Hello World`.



Now we return to our Hello World proxy session and click on the Progress tab to view our HTTP transactions.

Any of the transactions in the table can be made into scrapeable files in our scrape. We choose one to load its information in the Request tab.

To create a scrapeable file from this transaction we just select the scraping session that we want the file to be created in. In the Generate scrapeable file in drop-down, we select Hello World then we click the Go button. Now we will see a scrapeable file in the objects tree.



Just to make sure things are correctly setup let's run a quick test. For this test we are going to run the scraping session.

To start the scrape, we click on the Hello World scraping session in the objects tree, then click the Run Scraping Session button. This will start the scraping session and transition you to the Log tab. It should just take a moment to run.

We can view the text of the file that was scraped by clicking on the Form submission scrapeable file in the objects tree, then on the Last Response tab. This will show the whole of the HTTP response that the server sent back to screen-scraper.
We can view what the page looks like when it is rendered by clicking the Display Response in Browser button. It's often helpful to view the last response for a scrapeable file after running a scraping session so that we can ensure that screen-scraper requested the right page.

What is an Extractor Pattern?

An extractor pattern is a block of text (usually HTML) that contains special tokens that will match the pieces of data we 're interested in gathering. These tokens are text labels surrounded by the delimiters ~@ and @~.

We can think of an extractor pattern like a stencil. A stencil is an image in cut-out form, often made of thin cardboard. To use a stencil we place it over a piece of paper, apply paint, then remove the stencil. The effect is the paint only remains where there were holes in the stencil. Analogously, we can think of placing an extractor pattern over the HTML of a web page. The tokens correspond to the holes where the paint would pass through. After an extractor pattern is applied it reveals only the portions of the web page where tokens were added.

Creating and Extractor Pattern (conceptually)

Let's consider this snippet of HTML that was taken from the page:

You typed: Hello world!

As we're interested in extracting the string "Hello world!" our extractor pattern would look like this:

You typed: ~@FORM_SUBMITTED_TEXT@~

We have added an extractor token with the name FORM_SUBMITTED_TEXT. In this form the pattern is very exact and so prone to breaking if the page were to experience a minor change like adding another attribute to the span tag, changing the style attribute to a class assignment, or changing the tag that is used. To avoid these simple problems we will simplify our pattern.

>You typed: ~@FORM_SUBMITTED_TEXT@~<

As you can guess this does not make the pattern unbreakable, just more resilient. If the label before the submitted text was changed it would no longer match and if the pattern added something after the submitted text (but within the same tag) then the token would match too much. That said, we have made it as stable as we can while making sure that it only matches what we want.

Creating an Extractor Pattern

We can have as many extractor patterns as you'd like in a given scrapeable file. screen-scraper will invoke each of them in sequence after requesting the scrapeable file.

Using the Last Response Tab

The last response tab can be used to create extractor patterns using the HTML of the page. To view the HTML of the page, we click on the Form submission scrapeable file, then on the Last Response tab. Now we select the portion of the HTML that we want to use to create the extractor pattern:

With the text selected, right click and select Generate extractor pattern from selected text from the menu that shows up. This will transition you to the Extractor Patterns tab of the scrapeable file and place the selected text in the Pattern text field.

