Scrape

Scrape

Turn any url into clean data

Firecrawl converts web pages into markdown, ideal for LLM applications.

- It manages complexities: proxies, caching, rate limits, js-blocked content
- Handles dynamic content: dynamic websites, js-rendered sites, PDFs, images
- Outputs clean markdown, structured data, screenshots or html.

For details, see the **Scrape Endpoint API Reference**.

Scraping a URL with Firecrawl

/scrape endpoint

Used to scrape a URL and get its content.

Installation

Python Node Go Rust

```
pip install firecrawl-py
```

Usage

```
from firecrawl import FirecrawlApp

app = FirecrawlApp(api_key="fc-YOUR_API_KEY")

# Scrape a website:
scrape_result = app.scrape_url('firecrawl.dev', params={'formats': ['markdown', '
print(scrape_result)
```

For more details about the parameters, refer to the **API Reference**.

Response

SDKs will return the data object directly. cURL will return the payload exactly as shown below.

```
{
  "success": true,
  "data" : {
    "markdown": "Launch Week I is here! [See our Day 2 Release 🚀](https://www.fi
    "html": "<!DOCTYPE html><html lang=\"en\" class=\"light\" style=\"color-schem
    "metadata": {
      "title": "Home - Firecrawl",
      "description": "Firecrawl crawls and converts any website into clean markdo
      "language": "en",
      "keywords": "Firecrawl,Markdown,Data,Mendable,Langchain",
      "robots": "follow, index",
      "ogTitle": "Firecrawl",
      "ogDescription": "Turn any website into LLM-ready data.",
      "ogUrl": "https://www.firecrawl.dev/",
      "ogImage": "https://www.firecrawl.dev/og.png?123",
      "ogLocaleAlternate": [],
      "ogSiteName": "Firecrawl",
      "sourceURL": "https://firecrawl.dev",
      "statusCode": 200
```

```
        }
    }
}
```

Firecrawl

Scrape › **Scrape**

Scrape Formats

You can now choose what formats you want your output in. You can specify multiple output formats. Supported formats are:

- Markdown (markdown)
- HTML (html)
- Raw HTML (rawHtml) (with no modifications)
- Screenshot (screenshot or screenshot@fullPage)
- Links (links)
- Extract (extract) - structured output

Output keys will match the format you choose.

Extract structured data

/scrape (with extract) endpoint

Used to extract structured data from scraped pages.

Python Node cURL

```
from firecrawl import FirecrawlApp
from pydantic import BaseModel, Field

# Initialize the FirecrawlApp with your API key
app = FirecrawlApp(api_key='your_api_key')

class ExtractSchema(BaseModel):
    company_mission: str
    supports_sso: bool
    is_open_source: bool
    is_in_yc: bool

data = app.scrape_url('https://docs.firecrawl.dev/', {
    'formats': ['json'],
    'jsonOptions': {
        'schema': ExtractSchema.model_json_schema(),
    }
})
print(data["json"])
```

Output:

JSON

```
{
    "success": true,
    "data": {
      "json": {
        "company_mission": "Train a secure AI on your technical resources that an
        "supports_sso": true,
        "is_open_source": false,
        "is_in_yc": true
      },
      "metadata": {
        "title": "Mendable",
        "description": "Mendable allows you to easily build AI chat applications.
        "robots": "follow, index",
        "ogTitle": "Mendable",
        "ogDescription": "Mendable allows you to easily build AI chat application
        "ogUrl": "https://docs.firecrawl.dev/",
        "ogImage": "https://docs.firecrawl.dev/mendable_new_og1.png",
        "ogLocaleAlternate": [],
        "ogSiteName": "Mendable",
        "sourceURL": "https://docs.firecrawl.dev/"
      },
    }
}
```

Firecrawl

Scrape › **Scrape**

Extracting without schema (New)

You can now extract without a schema by just passing a `prompt` to the endpoint. The llm chooses the structure of the data.

cURL

```
curl -X POST https://api.firecrawl.dev/v1/scrape \
    -H 'Content-Type: application/json' \
    -H 'Authorization: Bearer YOUR_API_KEY' \
    -d '{
      "url": "https://docs.firecrawl.dev/",
      "formats": ["json"],
      "jsonOptions": {
        "prompt": "Extract the company mission from the page."
      }
    }'
```

Output:

JSON

```
{
    "success": true,
    "data": {
      "json": {
        "company_mission": "Train a secure AI on your technical resources that an
      },
      "metadata": {
        "title": "Mendable",
        "description": "Mendable allows you to easily build AI chat applications.
        "robots": "follow, index",
        "ogTitle": "Mendable",
        "ogDescription": "Mendable allows you to easily build AI chat application
        "ogUrl": "https://docs.firecrawl.dev/",
        "ogImage": "https://docs.firecrawl.dev/mendable_new_og1.png",
        "ogLocaleAlternate": [],
        "ogSiteName": "Mendable",
        "sourceURL": "https://docs.firecrawl.dev/"
      },
    }
}
```

Extract object

The `extract` object accepts the following parameters:

- `schema` : The schema to use for the extraction.
- `systemPrompt` : The system prompt to use for the extraction.
- `prompt` : The prompt to use for the extraction without a schema.

Interacting with the page with Actions

Firecrawl allows you to perform various actions on a web page before scraping its content. This is particularly useful for interacting with dynamic content, navigating through pages, or accessing content that requires user interaction.

Here is an example of how to use actions to navigate to google.com, search for Firecrawl, click on the first result, and take a screenshot.

It is important to almost always use the `wait` action before/after executing other actions to give enough time for the page to load.

