Class java.net.URL

```
java.lang.Object
  |
  +--java.net.URL
```

public final class **URL**
extends Object
implements Serializable, Comparable

Class `URL` represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web. A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine. More information on the types of URLs and their formats can be found at:

```
http://www.ncsa.uiuc.edu/demoweb/url-primer.html
```

In general, a URL can be broken into several parts. The previous example of a URL indicates that the protocol to use is `http` (HyperText Transport Protocol) and that the information resides on a host machine named `www.ncsa.uiuc.edu`. The information on that host machine is named `demoweb/url-primer.html`. The exact meaning of this name on the host machine is both protocol dependent and host dependent. The information normally resides in a file, but it could be generated on the fly. This component of the URL is called the *file* component, even though the information is not necessarily in a file.

A URL can optionally specify a "port", which is the port number to which the TCP connection is made on the remote host machine. If the port is not specified, the default port for the protocol is used instead. For example, the default port for `http` is `80`. An alternative port could be specified as:

```
http://www.ncsa.uiuc.edu:8080/demoweb/url-primer.html
```

A URL may have appended to it an "anchor", also known as a "ref" or a "reference". The anchor is indicated by the sharp sign character "#" followed by more characters. For example,

```
http://java.sun.com/index.html#chapter1
```

This anchor is not technically part of the URL. Rather, it indicates that after the specified resource is retrieved, the application is specifically interested in that part of the document that has the tag `chapter1` attached to it. The meaning of a tag is resource specific.

An application can also specify a "relative URL", which contains only enough information to reach the resource relative to another URL. Relative URLs are frequently used within HTML pages. For example, if the contents of the URL:

```
http://java.sun.com/index.html
```

contained within it the relative URL:

```
FAQ.html
```

it would be a shorthand for:

```
http://java.sun.com/FAQ.html
```

The relative URL need not specify all the components of a URL. If the protocol, host name, or port number is missing, the value is inherited from the fully specified URL. The file component must be specified. The optional anchor is not inherited.

Since:
JDK1.0
See Also:
Serialized Form

Constructor Summary

URL(String protocol, String host, int port, String file) Creates a `URL` object from the specified `protocol`, `host`, `port` number, and `file`.
URL(String protocol, String host, String file) Creates an absolute URL from the specified `protocol` name, `host` name, and `file` name.
URL(String spec) Creates a `URL` object from the `String` representation.
URL(URL context, String spec) Creates a URL by parsing the specification `spec` within a specified context.

Method Summary

int	**compareTo**(Object url) Compares a URL to another URL. The comparison is done on a canonical string representation of the URL, which includes the URL's protocol, ref, host, port and file.
boolean	**equals**(Object obj) Compares two URLs.
Object	**getContent**() Returns the contents of this URL. This method is a shorthand for: openConnection().getContent()
String	**getFile**() Returns the file name of this `URL`.
String	**getHost**() Returns the host name of this `URL`, if applicable.
int	**getPort**() Returns the port number of this `URL`.
String	**getProtocol**() Returns the protocol name this `URL`.
String	**getRef**() Returns the anchor (also known as the "reference") of this `URL`.
int	**hashCode**() Creates an integer suitable for hash table indexing.
URLConnection	**openConnection**() Returns a `URLConnection` object that represents a connection to the remote object referred to by the `URL`.

InputStream	**openStream**() Opens a connection to this `URL` and returns an `InputStream` for reading from that connection.
boolean	**sameFile**(URL other) Compares two URLs, excluding the "ref" fields.
protected void	**set**(String protocol, String host, int port, String file, String ref) Sets the fields of the URL. This is not a public method so that only URLStreamHandlers can modify URL fields.
static void	**setURLStreamHandlerFactory**(URLStreamHandlerFactory fac) Sets an application's `URLStreamHandlerFactory`.
String	**toExternalForm**() Constructs a string representation of this `URL`.
String	**toString**() Constructs a string representation of this `URL`.

Methods inherited from class java.lang.Object
clone , finalize , getClass , notify , notifyAll , wait , wait , wait

Constructor Detail

URL

```
public URL(String protocol,
           String host,
           int port,
           String file)
    throws MalformedURLException
```

Creates a `URL` object from the specified `protocol`, `host`, `port` number, and `file`. Specifying a `port` number of `-1` indicates that the URL should use the default port for the protocol.

