



v1 ▾

Get Started ▸ Quickstart

Get Started

Quickstart

Firecrawl allows you to turn entire websites into LLM-ready markdown



Welcome to Firecrawl

Firecrawl is an API service that takes a URL, crawls it, and converts it into clean markdown. We crawl all accessible subpages and give you clean markdown for each. No sitemap required.

How to use it?



Get Started > **Quickstart**

- ☒ **API: Documentation**
- ☒ **SDKs: Python, Node, Go, Rust**
- ☒ **LLM Frameworks: Langchain (python), Langchain (js), Llama Index, Crew.ai, Composio, PraisonAI, Superinterface, Vectorize**
- ☒ **Low-code Frameworks: Dify, Langflow, Flowise AI, Cargo, Pipedream**
- ☒ **Others: Zapier, Pabbly Connect**
- ☐ Want an SDK or Integration? Let us know by opening an issue.

Self-host: To self-host refer to guide [here](#).

API Key

To use the API, you need to sign up on **Firecrawl** and get an API key.

Features

Scrape: scrapes a URL and get its content in LLM-ready format (markdown, structured data via **LLM Extract**, screenshot, html)

Crawl: scrapes all the URLs of a web page and return content in LLM-ready format

Map: input a website and get all the website urls - extremely fast

Extract: get structured data from single page, multiple pages or entire websites with AI.

Powerful Capabilities

LLM-ready formats: markdown, structured data, screenshot, HTML, links, metadata

The hard stuff: proxies, anti-bot mechanisms, dynamic content (js-rendered), output parsing, orchestration



Get Started > **Quickstart**

Actions: click, scroll, input, wait and more before extracting data

You can find all of Firecrawl's capabilities and how to use them in our **documentation**

Crawling

Used to crawl a URL and all accessible subpages. This submits a crawl job and returns a job ID to check the status of the crawl.

Installation

Python Node Go Rust

```
pip install firecrawl-py
```

Usage

Python Node Go Rust cURL

```
from firecrawl import FirecrawlApp

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

# Crawl a website:
crawl_status = app.crawl_url(
    'https://firecrawl.dev',
    params={
        'limit': 100,
        'scrapeOptions': {'formats': ['markdown', 'html']}
```



[Get Started](#) > **Quickstart**

If you're using `cURL` or `async crawl` functions on SDKs, this will return an `ID` where you can use to check the status of the crawl.

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

Check Crawl Job

Used to check the status of a crawl job and get its result.

[Python](#) [Node](#) [Go](#) [Rust](#) [cURL](#)

```
crawl_status = app.check_crawl_status("<crawl_id>")
print(crawl_status)
```

Response

The response will be different depending on the status of the crawl. For not completed or large responses exceeding 10MB, a `next` URL parameter is provided. You must request this URL to retrieve the next 10MB of data. If the `next` parameter is absent, it indicates the end of the crawl data.

[Scraping](#) [Completed](#)

```
{
  "status": "scraping",
  "total": 36,
```



Get Started > **Quickstart**

```
{
  "markdown": "[Firecrawl Docs home page! [light logo](https://mintlify.s3-us
  "html": "<!DOCTYPE html><html lang=\"en\" class=\"js-focus-visible lg:[--s
  "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
    "ogLocaleAlternate": [],
    "statusCode": 200
  }
},
...
]
```

Scraping

To scrape a single URL, use the `scrape_url` method. It takes the URL as a parameter and returns the scraped data as a dictionary.

Python Node Go Rust cURL

```
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)
```



Get Started > **Quickstart**

```
{
  "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
    }
  }
}
```

Extraction

With LLM extraction, you can easily extract structured data from any URL. We support pydantic schemas to make it easier for you too. Here is how you to use it:

v1 is only supported on node, python and cURL at this time.

