Introduction to WebQL

A Modern Web Data Extraction Language

Version 1.0

Ylli Prifti
ylli@prifti.us

September 12, 2025

Abstract

WebQL (Web Query Language) is a declarative language for web data extraction that combines the simplicity of JSON5 syntax with powerful extraction capabilities. Building upon the foundations of OXPath and modern web technologies, WebQL provides a unified approach to extracting structured data from web pages, handling dynamic content, multi-modal data, and integrating AI-powered selectors. This manual provides a comprehensive introduction to WebQL, its syntax, semantics, and practical applications in modern web data extraction scenarios.

Contents

1 Introduction

The web has become the largest repository of information in human history, yet extracting structured data from web pages remains a challenging task. Traditional approaches using imperative programming with libraries like BeautifulSoup or Puppeteer require significant coding effort and domain expertise. WebQL addresses this challenge by providing a declarative language specifically designed for web data extraction.

1.1 Motivation

Modern web applications present unique challenges for data extraction:

- **Dynamic Content**: Single-page applications (SPAs) load content dynamically
- **Multi-modal Data**: Web pages contain text, images, videos, and embedded documents
- **Complex Interactions**: Data often requires user interactions (clicks, scrolls, form submissions)
- **Anti-bot Measures**: Websites implement CAPTCHAs and rate limiting
- **Maintenance Burden**: Selectors break when websites update their structure

WebQL addresses these challenges through:

- Declarative syntax that focuses on *what* to extract, not *how*
- Built-in support for browser automation and interactions
- AI-powered selectors that adapt to structural changes
- Plugin architecture for extending functionality
- Native handling of multi-modal content

1.2 Design Principles

WebQL is designed around four core principles:

1. **Simplicity**: Use familiar JSON5 syntax that's readable and writable by humans
2. **Composability**: Build complex extractions from simple, reusable components
3. **Extensibility**: Support custom processors through a plugin architecture
4. **Robustness**: Handle failures gracefully with built-in retry and error handling

1.3 Comparison with Existing Approaches

Feature	WebQL	OXPath	Scrapy	Puppeteer
Declarative	✓	✓	Partial	×
Browser Automation	✓	✓	×	✓
AI Integration	✓	×	×	×
Multi-modal	✓	×	Partial	Partial
Plugin System	✓	×	✓	×
JSON Output	Native	Extension	✓	Manual

Table 1: Comparison of WebQL with existing web extraction tools

2 Getting Started

2.1 Installation

WebQL is implemented through the DR Web Engine, which can be installed via pip:

```
pip install dr-web-engine
playwright install chromium
```

2.2 Your First Query

Let's start with a simple example that extracts quotes from a website:

```
{
  "@url": "https://quotes.toscrape.com",
  "@steps": [
    {
      "@xpath": "//div[@class='quote'][1]",
      "@fields": {
        "text": ".//span[@class='text']/text()",
        "author": ".//small[@class='author']/text()"
      }
    }
  ]
}
```

To execute this query:

```
drweb -q quotes.json5 -o results.json
```

2.3 Understanding the Structure

Every WebQL query consists of:

- `@url`: The target URL for extraction
- `@steps`: An array of extraction steps, each containing:
  - `@xpath`: XPath selector to find elements
  - `@fields`: Field mappings using relative XPath
  - Optional format directives like `@format`, `@output`

3 Language Syntax

3.1 Basic Structure

A WebQL query follows this structure:

```
{
  "@url": "https://example.com",   // Required: Target URL
  "@steps": [                      // Required: Extraction steps
    {
      "@xpath": "//div[@class='item']",  // XPath selector
      "@fields": {                       // Field mappings
        "title": ".//h2/text()",
        "description": ".//p/text()"
      }
    }
  ]
}
```

3.2 Core Directives

WebQL uses directives prefixed with `@` to define extraction behavior:

3.2.1 Basic Extraction

The fundamental extraction step uses `@xpath` and `@fields`:

```
{
  "@xpath": "//div[@class='quote']",
  "@fields": {
    "text": ".//span[@class='text']/text()",
    "author": ".//small[@class='author']/text()"
  }
}
```

3.2.2 Output Format Directives

Control output format and structure:

```
{
  "@format": "jsonl",                    // Output format
  "@output": "results.jsonl",            // Output filename
  "@compression": "minimal",             // Compression level
  "@include-metadata": true,             // Include extraction metadata
  "@streaming": true                     // Stream results
}
```

