



Contents

- [1 HTTP \(Hyper Text Transfer Protocol\)](#)
- [2 HTTPS \(Hyper Text Transfer Protocol - Secure\)](#)
- [3 Reading and writing data using HTTP](#)
- [4 Reading and writing data using HttpsConnection](#)
- [5 Sample code](#)

HTTP (Hyper Text Transfer Protocol)

HTTP is a request response protocol. It means that for every request send by the client to the server, the sever sends back a response to the client. HttpURLConnection interface of Generic Connection Framework provides the necessary methods and constants for the HTTP Connection in MIDP 2.0.

HttpConnection provides three methods ? GET, POST and HEAD - for data transfer.

GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. In GET data to be send to the server, can be send only by adding the parameters (we don?t support persistent connections) at the end of the server URL in the encoded form. For example the URL

```
http://www.xyz.com?h1=en&q=http
```

denotes the parameters passed to the server "h1=en" and "q=http" in the URL as name and value pairs. Since data is passed as part of URL, the maximum possible length of URL on a device limits the size of data to be send to the server.

POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI . In POST, data to be send to the server are sent as part of the message body. Since data is send to the server as a message body there is no limitation on the size of data to be sent to the server.

HEAD is same as GET, except that the server does not return the message body in the response, only headers are returned.

HTTPS (Hyper Text Transfer Protocol - Secure)

HTTP operates on the TCP/IP, it is not safe to send confidential messages. The messages can be read in between. HTTPS provides a way to overcome it. HTTPS, run over the Transport Layer Security (TLS), Secure Sockets Layer (SSL) or similar protocols. Our phones support the HTTPS over SSL and TLS v1.0.

MIDP2.0 supports HTTPS using the HttpsConnection interface of the Generic Connection Framework.

Since HttpsConnection extends HttpURLConnection, it provides support for all the methods?GET, POST and HEAD that are available with HttpURLConnection.

Reading and writing data using HTTP

1. **Get the HttpURLConnection Object** Connector class of javax.microedition.io package provides three ways to obtain the HttpURLConnection Object:

```
HttpURLConnection httpConn = (HttpURLConnection)Connector.open(String url);  
HttpURLConnection httpConn = (HttpURLConnection)Connector.open(String url, int mode);  
HttpURLConnection httpConn = (HttpURLConnection)Connector.open(String url, int mode, boolean timeout);
```

In the above methods:

- url is the URL of the server. For HttpURLConnection it will be of the format:
<http://www.wiki.forum.nokia.com>.
- mode specifies one of the READ, READ_WRITE or WRITE modes.

If no mode is specified, the default mode is READ_WRITE, i.e. both read and write operations can be performed.

- timeout ? a flag to indicate if the caller wants to see the exceptions on time out. If this flag is true and if connection cannot be made after 60 seconds, the open() will throw IOException. If no flag is specified, the default value is true.

2. **Reading from the server** ? The HttpURLConnection object so obtained can be used to read data from the server by obtaining the input stream or data input stream. The input stream can be obtained by using the openInputStream() of the obtained HttpURLConnection object, whereas the data input stream can be obtained by using the openDataInputStream() of HttpURLConnection object as shown below:

```
InputStream is = httpConn.openInputStream();  
DataInputStream dis = httpConn.openDataInputStream();
```

Data can be read using the read method of the InputStream and DataInputStream class. It should be noted that if no data can be read from the server within 40 seconds, the read method will throw IOException.

3. **Writing to server** ? In order to write or post some data to a server, an output stream or data output stream is required. The request method?POST should be specified by using setRequestMethod() of HttpURLConnection, before writing data to server. It is also recommended to send the Content-Length header in the request with the exact length of data to be read. After setting the headers, the output stream can be obtained by using the openOutputStream method of the HttpURLConnection, whereas the data output stream can be obtained by using the openDataOutputStream() of HttpURLConnection as shown below:

```
OutputStream os = httpConn.openOutputStream();  
DataOutputStream dos = httpConn.openDataOutputStream();
```

Data can then be written using the write method of the OutputStream or DataOutputStream. It should be noted that if no data can be written to the server within 60 seconds, the read method will throw IOException.