Now highlight the text that we want to be a token

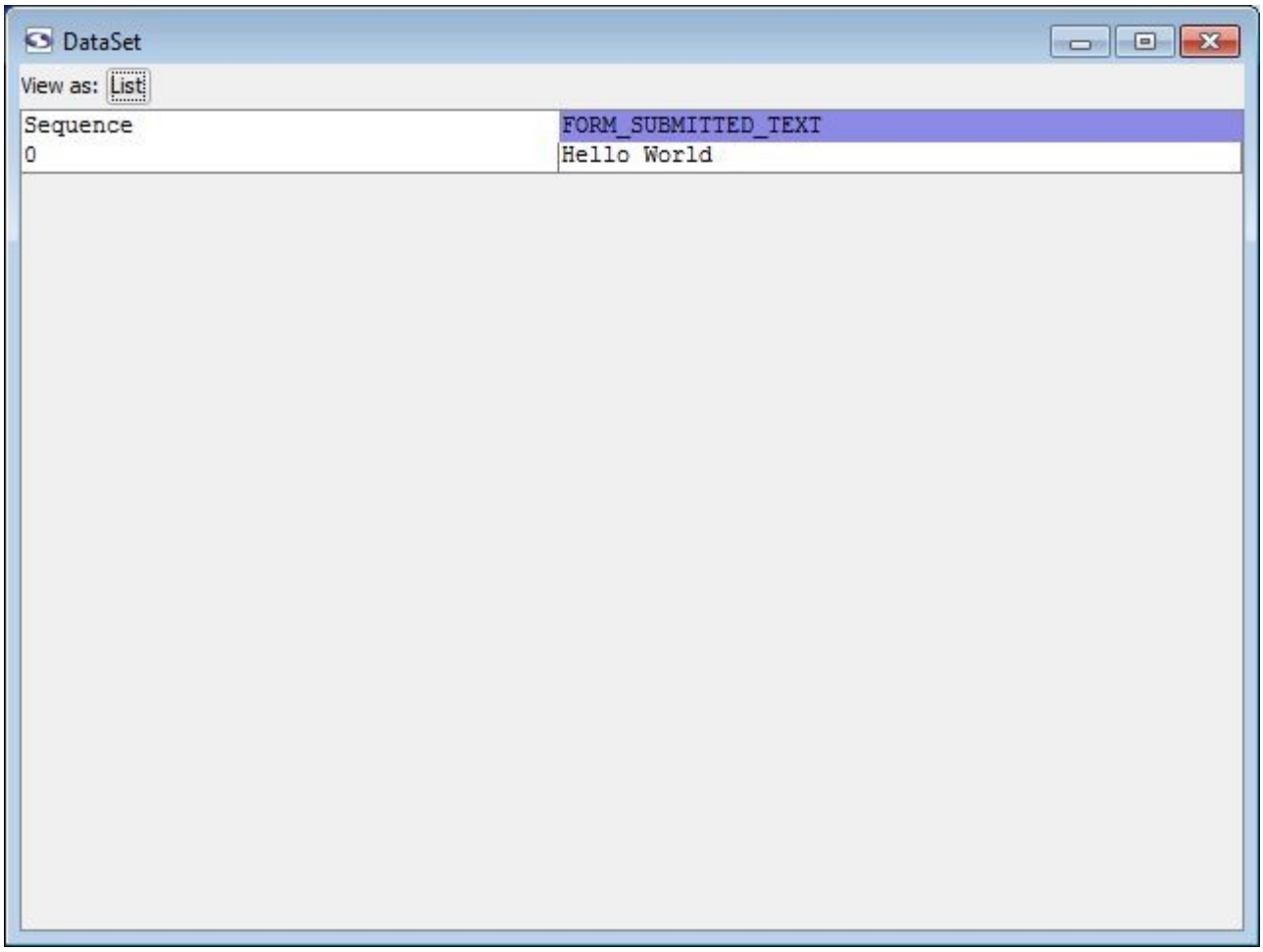
With the text selected right click and select Generate extractor pattern token from selected text. In the window that opens type `FORM_SUBMITTED_TEXT` in the Identifier textbox and close the window by clicking on the X in the upper right-hand corner. Don't worry settings are saved when the window closes.

We will notice that screen-scraper has automatically added the delimiters (~@ and @~) to the extractor token.

We give our extractor pattern the identifier Form data.