Python Node cURL



Get Started > **Quickstart**

```
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",
```



Get Started > **Quickstart**

```
    "sourceURL": "https://docs.firecrawl.dev/"
  },
}
```

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": {
```




Get Started > **Quickstart**

```
"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/"
},
}
```

Extraction (v0)

Python

JavaScript

Go

Rust

cURL

```
app = FirecrawlApp(version="v0")
```

```
class ArticleSchema(BaseModel):
```

```
    title: str
```

```
    points: int
```

```
    by: str
```

```
    commentsURL: str
```

```
class TopArticlesSchema(BaseModel):
```

```
top: List[ArticleSchema] = Field(..., max_items=5, description="Top 5 stories")
```

```
data = app.scrape_url('https://news.ycombinator.com', {
```

```
    'extractorOptions': {
```

```
        'extractionSchema': TopArticlesSchema.model_json_schema(),
```

```
        'mode': 'llm-extraction'
```

```
    },
```



Get Started > **Quickstart**

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

```
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},
```



Get Started > **Quickstart**

```
        {"type": "screenshot"}
    ]
}
)
print(scrape_result)
```

Output

JSON

```
{
  "success": true,
  "data": {
    "markdown": "Our first Launch Week is over! [See the recap 🚀](blog/firecrawl",
    "actions": {
      "screenshots": [
        "https://alttmdsdujxrfrnakrkyi.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 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/"
    }
  }
}
```



Get Started > Quickstart

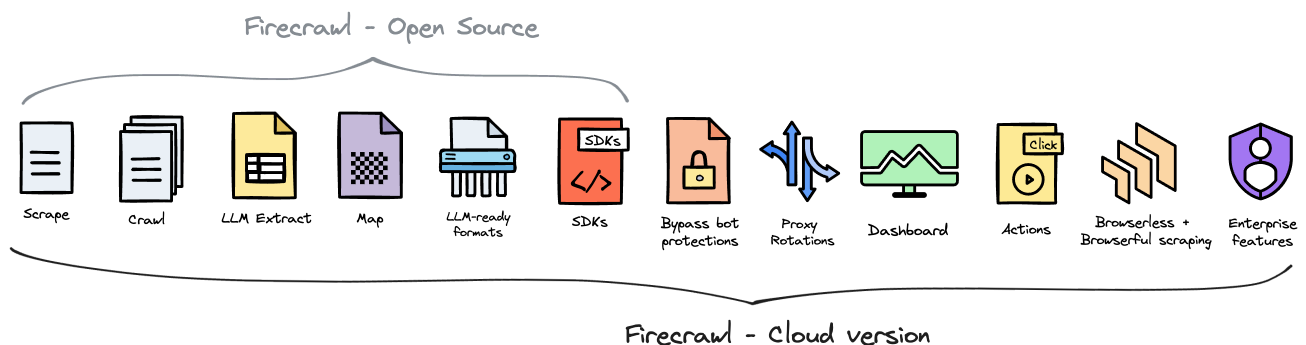
```
}  
}  
}
```

Open Source vs Cloud

Firecrawl is open source available under the **AGPL-3.0 license**.

To deliver the best possible product, we offer a hosted version of Firecrawl alongside our open-source offering. The cloud solution allows us to continuously innovate and maintain a high-quality, sustainable service for all users.

Firecrawl Cloud is available at **firecrawl.dev** and offers a range of features that are not available in the open source version:



Contributing

We love contributions! Please read our **contributing guide** before submitting a pull request.



[Get Started](#) > **Quickstart**

Powered by Mintlify



Scrape

Batch Scrape

Batch scrape multiple URLs

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'], {'
```

```
print(batch_scrape_result)
```

 **Firecrawl**

Or, you can use the asynchronous method:

```
batch_scrape_job = app.async_batch_scrape_urls(['firecrawl.dev', 'mendable.ai'],
Scrape > Batch Scrape
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

Completed

```
{
  "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

# Scrape multiple websites:
batch_scrape_result = app.batch_scrape_urls(
    ['https://docs.firecrawl.dev', 'https://docs.firecrawl.dev/sdks/overview'],
    {
        'formats': ['extract'],

```


**Firecrawl**