Example

Python Node cURL

```python
from firecrawl import FirecrawlApp

app = FirecrawlApp(api_key="fc-YOUR_API_KEY")

# Scrape a website:
scrape_result = app.scrape_url('firecrawl.dev',
    params={
        'formats': ['markdown', 'html'],
        'actions': [
            {"type": "wait", "milliseconds": 2000},
            {"type": "click", "selector": "textarea[title=\"Search\"]"},
            {"type": "wait", "milliseconds": 2000},
            {"type": "write", "text": "firecrawl"},
            {"type": "wait", "milliseconds": 2000},
            {"type": "press", "key": "ENTER"},
            {"type": "wait", "milliseconds": 3000},
            {"type": "click", "selector": "h3"},
            {"type": "wait", "milliseconds": 3000},
            {"type": "scrape"},
            {"type": "screenshot"}
        ]
    }
)
print(scrape_result)
```

Output

JSON

```json
{
  "success": true,
  "data": {
    "markdown": "Our first Launch Week is over! [See the recap 🚀](blog/firecraw
    "actions": {
      "screenshots": [
        "https://alttmdsdujxrfnakrkyi.supabase.co/storage/v1/object/public/media
      ],
      "scrapes": [
        {
          "url": "https://www.firecrawl.dev/",
          "html": "<html><body><h1>Firecrawl</h1></body></html>"
        }
      ]
    },
    "metadata": {
      "title": "Home - Firecrawl",
      "description": "Firecrawl crawls and converts any website into clean markd
      "language": "en",
      "keywords": "Firecrawl,Markdown,Data,Mendable,Langchain",
      "robots": "follow, index",
      "ogTitle": "Firecrawl",
      "ogDescription": "Turn any website into LLM-ready data.",
      "ogUrl": "https://www.firecrawl.dev/",
      "ogImage": "https://www.firecrawl.dev/og.png?123",
      "ogLocaleAlternate": [],
      "ogSiteName": "Firecrawl",
      "sourceURL": "http://google.com",
      "statusCode": 200
    }
  }
}
```

For more details about the actions parameters, refer to the **API Reference**.

Location and Language

Specify country and preferred languages to get relevant content based on your target location and language preferences.

How it works

When you specify the location settings, Firecrawl will use an appropriate proxy if available and emulate the corresponding language and timezone settings. By default, the location is set to ‘US’ if not specified.

Usage

To use the location and language settings, include the `location` object in your request body with the following properties:

- `country` : ISO 3166-1 alpha-2 country code (e.g., ‘US’, ‘AU’, ‘DE’, ‘JP’). Defaults to ‘US’.
- `languages` : An array of preferred languages and locales for the request in order of priority. Defaults to the language of the specified location.

Python Node cURL

```
from firecrawl import FirecrawlApp

app = FirecrawlApp(api_key="fc-YOUR_API_KEY")

# Scrape a website:
scrape_result = app.scrape_url('airbnb.com',
    params={
        'formats': ['markdown', 'html'],
        'location': {
            'country': 'BR',
            'languages': ['pt-BR']
        }
    }
)
print(scrape_result)
```

Batch scraping multiple URLs

You can now batch scrape multiple URLs at the same time. It takes the starting URLs and optional parameters as arguments. The params argument allows you to specify additional options for the batch scrape job, such as the output formats.

How it works

It is very similar to how the `/crawl` endpoint works. It submits a batch scrape job and returns a job ID to check the status of the batch scrape.

The sdk provides 2 methods, synchronous and asynchronous. The synchronous method will return the results of the batch scrape job, while the asynchronous method will return a job ID that you can use to check the status of the batch scrape.

Usage

Python Node cURL

```
from firecrawl import FirecrawlApp

app = FirecrawlApp(api_key="fc-YOUR_API_KEY")

# Scrape multiple websites:
batch_scrape_result = app.batch_scrape_urls(['firecrawl.dev', 'mendable.ai'], {'f
print(batch_scrape_result)

# Or, you can use the asynchronous method:
batch_scrape_job = app.async_batch_scrape_urls(['firecrawl.dev', 'mendable.ai'],
print(batch_scrape_job)

# (async) You can then use the job ID to check the status of the batch scrape:
batch_scrape_status = app.check_batch_scrape_status(batch_scrape_job['id'])
print(batch_scrape_status)
```

Response

If you're using the sync methods from the SDKs, it will return the results of the batch scrape job. Otherwise, it will return a job ID that you can use to check the status of the batch scrape.

Synchronous

```
{
  "status": "completed",
  "total": 36,
  "completed": 36,
  "creditsUsed": 36,
  "expiresAt": "2024-00-00T00:00:00.000Z",
  "next": "https://api.firecrawl.dev/v1/batch/scrape/123-456-789?skip=26",
  "data": [
    {
      "markdown": "[Firecrawl Docs home page![light logo](https://mintlify.s3-us-
      "html": "<!DOCTYPE html><html lang=\"en\" class=\"js-focus-visible lg:[--sc
      "metadata": {
        "title": "Build a 'Chat with website' using Groq Llama 3 | Firecrawl",
        "language": "en",
        "sourceURL": "https://docs.firecrawl.dev/learn/rag-llama3",
        "description": "Learn how to use Firecrawl, Groq Llama 3, and Langchain t
        "ogLocaleAlternate": [],
        "statusCode": 200
      }
    },
    ...
  ]
}
```

Asynchronous

You can then use the job ID to check the status of the batch scrape by calling the `/batch/scrape/{id}` endpoint. This endpoint is meant to be used while the job is still running or right after it has completed **as batch scrape jobs expire after 24 hours**.

```
{
  "success": true,
  "id": "123-456-789",
  "url": "https://api.firecrawl.dev/v1/batch/scrape/123-456-789"
}
```

Firecrawl

Suggest edits Raise issue

Scrape › **Scrape**

‹ **Advanced Scraping Guide** **Batch Scrape** ›

Powered by Mintlify