3.2.3 LLM Training Format

Format data for AI model training:

```
{
  "@format": "openai-chat",
  "@system-prompt": "You are a helpful assistant.",
  "@user-template": "Quote: {text} by {author}",
  "@field-mapping": {
    "quote_text": "text",
    "quote_author": "author"
  }
}
```

3.2.4 Navigation and Following Links

Follow links to extract data from multiple pages:

```
{
  "@follow": {
    "@xpath": ".//a[contains(@href, '/details/')]",
    "@max-depth": 2,
    "@steps": [
      {
        "@xpath": "//div[@class='detail-content']",
        "@fields": {
          "detailed_info": ".//p/text()",
          "specs": ".//ul[@class='specs']//li/text()"
        }
      }
    ]
  }
}
```

3.2.5 Nested Step Definitions

Use `@step` to define nested extraction logic:

```
{
  "@format": "jsonl",
  "@step": {
    "@xpath": "//article[@class='news-item']",
    "@fields": {
      "headline": ".//h2/text()",
      "summary": ".//p[@class='summary']/text()",
      "date": ".//time/@datetime"
    }
  }
}
```

3.3 XPath Selectors

WebQL primarily uses XPath for element selection:

- **Element Selection**: `"//div[@class='content']"`
- **Text Extraction**: `".//span/text()"`
- **Attribute Values**: `".//a/@href"`
- **Relative Paths**: Use `".//"` for relative to current context
- **Multiple Matches**: XPath naturally handles multiple matching elements

4 Advanced Features

4.1 Output Formatting for LLM Training

WebQL supports various output formats optimized for machine learning:

```
{
  "@url": "https://quotes.toscrape.com",
  "@steps": [
    {
      "@format": "openai-chat",
      "@output": "quotes_training.jsonl",
      "@system-prompt": "You are a wisdom analyzer.",
      "@user-template": "Quote: {text} by {author}",
      "@compression": "minimal",
      "@include-metadata": true,
      "@step": {
        "@xpath": "//div[@class='quote'][1]",
        "@fields": {
          "text": ".//span[@class='text']/text()",
          "author": ".//small[@class='author']/text()"
        }
      }
    }
  ]
}
```

4.2 Multi-Page Navigation

Follow links to extract data from multiple pages:

```
{
  "@url": "https://example.com/articles",
  "@steps": [
    {
      "@xpath": "//article[@class='summary']",
      "@fields": {
        "title": ".//h2/text()",
        "summary": ".//p/text()"
      },
      "@follow": {
        "@xpath": ".//a[@class='read-more']",
        "@max-depth": 2,
        "@steps": [
          {
            "@xpath": "//article[@class='full-content']",
            "@fields": {
              "full_content": ".//div[@class='content']//text()",
              "tags": ".//span[@class='tag']/text()"
            }
          }
        ]
      }
    }
  ]
}
```

4.3 Multi-modal Extraction

Extract different types of content:

```
{
  "@extract": {
    text_content: ".article-body",
    images: [{
      "@base": "img",
      url: "@attr:src",
      alt: "@attr:alt",
      ocr_text: "@ocr"  // Extract text from image
    }],
    pdf_link: {
      selector: "a[href$='.pdf']",
      url: "@attr:href",
      content: "@pdf_text"  // Extract text from PDF
    }
  }
}
```

4.4 AI-Powered Extraction

Use natural language to describe what to extract:

```
{
  "@ai_extract": {
    prompt: "Extract product specifications as key-value pairs",
    selector: ".specifications-section"
  }
}

```

```
{
  "@ai_select": {
    description: "Find the 'Add to Cart' button",
    action: "click"
  }
}
```

5 Plugin System

5.1 Built-in Plugins

WebQL includes several built-in plugins:

- **smart-retry**: Intelligent retry with exponential backoff
- **proxy-rotation**: Automatic proxy rotation
- **ai-selector**: Natural language selectors
- **api-extractor**: API response extraction
- **jsonld-extractor**: JSON-LD structured data extraction

5.2 Using Plugins in Queries

```
{
  url: "https://example.com",
  plugins: {
    "smart-retry": {
      max_attempts: 3,
      backoff: "exponential"
    }
  },
  steps: [
    {
      "@with-retry": {
        "@extract": {
          data: ".fragile-selector"
        }
      }
    }
  ]
}
```