```

        'extract': {
            'prompt': 'Extract the title and description from the page.',
            'schema': {
                'type': 'object',
                'properties': {
                    'title': {'type': 'string'},
                    'description': {'type': 'string'}
                },
                'required': ['title', 'description']
            }
        }
    }
)
print(batch_scrape_result)

# Or, you can use the asynchronous method:
batch_scrape_job = app.async_batch_scrape_urls(
    ['https://docs.firecrawl.dev', 'https://docs.firecrawl.dev/sdks/overview'],
    {
        'formats': ['extract'],
        'extract': {
            'prompt': 'Extract the title and description from the page.',
            'schema': {
                'type': 'object',
                'properties': {
                    'title': {'type': 'string'},
                    'description': {'type': 'string'}
                },
                'required': ['title', 'description']
            }
        }
    }
)
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

Firecrawl

Synchronous

Scrape > **Batch Scrape**


Completed


```
{
  "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": [
    {
      "extract": {
        "title": "Build a 'Chat with website' using Groq Llama 3 | Firecrawl",
        "description": "Learn how to use Firecrawl, Groq Llama 3, and Langchain to"
      }
    },
    ...
  ]
}
```

Asynchronous

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



 Suggest edits

 Raise issue

Scrape > **Batch Scrape**
< Scrape

LLM Extract >

Powered by Mintlify

Features

Crawl

Firecrawl can recursively search through a urls subdomains, and gather the content

Firecrawl thoroughly crawls websites, ensuring comprehensive data extraction while bypassing any web blocker mechanisms. Here's how it works:

1. **URL Analysis:** Begins with a specified URL, identifying links by looking at the sitemap and then crawling the website. If no sitemap is found, it will crawl the website following the links.
2. **Recursive Traversal:** Recursively follows each link to uncover all subpages.
3. **Content Scraping:** Gathers content from every visited page while handling any complexities like JavaScript rendering or rate limits.
4. **Result Compilation:** Converts collected data into clean markdown or structured output, perfect for LLM processing or any other task.

This method guarantees an exhaustive crawl and data collection from any starting URL.

Crawling

/crawl endpoint

Used to crawl a URL and all accessible subpages. This submits a crawl job and returns a job ID to check the status of the crawl.



Features > **Crawl**

Installation

Python Node Go Rust

```
pip install firecrawl-py
```

Usage

Python Node Go Rust cURL

```
from firecrawl import FirecrawlApp

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

# Crawl a website:
crawl_status = app.crawl_url(
    'https://firecrawl.dev',
    params={
        'limit': 100,
        'scrapeOptions': {'formats': ['markdown', 'html']}
    },
    poll_interval=30
)
print(crawl_status)
```

Response

If you're using cURL or `async crawl` functions on SDKs, this will return an `ID` where you can use to check the status of the crawl.



Features > **Crawl**

}

Check Crawl Job

Used to check the status of a crawl job and get its result.

! This endpoint only works for crawls that are in progress or crawls that have completed recently.

Python Node Go Rust cURL

```
crawl_status = app.check_crawl_status("<crawl_id>")
print(crawl_status)
```

Response Handling

The response varies based on the crawl's status.

For not completed or large responses exceeding 10MB, a `next` URL parameter is provided. You must request this URL to retrieve the next 10MB of data. If the `next` parameter is absent, it indicates the end of the crawl data.

The skip parameter sets the maximum number of results returned for each chunk of results returned.

i The skip and next parameter are only relevant when hitting the api directly. If you're using the SDK, we handle this for you and will return all the results at once.

Scraping Completed

[Features](#) > **Crawl**

```

"creditsUsed": 10,
"expiresAt": "2024-00-00T00:00:00.000Z",
"next": "https://api.firecrawl.dev/v1/crawl/123-456-789?skip=10",
"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
    }
  },
  ...
]
}

```

Crawl WebSocket

Firecrawl's WebSocket-based method, `Crawl URL and Watch`, enables real-time data extraction and monitoring. Start a crawl with a URL and customize it with options like page limits, allowed domains, and output formats, ideal for immediate data processing needs.

