

Firecrawl API Documentation

Firecrawl transforms websites into clean, LLM-ready data through a single API call. It handles JavaScript rendering, proxy rotation, anti-bot mechanisms, and rate limiting so you can focus on building your application. Output formats include Markdown, structured JSON, screenshots, HTML, and more.

This document covers the complete Firecrawl API surface including the Scrape, Crawl, and Extract endpoints, along with SDK usage for Python, Node.js, and cURL.

Quick Start

Install the SDK for your preferred language and initialize with your API key:

Python

```
pip install firecrawl-py

from firecrawl import Firecrawl

app = Firecrawl(api_key="fc-YOUR-API-KEY")
result = app.scrape("https://example.com")
print(result.markdown)
```

Node.js

```
npm install @mendable/firecrawl-js

import Firecrawl from '@mendable/firecrawl-js';

const app = new Firecrawl({ apiKey: 'fc-YOUR-API-KEY' });
const result = await app.scrape('https://example.com');
console.log(result.markdown);
```

cURL

```
curl -X POST 'https://api.firecrawl.dev/v2/scrape' \
  -H 'Authorization: Bearer fc-YOUR-API-KEY' \
  -H 'Content-Type: application/json' \
  -d '{"url": "https://example.com"}'
```

Scrape Endpoint

The /v2/scrape endpoint converts a single URL into clean data. It handles JavaScript rendering, manages proxies and caching, and returns content in your preferred format.

Request Parameters

Parameter	Type	Default	Description
url	string	required	The URL to scrape
formats	string[]	["markdown"]	Output formats
onlyMainContent	boolean	true	Primary content only
includeTags	string[]	[]	CSS selectors to include
excludeTags	string[]	[]	CSS selectors to exclude
waitFor	integer	0	Wait ms before capture
timeout	integer	30000	Max page load time ms
mobile	boolean	false	Use mobile viewport

Table 1: Scrape endpoint request parameters

Response Format

```
{
  "success": true,
  "data": {
    "markdown": "# Page Title\n\nContent...",
    "html": "<html>...</html>",
    "metadata": {
      "title": "Example Page",
      "description": "A sample page",
      "language": "en",
      "sourceURL": "https://example.com",
      "statusCode": 200
    }
  }
}
```

Figure 1: Typical scrape response structure

Scrape with Structured Extraction

Use a JSON schema or Pydantic model to extract structured data from any page. This is useful for pulling product information, article metadata, or any structured content from web pages.

```
from pydantic import BaseModel
from firecrawl import Firecrawl

class Product(BaseModel):
    name: str
    price: float
    currency: str
    in_stock: bool
    description: str

app = Firecrawl(api_key="fc-YOUR-API-KEY")
result = app.scrape(
    "https://example.com/product/123",
    params={
        "formats": ["extract"],
        "extract": {
            "schema": Product.model_json_schema()
        }
    }
)
product = Product(**result.extract)
print(f"{product.name}: ${product.price}")
```

Crawl Endpoint

The `/v2/crawl` endpoint recursively discovers and scrapes every reachable subpage from a starting URL. It automatically follows links, discovers sitemaps, and handles JavaScript-rendered content across the entire site.

How Crawling Works

- Submit a crawl job with a starting URL and configuration options
- Firecrawl discovers linked pages using sitemaps and page links
- Each discovered page is scraped with your specified format options
- Results are returned via polling, WebSocket, or webhook notifications
- Large result sets are paginated with cursor-based pagination

Crawl Parameters

Parameter	Type	Default	Description
url	string	required	Starting URL for the crawl
maxDepth	integer	10	Max link depth to follow
limit	integer	10000	Max number of pages
includePaths	string[]	[]	Regex patterns to include
excludePaths	string[]	[]	Regex patterns to exclude
allowSubdomains	boolean	false	Follow subdomain links
ignoreSitemap	boolean	false	Skip sitemap.xml
deduplicateSimilar	boolean	true	Skip similar pages