Testing the Extractor Pattern

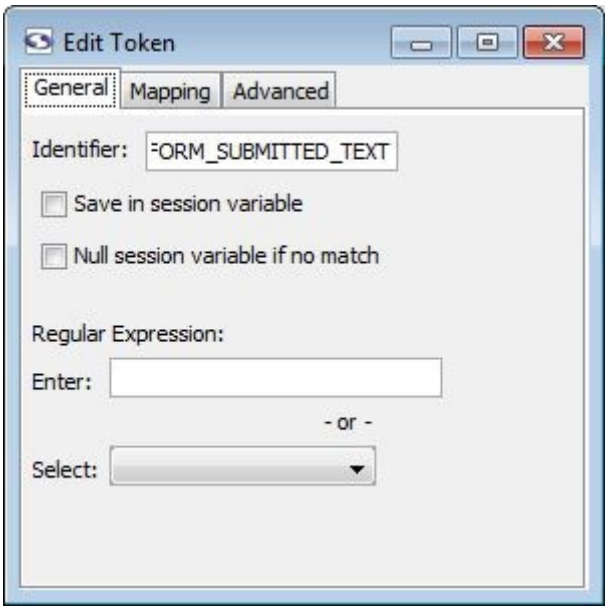
Go ahead and give the extractor pattern a try by clicking on the Test Pattern button. This will open a window displaying the text that the extractor pattern extracted from the page.



Looks like our extractor pattern has matched the snippet of text we were after. The Test Pattern is another invaluable tool we'll use often to make sure we're getting the right data. It simply uses the HTML from the Last Response tab, and applies the extractor pattern to it.

Adding Properties to Extractor Token

To modify the properties for our FORM_SUBMITTED_TEXT extractor token, we double-click on the text `FORM_SUBMITTED_TEXT` found between the ~@ @~ tokens in the Pattern text field.



In the window that opens, you can specify settings to the extracted data. . We are only worried about Save in session variable property.

screen-scraper uses session variables as a means to allow us to save and persist objects throughout the life of a scraping session. This means that screen-scraper will save the extracted data in memory so that it can be used later in scripts and such. In this case we'd like to save the text that our FORM_SUBMITTED_TEXT extractor pattern token retrieves. To indicate this, we click the Save in session variable checkbox, then we close the Edit Token window.

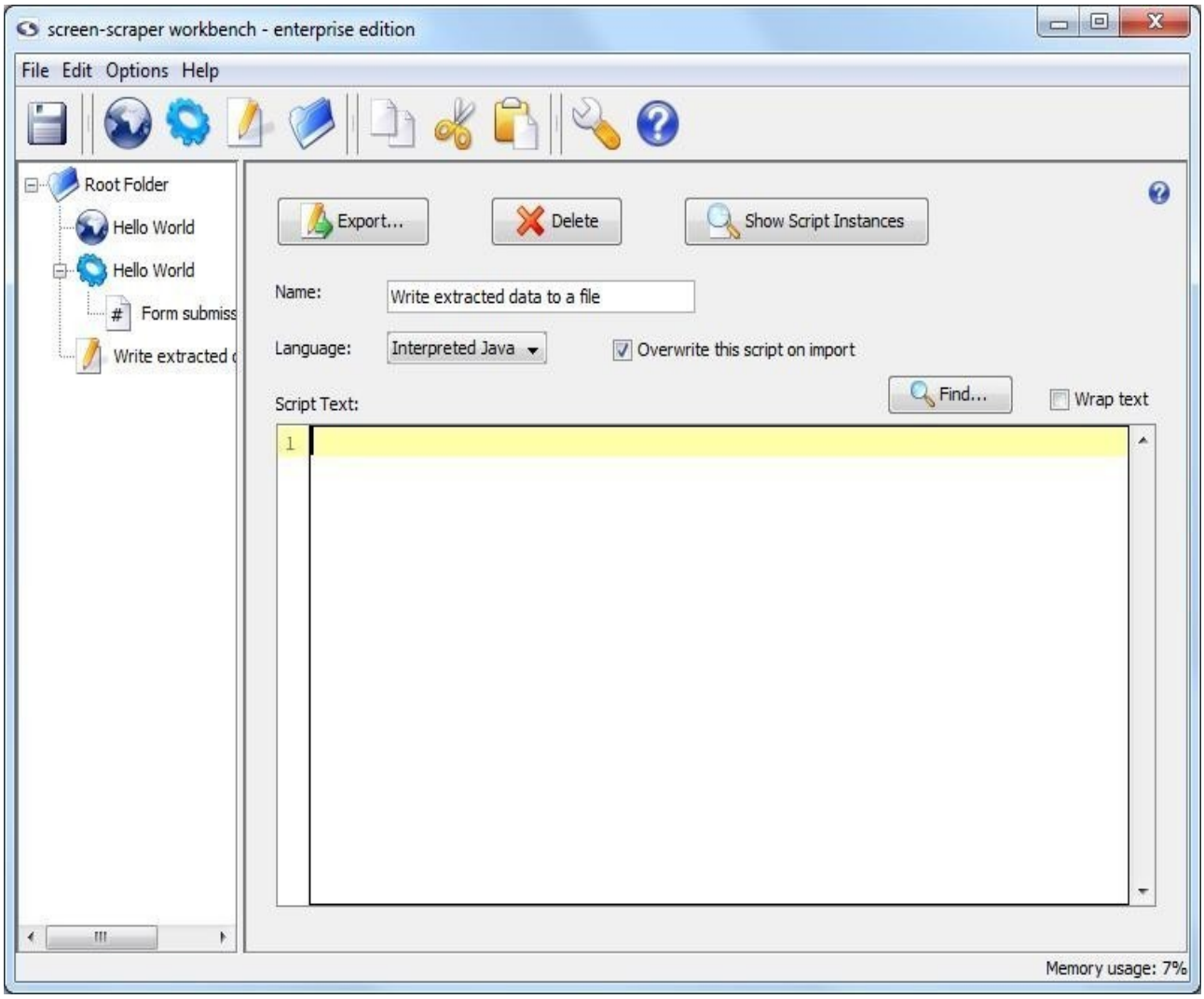
Now when screen-scraper runs this scraping session and extracts the text for this extractor pattern it will save the text in a session variable so that we can do something with it later.