If this is the first URL object being created with the specified protocol, a *stream protocol handler* object, an instance of class `URLStreamHandler`, is created for that protocol:

1. If the application has previously set up an instance of `URLStreamHandlerFactory` as the stream handler factory, then the `createURLStreamHandler` method of that instance is called with the protocol string as an argument to create the stream protocol handler.
2. If no `URLStreamHandlerFactory` has yet been set up, or if the factory's `createURLStreamHandler` method returns `null`, then the constructor finds the value of the system property:

   ```
   java.protocol.handler.pkgs
   ```

   If the value of that system property is not `null`, it is interpreted as a list of packages separated by a vertical slash character '`|`'. The constructor tries to load the class named:

   ```
   <package>.<protocol>.Handler
   ```

   where *<package>* is replaced by the name of the package and *<protocol>* is replaced by the name of the protocol. If this class does not exist, or if the class exists but it is not a subclass of `URLStreamHandler`, then the next package in the list is tried.
3. If the previous step fails to find a protocol handler, then the constructor tries to load the class named:

If this class does not exist, or if the class exists but it is not a subclass of `URLStreamHandler`, then a `MalformedURLException` is thrown.

Parameters:

`protocol` - the name of the protocol.
`host` - the name of the host.
`port` - the port number.
`file` - the host file.

Throws:

MalformedURLException - if an unknown protocol is specified.

See Also:

`System.getProperty(java.lang.String)`, `setURLStreamHandlerFactory(java.net.URLStreamHandlerFactory)`, `URLStreamHandler`, `URLStreamHandlerFactory.createURLStreamHandler(java.lang.String)`

URL

```
public URL(String protocol,
           String host,
           String file)
    throws MalformedURLException
```

Creates an absolute URL from the specified `protocol` name, `host` name, and `file` name. The default port for the specified protocol is used.

This method is equivalent to calling the four-argument constructor with the arguments being `protocol`, `host`, `-1`, and `file`.

Parameters:

`protocol` - the protocol to use.
`host` - the host to connect to.
`file` - the file on that host.

Throws:

MalformedURLException - if an unknown protocol is specified.

See Also:

`URL(java.lang.String, java.lang.String, int, java.lang.String)`

URL

```
public URL(String spec)
    throws MalformedURLException
```

Creates a `URL` object from the `String` representation.

This constructor is equivalent to a call to the two-argument constructor with a `null` first argument.

Parameters:

`spec` - the `String` to parse as a URL.

Throws:

MalformedURLException - If the string specifies an unknown protocol.

See Also:

`URL(java.net.URL, java.lang.String)`

URL

```
public URL(URL context,
           String spec)
    throws MalformedURLException
```

Creates a URL by parsing the specification `spec` within a specified context. If the `context` argument is not `null` and the `spec` argument is a partial URL specification, then any of the strings missing components are inherited from the `context` argument.

The specification given by the `String` argument is parsed to determine if it specifies a protocol. If the `String` contains an ASCII colon ':' character before the first occurrence of an ASCII slash character '/', then the characters before the colon comprise the protocol.

- If the `spec` argument does not specify a protocol:
  - If the context argument is not `null`, then the protocol is copied from the context argument.
  - If the context argument is `null`, then a `MalformedURLException` is thrown.
- If the `spec` argument does specify a protocol:
  - If the context argument is `null`, or specifies a different protocol than the specification argument, the context argument is ignored.
  - If the context argument is not `null` and specifies the same protocol as the specification, the `host`, `port` number, and `file` are copied from the context argument into the newly created `URL`.

The constructor then searches for an appropriate stream protocol handler of type `URLStreamHandler` as outlined for:

```
java.net.URL#URL(java.lang.String, java.lang.String, int,
                 java.lang.String)
```

The stream protocol handler's `parseURL` method is called to parse the remaining fields of the specification that override any defaults set by the context argument.

Parameters:

`context` - the context in which to parse the specification.
`spec` - a `String` representation of a URL.

Throws:

MalformedURLException - if no protocol is specified, or an unknown protocol is found.

See Also:

`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandler`, `URLStreamHandler.parseURL(java.net.URL, java.lang.String, int, int)`

Method Detail

set

```
protected void set(String protocol,
                   String host,
                   int port,
                   String file,
                   String ref)
```

Sets the fields of the URL. This is not a public method so that only URLStreamHandlers can modify URL fields. URLs are otherwise constant. REMIND: this method will be moved to URLStreamHandler

Parameters:

`protocol` - the protocol to use

`host` - the host name to connecto to
`port` - the protocol port to connect to
`file` - the specified file name on that host
`ref` - the reference

---

getPort

```
public int getPort()
```

Returns the port number of this `URL`. Returns -1 if the port is not set.