[Python](#)[Node](#)

```

# inside an async function...
nest_asyncio.apply()

# Define event handlers

```



Features > **Crawl**

```
def on_done(detail):
    print("DONE", detail['status'])

    # Function to start the crawl and watch process
async def start_crawl_and_watch():
    # Initiate the crawl job and get the watcher
    watcher = app.crawl_url_and_watch('firecrawl.dev', { 'excludePaths': ['blog/

    # Add event listeners
    watcher.add_event_listener("document", on_document)
    watcher.add_event_listener("error", on_error)
    watcher.add_event_listener("done", on_done)

    # Start the watcher
    await watcher.connect()

# Run the event loop
await start_crawl_and_watch()
```

Crawl Webhook

You can now pass a `webhook` parameter to the `/crawl` endpoint. This will send a POST request to the URL you specify when the crawl is started, updated and completed.

The webhook will now trigger for every page crawled and not just the whole result at the end.

cURL

```
curl -X POST https://api.firecrawl.dev/v1/crawl \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_API_KEY' \
```




Features > **Crawl**

Webhook Events

There are now 4 types of events:

`crawl.started` - Triggered when the crawl is started.

`crawl.page` - Triggered for every page crawled.

`crawl.completed` - Triggered when the crawl is completed to let you know it's done
(Beta)**

`crawl.failed` - Triggered when the crawl fails.

Webhook Response

`success` - If the webhook was successful in crawling the page correctly.

`type` - The type of event that occurred.

`id` - The ID of the crawl.

`data` - The data that was scraped (Array). This will only be non empty on `crawl.page` and will contain 1 item if the page was scraped successfully. The response is the same as the `/scrape` endpoint.

`error` - If the webhook failed, this will contain the error message.

**Beta consideration

There is a very tiny chance that the `crawl.completed` event may be triggered while the final `crawl.page` events are still being processed. We're working on a fix for this.



Features > **Crawl**

Powered by Mintlify



Integrations

CrewAI

Learn how to use Firecrawl with CrewAI

Using Firecrawl with CrewAI

Firecrawl is integrated with **CrewAI**, the framework for orchestrating AI agents. This page introduces all of the Firecrawl tools added to the framework.

Installing Firecrawl Tools inside of CrewAI

Get an API key from your **firecrawl.dev dashboard** and set it in environment variables (`FIRECRAWL_API_KEY`).

Install the **Firecrawl SDK** along with `crewai[tools]` package:

```
pip install firecrawl-py 'crewai[tools]'
```

Tools

FirecrawlCrawlWebsiteTool

Example

Utilize the `FirecrawlScrapeFromWebsiteTool` as follows to allow your agent to load websites:

```
from crewai_tools import FirecrawlCrawlWebsiteTool
```



Integrations > CrewAI

api_key : Optional. Specifies Firecrawl API key. Defaults is the `FIRECRAWL_API_KEY` environment variable.

url : The base URL to start crawling from.

page_options : Optional.

onlyMainContent : Optional. Only return the main content of the page excluding headers, navs, footers, etc.

includeHtml : Optional. Include the raw HTML content of the page. Will output a `html` key in the response.

crawler_options : Optional. Options for controlling the crawling behavior.

includes : Optional. URL patterns to include in the crawl.

exclude : Optional. URL patterns to exclude from the crawl.

generateImgAltText : Optional. Generate alt text for images using LLMs (requires a paid plan).

returnOnlyUrls : Optional. If true, returns only the URLs as a list in the crawl status. Note: the response will be a list of URLs inside the data, not a list of documents.

maxDepth : Optional. Maximum depth to crawl. Depth 1 is the base URL, depth 2 includes the base URL and its direct children, and so on.

mode : Optional. The crawling mode to use. Fast mode crawls 4x faster on websites without a sitemap but may not be as accurate and shouldn't be used on heavily JavaScript-rendered websites.

limit : Optional. Maximum number of pages to crawl.

timeout : Optional. Timeout in milliseconds for the crawling operation.