5.3 Creating Custom Plugins

Extend WebQL with custom functionality:

```
from engine.web_engine.plugin_interface import DrWebPlugin

class CustomPlugin(DrWebPlugin):
    @property
    def metadata(self):
        return PluginMetadata(
            name="custom-extractor",
            version="1.0.0",
            description="Custom extraction logic"
```

```
        )

    def get_processors(self):
        return [CustomProcessor()]
```

6 Query Patterns

6.1 E-commerce Product Extraction

```
{
  url: "https://shop.example.com/products",
  steps: [
    {
      "@loop": {
        while: "exists(.product-card)",
        steps: [
          {
            "@extract": {
              products: [{
                "@base": ".product-card",
                name: ".product-title",
                price: ".price-now",
                original_price: ".price-was",
                rating: ".rating-value",
                reviews: ".review-count",
                image: "img@attr:src",
                link: "a@attr:href"
              }]
            }
          },
          {
            "@click": ".load-more-products",
            "@wait": 2000
          }
        ]
      }
    }
  ]
}
```

6.2 News Article Extraction

```
{
  url: "https://news.example.com",
  steps: [
    {
      "@extract": {
        articles: [{
          "@base": "article",
          headline: "h1",
          author: ".author-name",
          date: "time@attr:datetime",
          content: ".article-body",
          tags: [".tag"],
          related: [{
            "@base": ".related-article",
            title: ".title",
            url: "a@attr:href"
          }]
```

```
        }]
      }
    }
  ]
}
```

6.3 Form Interaction and Search

```
{
  url: "https://example.com/search",
  steps: [
    {
      "@type": {
        selector: "input[name='query']",
        text: "machine learning"
      }
    },
    {
      "@select": {
        selector: "select[name='category']",
        value: "research"
      }
    },
    {
      "@click": "button[type='submit']"
    },
    {
      "@wait": "selector:.search-results"
    },
    {
      "@extract": {
        results: [{
          "@base": ".result-item",
          title: ".result-title",
          url: "a@attr:href",
          snippet: ".result-snippet"
        }]
      }
    }
  ]
}
```

7 Error Handling and Debugging

7.1 Error Handling Strategies

WebQL provides multiple error handling mechanisms:

```
{
  url: "https://example.com",
  on_error: "continue",  // continue, stop, retry
  steps: [
    {
      "@try": {
        steps: [
          {"@extract": {data: ".may-not-exist"}}
        ],
        catch: [
          {"@log": "Extraction failed, using fallback"},
          {"@extract": {data: ".fallback-selector"}}
```

```
        ]
      }
    }
  ]
}
```

7.2 Debugging Queries

Enable debug mode for detailed execution logs:

```
drweb -q query.json5 -o output.json --log-level debug
```

7.3 Common Issues and Solutions

Issue	Solution
Selector returns empty	Use browser DevTools to verify selector; try AI selector as fallback
Dynamic content not loaded	Add `@wait` step or use `wait_until: "networkidle"`
Rate limiting	Enable proxy-rotation plugin or add delays between requests
CAPTCHA blocking	Use residential proxies or implement CAPTCHA solving service

Table 2: Common issues and their solutions

8 Performance Optimization

8.1 Parallel Extraction

Execute multiple extractions concurrently:

```
{
  urls: [
    "https://example.com/page1",
    "https://example.com/page2",
    "https://example.com/page3"
  ],
  parallel: 3,  // Process 3 URLs concurrently
  steps: [
    {"@extract": {/* ... */}}
  ]
}
```

8.2 Caching

Enable caching to avoid redundant requests:

```
{
  config: {
    cache: {
      enabled: true,
      ttl: 3600,  // Cache for 1 hour
      storage: "redis://localhost:6379"
    }
  }
}
```

8.3 Selective Extraction

Optimize by extracting only required fields:

```
{
  "@extract": {
    // Only extract if condition is met
    "@if": "exists(.premium-content)",
    premium_data: ".premium-content"
  }
}
```

9 Integration

9.1 Python Integration

```
from dr_web_engine import WebQL

# Initialize engine
engine = WebQL()

# Define query
query = {
    "url": "https://example.com",
    "steps": [
        {"@extract": {"title": "h1"}}
    ]
}

# Execute
results = engine.execute(query)
print(results)
```

9.2 REST API

WebQL can be exposed as a REST API:

```
# Start API server
drweb-server --port 8080

# Submit query
curl -X POST http://localhost:8080/extract \
  -H "Content-Type: application/json" \
  -d @query.json5
```

