Parsing Web Request Strings

John Saysitall

Table of Contents

- The Purpose of the API
- Structure of URLs
- Functions for Parsing URLs
  - CreateRequest
  - GetProtocol
  - GetRootDomain
  - GetSubdomain
  - GetEndpoint
  - GetQueryParameters
- References

This document provides detailed information about the functions in the **GoodWeb** API. These methods are used to parse request strings to create objects that can respond to the user to send them the information they want.

The Purpose of The API

Users interact with web sites by sending requests. These are sent in the form of *Uniform Resource Locater* or *URL* strings. When they arrive at the site, the server parses that URL string and creates a request object. The request object interprets various parts of this string to determine what information is being requested, whether they exist or not, and if they do on which pages of the site they can be found in order to direct the user to those pages.

The purpose of this API is provide the functionality necessary to carry out this process.

Structure of URLs

The syntax of a request URL can be summarized in *EBNF* (Extended Backus-Naur Form) as follows:

```
URL ::= <protocol> <domain-name> [<endpoint> [<query-string>]*]*
protocol ::= ("http" | "https" | "ftp" | "news" | "nntp" | "telnet")
domain-name ::= <path-segment> ["." <path-segment>]*
endpoint ::= ["/" <path-segment>]+
path-segment := <alphanumeric-token> [("-" | "_" | ".") <alphanumeric-token>]
query-string ::= "?" <field-value-pairs>

field-value-pairs ::= <field-value-pair> ["&" <field-value-pairs>]*
field-value-pair ::= <field-name> ("=" | "<" | ">" | "=<" | "=>") <field-value>
field-name := <alphanumeric-token>
field-value ::= (<alphanumeric-string> | <numeric-token>)

alphanumeric-string ::= <alphanumeric-token> ["+" <alphanumeric-token>]*
alphanumeric-token ::= [A-Za-z][A-Za-z0-9]*
numeric-token ::= [0-9][0-9]*
```

While being sent over the internet, the URL may incorporate a number of components:

1. The protocol used to access the site

2. The root domain
3. The subdomain
4. The path to the endpoint
5. The query string

The URL need not include all of these components except for the first two, but if it does, then they must be in the above order. The table below briefly describes the main components:

Component	Description
Protocol	This is the name of the set of rules the user's browser and the site's server have to observe to be able to communicate with each other. This will be of the form *http*, *https*, *ftp*, etc.
Root Domain	This is the domain through which all subdomains and endpoints of the site are accessed. For example, in the URL `https://www.goodfood.com` the root domain is *goodfood.com*. This is also used by the browser to look up the IP address of the site from a DNS server.
Subdomains	Subdomains are domains which can only be accessed via the root domain. For example, in the URLs `https://diabetic.goodfood.com` and `https://dietary.goodfood.com`, the bits *diabetic* and *dietary* are subdomains. The server of the site accessed uses this to find the local IP address to be used to serve pages.
Endpoints	These are the paths of the site used to receive requests and respond to them. It is the last portion of the URL before the query string, if any, and is part of the site that responds to a specific type of request. For example, in the URL `https://www.goodfood.com/vegetarian/products`, *products* is the endpoint.
Query String	This contains the parameters sent to the site to return specific information. The query starts at the question mark ("?"), the parameters are sent as key-value pairs—with the parameter name followed by a relational operator ("=", ">", "<", ">=", "<=;") followed by a value—and are separated by ampersands ("&"). In the URL `https://www.goodfood.com/vegetarian/products?brand=elite&price=15&home-delivery=true`, the query string is `?brand=elite&price=15&home-delivery=true`.

Functions for Parsing URLs

The library provides the functions documented below for parsing URLs, but before calling any of these methods, the URL has to be passed to the server's " CreateRequest" method.

CreateRequest

The first responder for the URL. This method does the preliminary work of parsing and validating the URL.

WebServer.CreateRequest(url: String): WebServer.Request

If the URL is empty no request object is created, and if it is malformed the server raises an exception. Otherwise, a request object is returned from which the other methods can be called to return the parts indicated by the method names.

Typical use:

```
try {
        server.CreateRequest(url)
        var request = server.CreateRequest()
        if request == null {
                // URL is empty: exit process
                return
        }
        // proceed to other actions
        ...
}
catch exception: MalformedURLException {
        server.DisplayErrorPage(exception.Message)
}
```

GetProtocol

Returns the protocol used to access the site. This determines whether the access is secure or whether the user wants to view pages on the site or download files, etc. The protocol cannot be empty, and if it is not supported an exception is raised.