FirecrawlScrapeWebsiteTool



Integrations > **CrewAI**

```
tool = FirecrawlScrapeWebsiteTool(url='firecrawl.dev')
```

Arguments

api_key : Optional. Specifies Firecrawl API key. Defaults is the `FIRECRAWL_API_KEY` environment variable.

url : The URL to scrape.

page_options : Optional.

onlyMainContent : Optional. Only return the main content of the page excluding headers, navs, footers, etc.

includeHtml : Optional. Include the raw HTML content of the page. Will output a `html` key in the response.

extractor_options : Optional. Options for LLM-based extraction of structured information from the page content

mode : The extraction mode to use, currently supports 'llm-extraction'

extractionPrompt : Optional. A prompt describing what information to extract from the page

extractionSchema : Optional. The schema for the data to be extracted

timeout : Optional. Timeout in milliseconds for the request

FirecrawlSearchTool

Example

Utilize the FirecrawlSearchTool as follows to allow your agent to load websites:



Integrations > **CrewAI**

Arguments

api_key : Optional. Specifies Firecrawl API key. Defaults is the `FIRECRAWL_API_KEY` environment variable.

query : The search query string to be used for searching.

page_options : Optional. Options for result formatting.

onlyMainContent : Optional. Only return the main content of the page excluding headers, navs, footers, etc.

includeHtml : Optional. Include the raw HTML content of the page. Will output a `html` key in the response.

fetchPageContent : Optional. Fetch the full content of the page.

search_options : Optional. Options for controlling the crawling behavior.

limit : Optional. Maximum number of pages to crawl.

 Suggest edits

 Raise issue

< [Llamaindex](#)

[Dify](#) >

Powered by Mintlify



v1 ▾

 Scrape > LLM Extract

Scrape

LLM Extract

Extract structured data from pages via LLMs

Scrape and extract structured data with Firecrawl

Firecrawl leverages Large Language Models (LLMs) to efficiently extract structured data from web pages. Here's how:

1. **Schema Definition:** Define the URL to scrape and the desired data schema using JSON Schema (following OpenAI tool schema). This schema specifies the data structure you expect to extract from the page.
2. **Scrape Endpoint:** Pass the URL and the schema to the scrape endpoint. Documentation for this endpoint can be found here: **Scrape Endpoint Documentation**
3. **Structured Data Retrieval:** Receive the scraped data in the structured format defined by your schema. You can then use this data as needed in your application or for further processing.

This method streamlines data extraction, reducing manual handling and enhancing efficiency.

Extract structured data

/scrape (with extract) endpoint

Used to extract structured data from scraped pages.

```
from firecrawl import FirecrawlApp
Scrape > LLM Extract
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."
    }
  }
}
```


**Firecrawl**

```

    "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/"
  },
}
}

```

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": {
      "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.

[✎ Suggest edits](#)
[⚠ Raise issue](#)
[< Batch Scrape](#)
[Crawl >](#)



Powered by Mintlify

Scrape > **LLM Extract**



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
Scrape > Scrape

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
    }
  }
}

```



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
```



Scrape > **Scrape**

```
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/"
    },
  },
}
```



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



```
from Firecrawl import FirecrawlApp
```

```
app = FirecrawlApp(api_key="fc-YOUR_API_KEY")
Scrape > Scrape
```

```
# 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

```
{
  "success": true,
  "data": {
    "markdown": "Our first Launch Week is over! [See the recap 🚀](blog/firecrawl)",
    "actions": {
      "screenshots": [
        "https://alttmdsdujxrfrnakrkyi.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 marked
  "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