Returns:
the port number

---

getProtocol

```
public String getProtocol()
```

Returns the protocol name this `URL`.

Returns:
the protocol of this `URL`.

---

getHost

```
public String getHost()
```

Returns the host name of this `URL`, if applicable. For "`file`" protocol, this is an empty string.

Returns:
the host name of this `URL`.

---

getFile

```
public String getFile()
```

Returns the file name of this `URL`.

Returns:
the file name of this `URL`.

---

getRef

```
public String getRef()
```

Returns the anchor (also known as the "reference") of this `URL`.

Returns:
the anchor (also known as the "reference") of this `URL`.

---

equals

`public boolean equals(Object obj)`

Compares two URLs. The result is `true` if and only if the argument is not `null` and is a `URL` object that represents the same `URL` as this object. Two URL objects are equal if they have the same protocol and reference the same host, the same port number on the host, and the same file and anchor on the host.

Parameters:
`obj` - the URL to compare against.
Returns:
`true` if the objects are the same; `false` otherwise.
Overrides:
equals in class Object

---

hashCode

`public int hashCode()`

Creates an integer suitable for hash table indexing.

Returns:
a hash code for this `URL`.
Overrides:
hashCode in class Object

---

compareTo

`public int compareTo(Object url)`

Compares a URL to another URL. The comparison is done on a canonical string representation of the URL, which includes the URL's protocol, ref, host, port and file.

Specified by:
compareTo in interface Comparable
Parameters:
`url` - the url to compare this to.
Since:
JDK1.2
See Also:
`getRef()`, `Comparable`, `String.compareTo(java.lang.String)`

---

sameFile

`public boolean sameFile(URL other)`

Compares two URLs, excluding the "ref" fields. Returns `true` if this `URL` and the `other` argument both refer to the same resource. The two `URL`s might not both contain the same anchor.

Parameters:
`other` - the `URL` to compare against.
Returns:
`true` if they reference the same remote object; `false` otherwise.

---

toString

```
public String toString()
```

Constructs a string representation of this `URL`. The string is created by calling the `toExternalForm` method of the stream protocol handler for this object.

Returns:
a string representation of this object.
Overrides:
toString in class Object
See Also:
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandler.toExternalForm(java.net.URL)`

---

toExternalForm

```
public String toExternalForm()
```

Constructs a string representation of this `URL`. The string is created by calling the `toExternalForm` method of the stream protocol handler for this object.

Returns:
a string representation of this object.
See Also:
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandler.toExternalForm(java.net.URL)`

---

openConnection

```
public URLConnection openConnection()
                           throws IOException
```

Returns a `URLConnection` object that represents a connection to the remote object referred to by the `URL`.

A new connection is opened every time by calling the `openConnection` method of the protocol handler for this URL.

If for the URL's protocol (such as HTTP or JAR), there exists a public, specialized URLConnection subclass belonging to one of the following packages or one of their subpackages: java.lang, java.io, java.util, java.net, the connection returned will be of that subclass. For example, for HTTP an HttpURLConnection will be returned, and for JAR a JarURLConnection will be returned.

Returns:
a `URLConnection` to the URL.
Throws:
IOException - if an I/O exception occurs.
See Also:
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLConnection`, `URLStreamHandler.openConnection(java.net.URL)`

---

openStream

```
public final InputStream openStream()
                           throws IOException
```

Opens a connection to this `URL` and returns an `InputStream` for reading from that connection. This method is a shorthand for:

```
openConnection().getInputStream()
```

Returns:
an input stream for reading from the URL connection.
Throws:
IOException - if an I/O exception occurs.
See Also:
openConnection(), URLConnection.getInputStream()

---

getContent

```
public final Object getContent()
                     throws IOException
```

Returns the contents of this URL. This method is a shorthand for:

```
openConnection().getContent()
```

Returns:
the contents of this URL.
Throws:
IOException - if an I/O exception occurs.
See Also:
URLConnection.getContent()

---

setURLStreamHandlerFactory

```
public static void setURLStreamHandlerFactory(URLStreamHandlerFactory fac)
```

Sets an application's `URLStreamHandlerFactory`. This method can be called at most once in a given Java Virtual Machine.

The `URLStreamHandlerFactory` instance is used to construct a stream protocol handler from a protocol name.

Parameters:
`fac` - the desired factory.
Throws:
Error - if the application has already set a factory.
See Also:
URL(java.lang.String, java.lang.String, int, java.lang.String), URLStreamHandlerFactory

---