# URLs and Resources

## Navigating the Internet's Resources

**URL** is standardized names of internet resources , points to something telling its location and how to interact with .

**URL** considered the humans access point to HTTP and other **protocols** , you just write **URL** in the browser and behind the scene it's know how to deal with it and give the user what he need .

**URL** is a subset of more general concept of **resource identifier** called uniform resource identifier **URI**
URI is split into URL and URN
URL -> identify the resource by its location (most common)
URN -> identify by its name despite its location

**URL** anatomy :
`http://www.joes-hardware.com/seasonal/index-fall.html`

`http://` -> scheme (how)
`www.joes-hardware.com` -> Host (where)
`seasonal/index-fall.html` -> path (what)

**URL** work with many protocols :
`ftp://ftp.lots-o-books.com/pub/complete-price-list.xls`

`rtsp://www.joes-hardware.com:554/interview/cto_video`

`http://www.joes-hardware.com/seasonal/index-fall.html`

## URL Syntax

**URL** syntax varies depends on which scheme used , but not that much.

Most URL schemes base their URL syntax on this nine-part general format:
`<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>`

**scheme** : Which protocol to use when accessing a server to get a resource.

**user**: The username some schemes require to access a resource.

**password** :The password that may be included after the username, separated by a colon (:).

**host**: The hostname or dotted IP address of the server hosting the resource.

**port**: The port number on which the server hosting the resource is listening. Many schemes have default port numbers (the default port number for HTTP is 80).

**path**: The local name for the resource on the server, separated from the previous URL components by a slash (/).

**params**: Used by some schemes to specify input parameters. Params are name/value pairs. A URL can contain multiple params fields, separated from themselves and the rest of the path by semicolons (;).

**query**: Used by some schemes to pass parameters to active applications (such as databases, bulletin boards, search engines, and other Internet gateways). There is no common format for the contents of the query component. It is separated from the rest of the URL by the "?" character.

**fragments**: A name for a piece or part of the resource. The frag field is not passed to the server when referencing the object; it is used internally by the client. It is separated from the rest of the URL by the "#" character.

## URL shortcuts

there are a flavors of **URLs** : relative and absolute

**Absolute** : full URL which you have all information you need to access resources

**relative** (incomplete) : and to get what you need to access resources , you must interpret it relative to another URL called **base**

**relative** are only a piece from the URL , and applications that process URLs need to be able to convert between **absolute and relative**

> the key advantage of using **relative** is to make it **portable**

## Base URL ways

- explicitly provide in source `<BASE>` tag
- base URL of encapsulated resource

## resolving relative reference

first resolving any relative URL must have a BASE URL it relative to
e.g. `http://example.com/dir1/dir2/page.html`

the algorithm of resolve relative is by examine and inherit missing components from the base URL . if relative provided it , it'll be merge with the base

### examples

1. scheme present in the relative
   `https://another.com/path/file.html` --> URL is already absolute , no change needed
2. all component is empty in the relative
   `""` -> inherit everything from the **base**
3. path is empty but query is present
   `"?id=123"` --> inherit everything from base but replace the query
   `http://example.com/dir1/dir2/page.html?id=123`
4. relative start with (/)
   `/image/logo.png` --> replaces the base path, keeping only the scheme and host
   `https://example.com/images/logo.png`
5. relative path without leading slash or with `./`
   `file.html` and `./file.html` --> the relative path is resolved within the base
   `http://example.com/dir1/dir2/file.html`
6. relative path with .. parent
   `../newpage.html` --> move one up before append the new path
   `https://example.com/dir1/newpage.html`
7. if the query or param is in base and not in relative it inherit too, if no it will overridden

## Shady Characters

URL designed to be :

- portable across protocols
- readable by humans
- complete
  ensuring the safe transmit and this need some restrictions and encoding mechanism to handle unsafe characters that would affect any of these points.

### why some characters restricted in URLs ?

1. it must safe transmit across different protocols as there are some protocols strip off certain characters which cause data loss e.g. SMTP
2. human readability so invisible or non-printable characters are prohibited
3. complete as there are some non-ASCII characters like in French or Spanish

all that solve is by using a **percent-encoding** `%XX` it's for unsafe characters and the special characters that used primary in URL if you need to append it as it is you need to use encoding

## Character category

1. safe used without encoding
   `A-Z a-z 0-9 - _ . ~`
2. reserved has a special meaning in URLs and may require encoding
   `: / ? # [ ] @ ! $ & ' ( ) * + , ; =`
3. unsafe must always have encoding
   - Are **not printable ASCII** (e.g., `é`, `ü`, `€`)
   - Are **whitespace characters** (space, tab, newline)
   - Can **break URL processing** in certain systems (`"`, `<`, `>`, `\`, `{`, `|`, `}`)