# Screen Scraper

Leon Agatic 1228015 e1228015@student.tuwien.ac.at
Brosch Florian 0727492 flo.brosch@gmail.com
Ivan Gusljesevic 1029600 e1029600@student.tuwien.ac.at
Robert Schäfer 1201110 e1201110@student.tuwien.ac.at
Michael Sobotka 0826407 michael.sobotka@gmx.at

## Document Model **(DOM, Text, anything in addition to these):**

The biggest disadvantage of screen scraper is the lack of xquery support or anything similar. Any extraction of web data has to be done via regular expressions. This includes important workflow steps like following links. Accessing a linked website requires at least three steps: parsing the link, extracting their values and using them as input for subsequent HTTP Get requests. Extracting web data from dynamically rendered html becomes a cumbersome thing since they often contain arbitrary data like images, tooltips and so on, making regular expressions unreliable and laborious.

## Form Filling and Macro Recording:

A proxy based approach is used to record macros and form-filling: The recording tool is used as a simple proxy server and a general purpose web browser is used to browse the web page. Tunneled http requests and responses are analyzed to extract all necessary information. (GET, POST-Data, the last content for analyzable files.)

The recorded pages are used to generate simple extractors. Pattern matching is used to trigger certain scripts / extract information.

However, scripts might be used to manipulate http headers as well.

## Transform original tree structure:

Since there is no way in Screen Scraper to access or interact with the DOM (see corresponding section) there is no possibility to transform the original tree structure in a clean fashion.

## Natural Language Processing Support:

There is no NLP available. However, scripting languages could be used to invoke external tools and libraries.

## Heuristics and Rule Generation:

Heuristics and rules simply correspond to the obtained page source. Regular expressions are used to extract information / mark individual content.

The provided user interface offers a predefined set of regular expressions to help to define variables in extracted code chunks.

## Image Recognition:

Image recognition is not supported. However, it is possible to download the image and to invoke external tools and libraries via scripts.

## Scripting:

The following scripting languages are supported:

- Interpreted Java
- Javascript
- Python

Several Joinpoints are defined to extend the basic behaviour with scripts.

Session:
- Before scraping session begins
- After scraping session ends
- Always at the end

File:
- Before file is scraped
- After file is scraped

Pattern matching:
- Before pattern is applied
- After pattern is applied
- Once if pattern is applied
- Once if no matches
- After each pattern match

## Ajax Support and DOM Freezing, DOM Event Support:

Ajax in Screen Scraper is supported via scripting. E.g. one can add a custom header of the POST data of an HTTP request in cases where a site is using AJAX, and the POST payload of a request is sent as XML. An example script that has to be invoked before the HTTP request is made would look like this:

```
// In script called "Before file is scraped"
// Sets the type of the POST entity to XML.
scrapeableFile.setContentType( "text/xml" );

// Set content of POST data
scrapeableFile.setRequestEntity( "<person><name>John Smith</name></person>" );
```

Screen Scraper has no support for DOM Freezing or DOM Events because it’s not able to parse the DOM at all (see corresponding section).

## Input and Output Formats:

### Input Formats:

Scripts are able to process any input file format. Extraction are html-based. However, it is quite easy to write transformers.

### Output Formats:

Scripts are used to generate the output. It is easily possible to write any format you want using javas FileWriter or OutputStreamWriter.

## Control Browser Settings like User Agent:

Higher browser settings are only editable by professional and enterprise edition. scrapeableFiles allows all paying users to manipulate the used http header.

## Parameterization:

There are several ways to pick up parameters from the external world:

- Load data via scripts (e.g. a xml file including all information we need)
- Load the scraper as a library and pass parameters to the session (e.g. write your own cmd parser, invoke the scraper via the provided library)
- Fetch parameters from the website directly

Variables can be easily extracted, inserted in several places and used inside scripts.

## Iteration, Conditions, Loops:

Two possibilities are available to scan through the content:

- Scripts

- Scraper and associated patterns

Scraper and Patterns are used to extract data and scripts are triggered on configurable conditions and states.

Scripts also have the ability to apply certain scraper.

It is easily possible to use scripts (including iterations, conditions, loops) to determine whether a file pattern should be applied and to decide whether a given extracted variable should be used.

See “scripting” for details.

## Robustness and Adaptation:

Screen scraper stores all of its files in a local database. If the software suffers a hard crash, the local database gets erased, and the only way to recover is an import from a backup database.

The extraction of data is primarily regex-based. The regular expressions are stored in a file, e.g. “News Page”. If there is a change on the “News Page”, the regex is most likely to break. The only chance to make the wrapper more robust is to write more general regex-patterns.

***Example:***
Instead of...

```
<span class=”info price”>Price: ~@PRICE@~</span>
```

...one could write

```
<span ~@css@~>Price: ~@PRICE@~</span>
```

...or even simpler

```
Price: ~@PRICE@~
```

## Automated Steps, Machine Learning **(from multiple examples):**

Neither any automated steps apart from the usual workflow nor any machine learning features are available with Screen Scraper.

## Storing screenshots / html source to file system:

There is no automated screenshot generation such as in Lixto.