Table 2: Crawl endpoint configuration parameters

Async Crawl with Polling

```
import time
from firecrawl import Firecrawl

app = Firecrawl(api_key="fc-YOUR-API-KEY")

# Start crawl job
job = app.async_crawl("https://docs.example.com")
job_id = job.id

# Poll for results
while True:
    status = app.check_crawl_status(job_id)
    print(f"Status: {status.status}, Pages: {len(status.data)}")
    if status.status == "completed":
        break
    time.sleep(5)

# Process results
for page in status.data:
    print(f"  {page.metadata.sourceURL}")
    print(f"  {len(page.markdown)} chars")
```

WebSocket Real-time Updates

```
const app = new Firecrawl({ apiKey: 'fc-YOUR-API-KEY' });

const watcher = await app.crawlUrlAndWatch(
  'https://docs.example.com',
  { limit: 100, maxDepth: 3 }
);

watcher.on('page', (page) => {
  console.log(`Scraped: ${page.metadata.sourceURL}`);
  console.log(`Content: ${page.markdown.slice(0, 200)}...`);
});

watcher.on('done', (result) => {
  console.log(`Crawl complete: ${result.data.length} pages`);
});

watcher.on('error', (err) => {
  console.error('Crawl failed:', err);
});
```

Extract Endpoint

The `/v2/extract` endpoint simplifies collecting structured data from any number of URLs or entire domains. It handles crawling, parsing, and data collation automatically.

Extraction Modes

1. Schema-based: Define a JSON schema or Pydantic model for precise output
2. Prompt-based: Describe what you want in natural language
3. Hybrid: Combine a schema with a prompt for guided extraction

Schema-based Extraction

```
from pydantic import BaseModel
from firecrawl import Firecrawl

class CompanyInfo(BaseModel):
    name: str
    mission: str
    founded_year: int
    headquarters: str
    employee_count: str
    products: list[str]

app = Firecrawl(api_key="fc-YOUR-API-KEY")
result = app.extract(
    ["https://example.com/*"],
    params={
        "schema": CompanyInfo.model_json_schema(),
        "prompt": "Extract company information"
    }
)
info = CompanyInfo(**result.data)
print(f"{info.name}, est. {info.founded_year}")
print(f"Products: {' '.join(info.products)}")
```

Pricing and Credits

Firecrawl uses a credit-based billing model. Each operation consumes credits based on the complexity of the task.

Operation	Credits	Notes
Basic scrape	1	Standard markdown scrape
JSON extraction	5	Structured extraction
Enhanced proxy	4	Premium proxy per page
PDF parsing	1/page	Per PDF page
Crawl	1/page	Per page discovered
Extract	15 tok/credit	LLM token billing
Zero retention	1/page	Enterprise compliance

Table 3: Credit usage by operation type

Advanced Features

Page Actions

Automate browser interactions before scraping using the `actions` parameter. Useful for login flows, cookie consent, or dynamic content loading.

```
result = app.scrape(
    "https://example.com/dashboard",
    params={
        "actions": [
            {"type": "click", "selector": "#cookie-accept"},
        ]
    }
)
```

```

        {"type": "wait", "milliseconds": 1000},
        {"type": "write", "selector": "#search", "text": "firecrawl"},
        {"type": "press", "key": "Enter"},
        {"type": "wait", "milliseconds": 2000},
        {"type": "screenshot"}
    ]
}
)

```

Webhook Notifications

Receive real-time notifications as pages are crawled. Webhooks include HMAC-SHA256 signatures for verification.

```

import hmac
import hashlib
from flask import Flask, request

app = Flask(__name__)
WEBHOOK_SECRET = "your-webhook-secret"

@app.route("/webhook", methods=["POST"])
def handle_webhook():
    signature = request.headers.get("X-Firecrawl-Signature")
    payload = request.get_data()
    expected = hmac.new(
        WEBHOOK_SECRET.encode(),
        payload,
        hashlib.sha256
    ).hexdigest()

    if not hmac.compare_digest(signature, expected):
        return "Invalid signature", 401

    data = request.json
    event = data["type"]
    if event == "crawl.page":
        print(f"Scraped: {data['data']['metadata']['sourceURL']}")
    elif event == "crawl.completed":
        print(f"Done: {data['data']['total']} pages")

    return "OK", 200