Firecrawl

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-1-1.amazonaws.com/firecrawl-docs-assets/logo-light.png)](https://docs.firecrawl.dev)",
      "html": "<!DOCTYPE html><html lang=\"en\" class=\"js-focus-visible lg:[--scrollbar-gutter:stable]\">
      \"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 to build a RAG application.\",
        \"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"
}
```



 Suggest edits

 Raise issue

Scrape > **Scrape**
< **Advanced Scraping Guide**

Batch Scrape >

Powered by Mintlify



Get Started

Welcome to V1

Firecrawl allows you to turn entire websites into LLM-ready markdown

Firecrawl V1 is here! With that we introduce a more reliable and developer friendly API.

Here is what's new:

Output Formats for `/scrape` . Choose what formats you want your output in.

New `/map` **endpoint** for getting most of the URLs of a webpage.

Developer friendly API for `/crawl/{id}` status.

2x Rate Limits for all plans.

Go SDK and Rust SDK

Teams support

API Key Management in the dashboard.

`onlyMainContent` is now default to `true` .

`/crawl` webhooks and websocket support.

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)



Get Started > **Welcome to V1**

Output keys will match the format you choose.

Python Node Go Rust cURL

```
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']})
print(scrape_result)
```

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.firecrawl.dev/)",
    "html": "<!DOCTYPE html><html lang=\"en\" class=\"light\" style=\"color-scheme: light;\"><body><div><h1>Launch Week I is here! [See our Day 2 Release 🚀](https://www.firecrawl.dev/)</h1></div></body></html>",
    "metadata": {
      "title": "Home - Firecrawl",
      "description": "Firecrawl crawls and converts any website into clean markdown",
      "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",
    }
  }
}
```



Get Started > **Welcome to V1**

```
}  
}
```

Introducing /map (Alpha)

The easiest way to go from a single url to a map of the entire website.

Usage

Python Node Go Rust cURL

```
from firecrawl import FirecrawlApp  
  
app = FirecrawlApp(api_key="fc-YOUR_API_KEY")  
  
# Map a website:  
map_result = app.map_url('https://firecrawl.dev')  
print(map_result)
```

Response

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

```
{  
  "status": "success",  
  "links": [  
    "https://firecrawl.dev",  
    "https://www.firecrawl.dev/pricing",
```



Get Started > **Welcome to V1**

}

WebSockets

To crawl a website with WebSockets, use the `Crawl URL` and `Watch` method.

Python

Node

```
# inside an async function...
nest_asyncio.apply()

# Define event handlers
def on_document(detail):
    print("DOC", detail)

def on_error(detail):
    print("ERR", detail['error'])

def on_done(detail):
    print("DONE", detail['status'])

# Function to start the crawl and watch process
async def start_crawl_and_watch():
    # Initiate the crawl job and get the watcher
    watcher = app.crawl_url_and_watch('firecrawl.dev', { 'excludePaths': ['blog/']

    # Add event listeners
    watcher.add_event_listener("document", on_document)
    watcher.add_event_listener("error", on_error)
    watcher.add_event_listener("done", on_done)

    # Start the watcher
    await watcher.connect()
```



[Get Started](#) > **Welcome to V1**

Extract format

LLM extraction is now available in v1 under the `extract` format. To extract structured from a page, you can pass a schema to the endpoint or just provide a prompt.

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



Get Started > **Welcome to V1**

```
"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/"
},
}
```

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"],
```



[Get Started](#) > **Welcome to V1**

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/"
    },
  },
}
```

New Crawl Webhook

You can now pass a `webhook` parameter to the `/crawl` endpoint. This will send a POST request to the URL you specify when the crawl is started, updated and completed.



Get Started > **Welcome to V1**

```
curl -X POST https://api.firecrawl.dev/v1/crawl \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_API_KEY' \
  -d '{
    "url": "https://docs.firecrawl.dev",
    "limit": 100,
    "webhook": "https://example.com/webhook"
  }'
```

Webhook Events

There are now 4 types of events:

`crawl.started` - Triggered when the crawl is started.

`crawl.page` - Triggered for every page crawled.

`crawl.completed` - Triggered when the crawl is completed to let you know it's done.

`crawl.failed` - Triggered when the crawl fails.

Webhook Response

`success` - If the webhook was successful in crawling the page correctly.

`type` - The type of event that occurred.

`id` - The ID of the crawl.