To download HTML source, the session includes a simple mechanism to download files and to store them to the file system:

```
session.downloadFile (url, filename,[, numAttemps[, lazy]])
```

The html page for a scrapable file is returned by the following function:

String **scrapeableFile.getContentAsString** ( )

Basic IO functions (like a FileWriter) can be used to store the file.

## Performance and Scalability:

Screen scraper is fairly a lightweight application. The following performance/scalability factors are listed on the screen scraper wiki:

- Bandwidth
- Memory (see Settings for details)
- The mode (command line, server mode, workbench)
- The amount of session variables
- The way HTML is processed (tidy / not tidy)
- Scripting language (compiled languages are significant faster)
- The amount of parallel sessions
- Log level

The following tips are listed in the FAQ:[1]

- Provide more bandwidth to screen-scraper. Bandwidth is by far the biggest limiting factor that will determine how fast screen-scraper runs.
- Allocate more memory to screen-scraper. This can be done under the "Settings" dialog box (click the wrench icon) via the "Maximum memory allocation" setting.
- Run long scrapes either from the **command line** or in **server mode**. The workbench is really just designed for creating scraping sessions and such; if you try to run long scrapes from it you could encounter memory problems.
- Only save values in session variables when you have to. This is especially true for data sets extracted by extractor patterns. Each time you save a value in a session variable screen-scraper keeps it in memory for the life of the scraping session unless you explicitly null it out. For an extractor pattern, under the "Advanced" tab, when you click the "Automatically save the data set generated by this extractor pattern in a session variable" checkbox you're telling screen-scraper to retain that entire data set in memory. This is fine for relatively small data sets, but should be avoided for large ones. The performance hit for doing this can be mitigated by also checking the "Cache the data set" checkbox (also found under the "Advanced" tab), but when the value for the variable is requested screen-scraper will still need to read it into memory temporarily.
- Write data out as it gets extracted. This is a corollary to the previous point. Rather than saving data sets in memory you should instead write scripts that will either write the data out to a file or insert it into a database as it gets extracted. A common way of doing this is to write compiled Java code that takes a DataRecord containing extracted data, and handles inserting it into a database. See "**I'd like to insert the data screen-scraper**

[1] http://community.screen-scraper.com/FAQ/Optimizing

**extracts into a database. How do I do that?**" for more on this.

- Don't tidy HTML. This can make working with extractor patterns a bit trickier, but can save a fair amount on CPU usage. You can tell screen-scraper not to tidy HTML by unchecking the "Tidy HTML after scraping?" box found under the "Advanced" tab for a scrapeable file.
- Reuse objects. This is a general principle of programming, and should be followed when using screen-scraper. For example, if you're connecting to a database within screen-scraper scripts, rather than disconnecting and reconnecting each time you need to issue a SQL statement, you should instead keep a connection object in a session variable so that it can be reused (either that or use a connection pooling library).
- Use compiled code where possible. This will generally mean writing Java code, compiling it into a jar file, then placing it into screen-scraper's "lib/ext" folder. The jar will then be automatically added to screen-scraper's classpath such that you can refer to it in your scripts (e.g., you can include "import" statements in your scripts in order to use your classes).
- Reduce the number of scraping sessions you run in parallel. screen-scraper has the ability to run multiple scraping sessions simultaneously. This is often necessary and desirable, but it can also have an impact on memory usage and the performance of each scraping session. You can set the number of scraping sessions you'd like to allow screen-scraper to run simultaneously by opening the "Settings" dialog box (click on the wrench icon), then adjusting the value labeled "Maximum number of concurrent running scraping sessions".
- Avoid requesting files that are unnecessary. Oftentimes in order to get to the page containing the data you'd like to extract screen-scraper will need to first request a few other pages (e.g., one that handles logging in to the site). It's often worth it to experiment a bit by disabling certain files that you would normally request in your web browser (e.g., frames in a frameset) to see if they're actually required in order to be able to request the page containing the data you want.
- Fix extractor patterns that are timing out. Extractor patterns that time out can leave threads running which, over time, can consume a fair amount of memory. To see if your extractor patterns are timing out look for a message like this in your log: "Warning! The operation timed out while applying the extractor pattern, so it is being skipped." You should also try to add regular expressions to other tokens so as to make the match more precise. You can also often avoid timeouts by using sub-extractor patterns instead of full extractor patterns. This allows the extraction to be done in a more piecemeal fashion, which is more efficient.
- Disable logging. This can be done in the "Settings" window (click on the wrench icon) under the "Servers" section, by un-checking the box labeled "Generate log files". You should, of course, only do this, though, once you're satisfied that your scraping sessions are all working as you'd like them to.
- You may also wish to read a blog entry written by one of screen-scraper's developers about how to optimize large scrapes, specifically involving web-page iteration: **Techniques for Scraping Large Datasets**

## Ease of Use:

Screen Scraper is easy to use for anyone with very basic programming skills and very basic regex knowledge. However, it's almost impossible to use for anyone without them.

The GUI is file and not task-based. Users are forced to switch between several files during a single task. You usually lose the latest position in script files during a switch. This is not a big deal for small wrappers but a deal breaker for larger projects.

## Proxy Variations:

The following variations are exposed in the GUI:

- External proxy authentication (standard)
- External NT proxy authentication (NTLM)

## Captcha Support:

Captchas are not directly supported by Screen Scraper. However, it is quite easy to use external solutions via scripts.