```

Batch Scraping

Scrape multiple URLs in a single request for improved throughput:

```

urls = [
    "https://example.com/page-1",
    "https://example.com/page-2",
    "https://example.com/page-3",
    "https://example.com/page-4",
    "https://example.com/page-5",
]

results = app.batch_scrape(urls, params={
    "formats": ["markdown", "links"],
    "onlyMainContent": True,
})

for result in results.data:
    url = result.metadata.sourceURL
    links = len(result.links)

```

```
chars = len(result.markdown)
print(f"{url}: {chars} chars, {links} links")
```

Error Handling

The API returns standard HTTP status codes. All error responses follow a consistent format:

```
{
  "success": false,
  "error": "Rate limit exceeded",
  "details": "You have exceeded the rate limit.",
  "code": "RATE_LIMIT_EXCEEDED"
}
```

Common Error Codes

Code	HTTP	Description	Resolution
RATE_LIMIT_EXCEEDED	429	Too many requests	Retry with backoff
INVALID_API_KEY	401	Invalid key	Check dashboard
INSUFFICIENT_CREDITS	402	No credits left	Purchase credits
URL_NOT_ACCESSIBLE	422	Cannot reach URL	Verify URL
TIMEOUT	408	Page timed out	Increase timeout
INTERNAL_ERROR	500	Server error	Retry after wait

Table 4: API error codes and resolutions

Retry Logic Example

```
import time
from firecrawl import Firecrawl
from firecrawl.exceptions import RateLimitError, FirecrawlError

app = Firecrawl(api_key="fc-YOUR-API-KEY")

def scrape_with_retry(url, max_retries=3):
    for attempt in range(max_retries):
        try:
            return app.scrape(url)
        except RateLimitError:
            wait = 2 ** attempt
            print(f"Rate limited, waiting {wait}s...")
            time.sleep(wait)
        except FirecrawlError as e:
            print(f"Error: {e.code} - {e.message}")
            if e.code == "INTERNAL_ERROR":
                time.sleep(1)
                continue
            raise
    raise Exception("Max retries exceeded")
```

SDK Reference

Python SDK Methods

Method	Description	Returns
scrape(url, params)	Scrape a single URL	ScrapeResult
crawl(url, params)	Crawl and wait	CrawlResult
async_crawl(url, params)	Start async crawl	CrawlJob
check_crawl_status(id)	Check job status	CrawlStatus
cancel_crawl(id)	Cancel crawl	None
batch_scrape(urls, params)	Scrape multiple URLs	BatchResult

<code>extract(urls, params)</code>	Extract structured data	<code>ExtractResult</code>
<code>map(url, params)</code>	Discover site URLs	<code>MapResult</code>

Table 5: Python SDK method reference

Rate Limits

- Free tier: 10 requests/minute, 500 credits/month
- Starter: 50 requests/minute, 3,000 credits/month
- Standard: 250 requests/minute, 100,000 credits/month
- Scale: 1,000 requests/minute, unlimited credits
- Enterprise: Custom rate limits and dedicated infrastructure

Environment Variables

```
# Required
export FIRECRAWL_API_KEY="fc-YOUR-API-KEY"

# Optional
export FIRECRAWL_BASE_URL="https://api.firecrawl.dev"
export FIRECRAWL_TIMEOUT=30000
export FIRECRAWL_RETRY_COUNT=3
```

For the latest documentation, visit docs.firecrawl.dev. For support, reach out via the Firecrawl Discord community or email support@firecrawl.dev.