9.3 Scheduling and Automation

Integrate with schedulers for automated extraction:

```
# cron-job.yaml
schedule: "0 */6 * * *"  # Every 6 hours
query:
  url: "https://example.com/prices"
  steps:
```

```
    - "@extract":
        prices: [".price"]
output:
  type: "webhook"
  url: "https://api.example.com/prices"
```

10 Best Practices

10.1 Query Design

1. **Start Simple**: Begin with basic selectors, add complexity gradually
2. **Use Specific Selectors**: Prefer IDs and unique classes over generic tags
3. **Test Incrementally**: Verify each step before adding the next
4. **Handle Edge Cases**: Account for missing elements and pagination limits
5. **Document Queries**: Add comments to explain complex logic

10.2 Reliability

1. **Implement Retries**: Use smart-retry plugin for transient failures
2. **Fallback Selectors**: Provide alternative selectors for critical data
3. **Monitor Changes**: Set up alerts for extraction failures
4. **Version Control**: Track query changes in Git
5. **Test Suite**: Maintain test queries for regression testing

10.3 Ethics and Compliance

1. **Respect robots.txt**: Check and comply with site policies
2. **Rate Limiting**: Implement delays to avoid overwhelming servers
3. **User-Agent**: Identify your bot appropriately
4. **Terms of Service**: Review and comply with website ToS
5. **Data Privacy**: Handle extracted data according to regulations

11 Case Studies

11.1 Academic Research: Citation Network Analysis

```
{
  url: "https://scholar.google.com/scholar?q=machine+learning",
  steps: [
    {
      "@follow": {
        selector: ".gs_r",
        max_items: 100,
        extract_each: {
          title: "h3 a",
```

```
          authors: ".gs_a",
          year: ".gs_a:regex(\\d{4})",
          citations: "a:contains('Cited by')",
          pdf_link: "a[href$='.pdf']@attr:href"
        },
        follow_next: "button[aria-label='Next']"
      }
    }
  ]
}
```

11.2 Real Estate Market Analysis

```
{
  url: "https://realestate.example.com/listings",
  config: {
    viewport: {width: 1920, height: 1080}
  },
  steps: [
    {
      "@interact": {
        filters: {
          "@type": {
            selector: "#price-min",
            text: "500000"
          },
          "@type": {
            selector: "#price-max",
            text: "1000000"
          },
          "@select": {
            selector: "#property-type",
            value: "house"
          },
          "@click": "button.apply-filters"
        }
      }
    },
    {
      "@wait": 3000
    },
    {
      "@extract": {
        listings: [{
          "@base": ".listing-card",
          address: ".address",
          price: ".price",
          bedrooms: ".beds",
          bathrooms: ".baths",
          sqft: ".square-feet",
          image: "img@attr:src",
          description: ".description",
          agent: ".agent-name",
          listed_date: ".list-date"
        }]
      }
    }
  ]
}
```

12 Conclusion

WebQL represents a significant advancement in web data extraction technology, combining declarative simplicity with powerful extraction capabilities. By abstracting away the complexities of web scraping while providing fine-grained control when needed, WebQL enables both technical and non-technical users to extract structured data from the modern web efficiently.

12.1 Future Directions

The WebQL language continues to evolve with planned enhancements including:

- **Visual Query Builder**: GUI for creating queries without writing code
- **ML-based Selector Generation**: Automatic selector learning from examples
- **Distributed Execution**: Built-in support for distributed scraping
- **Schema Validation**: Type system for extracted data validation
- **Query Optimization**: Automatic query optimization for performance

12.2 Community and Resources

- **Website**: https://webql.dev
- **GitHub**: https://github.com/starlitlog/dr-web-engine
- **Documentation**: https://docs.webql.dev
- **Community Forum**: https://discuss.webql.dev

A Appendix A: Complete Language Reference

A.1 Query Structure

```
{
  // Required fields
  url: String | Array<String>,
  steps: Array<Step>,

  // Optional fields
  config: Config,
  plugins: PluginConfig,
  on_error: "stop" | "continue" | "retry",
  parallel: Number,
  cache: CacheConfig
}
```

A.2 Step Types Reference

[Complete reference of all step types with parameters...]

B Appendix B: Selector Reference

[Complete list of selector types and syntax...]

C Appendix C: Plugin API

[Plugin development guide and API reference...]