How Scripts Work in screen-scraper

A screen-scraper script is a block of code that will get executed when a certain event occurs. For example, we might have a script that gets invoked at the beginning of a scraping session that initializes variables. Another script might get invoked each time a row in a list of search results is extracted from a site so that the information in that search result can be inserted into a database.

Creating a Script

To create a new script, we click on the (Add a new script) button. Give it the identifier `Write extracted data to a file`.



screen-scraper supports scripting in Interpreted Java, JavaScript, and Python when running on any operating system, and JScript, Perl, and VBScript when running on Windows. We will be using Java in this tutorial. Let's write our script.

```
// Output a message to the log so we know that we'll be writing the text out to a file.
session.log( "Writing data to a file." );

// Create a FileWriter object that we'll use to write out the text.
out = new FileWriter( "form_submitted_text.txt" );

// Write out the text.
out.write( session.getVariable( "FORM_SUBMITTED_TEXT" ) );

// Close the file.
out.close();
```

The `session.getVariable( "FORM_SUBMITTED_TEXT" )` method call retrieves the value of the FORM_SUBMITTED_TEXT session variable. This method call is able to get the value because we indicated earlier that the value for the FORM_SUBMITTED_TEXT extractor token was to be saved in a session variable (i.e., when we checked the Save in session variable checkbox).