`data` - The data that was scraped (Array). This will only be non empty on `crawl.page` and will contain 1 item if the page was scraped successfully. The response is the same as the `/scrape` endpoint.

`error` - If the webhook failed, this will contain the error message.



Get Started > **Welcome to V1**

/scrape endpoint

The updated `/scrape` endpoint has been redesigned for enhanced reliability and ease of use. The structure of the new `/scrape` request body is as follows:

```
{
  "url": "<string>",
  "formats": ["markdown", "html", "rawHtml", "links", "screenshot", "json"],
  "includeTags": ["<string>"],
  "excludeTags": ["<string>"],
  "headers": { "key": "value" },
  "waitFor": 123,
  "timeout": 123
}
```

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)

JSON (json)

By default, the output will include only the markdown format.



Get Started > **Welcome to V1**

Parameter	Change	Description
<code>onlyIncludeTags</code>	Moved and Renamed	Moved to root level. And renamed to <code>includeTags</code> .
<code>removeTags</code>	Moved and Renamed	Moved to root level. And renamed to <code>excludeTags</code> .
<code>onlyMainContent</code>	Moved	Moved to root level. <code>true</code> by default.
<code>waitFor</code>	Moved	Moved to root level.
<code>headers</code>	Moved	Moved to root level.
<code>parsePDF</code>	Moved	Moved to root level.
<code>extractorOptions</code>	No Change	
<code>timeout</code>	No Change	
<code>pageOptions</code>	Removed	No need for <code>pageOptions</code> parameter. The scrape options were moved to root level.
<code>replaceAllPathsWithAbsolutePaths</code>	Removed	<code>replaceAllPathsWithAbsolutePaths</code> is not needed anymore. Every path is now default to absolute path.
<code>includeHtml</code>	Removed	add <code>"html"</code> to <code>formats</code> instead.
<code>includeRawHtml</code>	Removed	add <code>"rawHtml"</code> to <code>formats</code> instead.
<code>screenshot</code>	Removed	add <code>"screenshot"</code> to <code>formats</code> instead.
<code>fullPageScreenshot</code>	Removed	add <code>"screenshot@fullPage"</code> to <code>formats</code> instead.
<code>extractorOptions</code>	Removed	Use <code>"extract"</code> format instead with <code>extract</code> object.

The new `extract` format is described in the **llm-extract** section.

/crawl endpoint



Get Started > **Welcome to V1**

```
"url": "<string>",
"excludePaths": [<string>],
"includePaths": [<string>],
"maxDepth": 2,
"ignoreSitemap": true,
"limit": 10,
"allowBackwardLinks": true,
"allowExternalLinks": true,
"scrapeOptions": {
  // same options as in /scrape
  "formats": ["markdown", "html", "rawHtml", "screenshot", "links"],
  "headers": { "<key>": "<value>" },
  "includeTags": [<string>],
  "excludeTags": [<string>],
  "onlyMainContent": true,
  "waitFor": 123
}
}
```

Details on the new request body

The table below outlines the changes to the request body parameters for the `/crawl` endpoint in V1.

Parameter	Change	Description
<code>pageOptions</code>	Renamed	Renamed to <code>scrapeOptions</code> .
<code>includes</code>	Moved and Renamed	Moved to root level. Renamed to <code>includePaths</code> .
<code>excludes</code>	Moved and Renamed	Moved to root level. Renamed to <code>excludePaths</code> .
<code>allowBackwardCrawling</code>	Moved and Renamed	Moved to root level. Renamed to <code>allowBackwardLinks</code> .



Get Started > **Welcome to V1**

<code>ignoreSitemap</code>	Moved	Moved to root level.
<code>limit</code>	Moved	Moved to root level.
<code>crawlerOptions</code>	Removed	No need for <code>crawlerOptions</code> parameter. The crawl options were moved to root level.
<code>timeout</code>	Removed	Use <code>timeout</code> in <code>scrapeOptions</code> instead.

Suggest edits

Raise issue

< **Launch Week II (New)**

Rate Limits >

Powered by Mintlify