WebServer.Request.GetProtocol(): String

The table below indicates the protocols that can be parsed from a URL:

URL	Protocol	Use
`http://www.goodfood.com`	*http*	Hypertext is served
`https://www.goodfood.com`	*https*	Hypertext is served over a secure connection
`ftp://www.goodfood.com`	*ftp*	Files available to the public can be downloaded
`news://www.goodfood.com`	*news*	The news feed is provided
`file://www.goodfood.com`	N/A	Undefined: `file` is not a valid Web protocol
`www.goodfood.com`	N/A	Protocol empty: defaults to `https`

Typical use:

```
try {
        // create request, etc.
        ...
        var protocol = request.GetProtocol()
        if protocol == "" {
                protocol = "https"
        }
        request.CommunicateOver(protocol)
        // proceed to other actions
        ...
}
catch exception: ProtocolNotSupportedException {
        server.DisplayErrorPage(exception.Message)
}
```

GetRootDomain

Returns the root domain of the URL. This can be used to check whether the user is trying to access the landing page of the site.

WebServer.Request.GetRootDomain(): String

The table below indicates what domain names can be returned from a URL:

URL	Root Domain
`https://www.goodfood.com`	*goodfood.com*
`https://goodfood.com`	*goodfood.com*
`https://www.com`	Undefined: malformed URL
`https://goodfood`	Undefined: malformed URL

Typical use:

```
try {
        // create request, etc.
        ...
        var root = request.GetRootDomain()
        request.SetRootTo(root)
        // proceed to other actions
        ...
}
catch MalformedURLException {
        // etc.
}
```

GetSubdomain

Returns the subdomain in the URL. If the subdomain is part of the site, the user can then be redirected to it.

WebServer.Request.GetSubdomain(): String

The table below indicates what subdomains can be returned from a URL:

URL	Subdomain
`https://www.diabetic.goodfood.com`	*diabetic*
`https://www.dietary.goodfood.com`	*dietary*
`https://goodfood.com`	Defaults to the root domain (i.e *goodfood.com*)

Typical use:

```
// create request, etc.
...
var subdomain = request.GetSubdomain()
if subdomain != root {
        server.RedirectTo(subdomain)
}
// ... etc.
```

GetEndpoint

Returns the endpoint the URL addresses. Can be used to verify that the endpoint exists. In order for the site to respond to the request, the query string must be routed to the correct endpoint.

WebServer.Request.GetEndPoint(): String

The table below indicates what domain names can be returned from various URLs:

URL	Endpoint
`https://www.goodfood.com`	None specified: defaults to the root domain (i.e *goodfood.com*)
`https://www.goodfood.com/products`	*products*
`https://www.goodfood.com/recipes/vegetarian`	*vegetarian*

Typical use:

```
// create request, etc.
...
var endpoint = request.GetEndpoint()
if endpoint != "" and server.Endpoints.Contains(endpoint) {
        server.RedirectTo(subdomain, endpoint)
        // ... etc.
}
```

GetQueryParameters

Returns the query string at the end of the URL as a dictionary. The names in the query string are used as keys and the actual parameters are used as values. Before the creation of the dictionary, the parameter names are checked to verify that the endpoint can respond to them.

If there are parameters that the endpoint cannot recognize (for example, due to a misspelling), then an error is raised. The error contains information about the malformed or non-existent parameter(s).

If the dictionary is created, the server can then proceed to display the results from the site database.

WebServer.Request.GetQueryParameters(): Dictionary

The table below indicates the parameters and the values contained in the created dictionary:

URL	Query String	Parameter Names	Parameter Values
`https://www.goodfood.com/vegetarian/products?brand=elite&price=15&home-delivery=true`	`brand=elite&price=15&home-delivery=true`	*brand*, *price*, *home-delivery*	*elite*, *15*, *true*
`https://www.goodfood.com/products?brand=elist`	`brand=elist`	*brand*	Unknown value: the brand *elist* does not exist. Raises an exception
`https://www.goodfood.com/products`	N/A	N/A	Undefined: empty

			query string
`https://www.goodfood.com`	N/A	N/A	Undefined: empty query string

Typical use:

```
try {
        // create request, etc.
        ...
        var parameters = request.GetQueryParameters()
        if parameters != null {
                // create database instance
                var database = Database.Instance()
                // proceed to displaying the results
                var dataset = database.GetDatasetFor(parameters)
                server.DisplayPageFor(dataset)
                // other actions
                ...
        }
}
catch exception: ParameterNotFoundException {
        server.DisplayErrorPage(exception.Message)
        // etc.
        ...
}
```

References

- The **GoodWeb** API Handbook, pp. 18–22
- The GoodWeb Server API discussion thread
- The **GoodWeb** message board on Pro-Coders