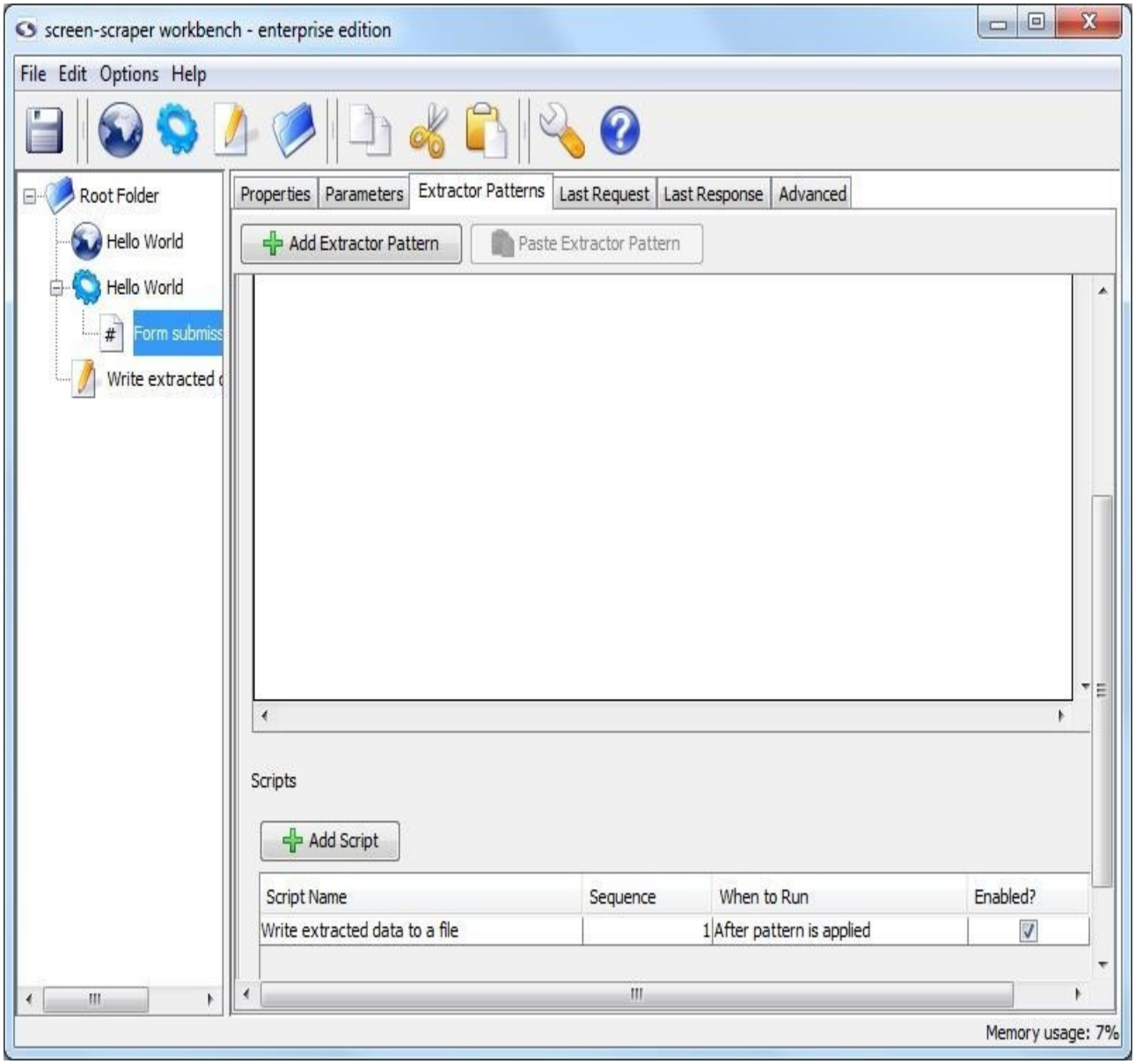
Adding Script Association/Trigger

A script is executed in screen-scraper by associating it with some event, such as before or after an extractor pattern is applied to the text of a web page. The script we've just written needs to be executed after screen-scraper has requested the web page and extracted the data we need from it.

We return to the extractor pattern we just created by clicking on the scrapeable file, then we click on the Extractor Patterns tab. In the lower section of our extractor pattern, we click on the Add Script button.

By default it adds an association to the first, alphabetically, script in our instance of screen scraper. If this is not Write extracted data to a file then click on the script name and select it.

The trigger (when it will run) is defined in the When to Run column. For our script select After pattern is applied.



Our Write extracted data to a file script will be invoked after screen-scraper has applied the Form data extractor pattern to the web page. That is, once the extractor pattern has applied as many times as it needs to (which, in this case, is only once), it will invoke the script.

Final Run

Now we have everything in place for our scrape. It knows what files to scrape, what data to extract, and what to do with the extracted data. Now it is time to make sure that it is working.

Just like with the test run, we click on the Hello World scraping session in the objects tree. First, we click on the Log tab. If there is existing text we get rid of it by clicking the Clear Log button. Now we click on the Run Scraping Session button.

After it finishes running, we can take a look at the contents of the form_submitted_text.txt file, which will be located in the screen-scraper installation directory (e.g., C:\Program Files\screen-scraper professional edition\).