Reading and writing data using `HttpsConnection`

Reading and writing data using `HttpsConnection` is very similar to reading data over the `HttpConnection` (as discussed previously in steps 1 to 3), except for the way the `HttpsConnection` object is obtained. The `HttpsConnection` object can be obtained by replacing the `HttpConnection` in step 1 of previous section with the `HttpsConnection`. So the `HttpsConnection` object can be obtained from one of the following open methods of `Connector` class:

```
HttpsConnection httpsConn = (HttpsConnection)Connector.open(String url);
HttpsConnection httpsConn = (HttpsConnection)Connector.open(String url, int mode);
HttpsConnection httpsConn = (HttpsConnection)Connector.open(String url, int mode, boolean timeout);
```

Sample code

```

/*****
 *This method retrieves the data from the server using HTTP GET.
 *****/
String getDataFromServer(){
    HttpConnection httpConn = null;
    InputStream is = null;
    String dataRead = "";
    try{
        httpConn = (HttpConnection)Connector.open(serverUrl);
        if((httpConn.getResponseCode() == HttpConnection.HTTP_OK)){
            int length = (int)httpConn.getLength();
            is = httpConn.openInputStream();
            if(length == -1){//unknown length returned by server.
//It is more efficient to read the data in chunks, so we
//will be reading in chunk of 1500 = Maximum MTU possible

                int chunkSize = 1500;
                byte[] data = new byte[chunkSize];
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                int dataSizeRead = 0;//size of data read from input stream.
                while((dataSizeRead = is.read(data))!= -1){
//it is not recommended to write to string in the
//loop as it causes heap defragmentation and it is
//inefficient, therefore we use the
//ByteArrayOutputStream.
                    baos.write(data, 0, dataSizeRead );
                    System.out.println("Data Size Read = "+dataSizeRead);
                }
                dataRead = new String(baos.toByteArray());
                baos.close();
            } else{//known length
                DataInputStream dis = new DataInputStream(is);
                byte[] data = new byte[length];
//try to read all the bytes returned from the server.
                dis.readFully(data);
                dataRead = new String(data);
            }
            resultForm.deleteAll();
            resultForm.append("Data Read from server--\n"+dataRead);
            System.out.println("Data Read from server--\n"+dataRead);
        } else{
            resultForm.append("\nServer returned unhandled " +
                "response code. "+httpConn.getResponseCode());
        }
    }
}

```

Using_Http_and_Https_in_Java_ME

```
    }
    display.setCurrent(resultForm);
} catch(Throwable t){
    System.out.println("Exception occurred during GET "+t.toString());
    resultForm.deleteAll();
    resultForm.append("\nNetwork failure in retrieving data from " +
        "server. Try Again!");
    display.setCurrent(resultForm);
}
//Since only limited number of network objects can be in open state
//it is necessary to clean them up as soon as we are done with them.
finally{//Networking done. Clean up the network objects
    try{
        if(is != null)
            is.close();
    } catch(Throwable t){
        System.out.println("Exception occurred while closing input " +
            "stream.");
    }
    try{
        if(httpConn != null)
            httpConn.close();
    } catch(Throwable t){
        System.out.println("Exception occurred "+t.toString());
    }
}
return dataRead;
}
/*****
 *This method posts the data entered by user in the text field to the server.
 *****/
void postDataToServer(String data){
    OutputStream os = null;
    InputStream is = null;
    HttpURLConnection httpConn = null;
    try{
        httpConn = (HttpURLConnection)Connector.open(serverUrl);
        httpConn.setRequestMethod(HttpURLConnection.POST);
        String dataToBeSend = data;
        httpConn.setRequestProperty("Content-Length",
            ""+dataToBeSend.length());
        os = httpConn.getOutputStream();
        os.write(dataToBeSend.getBytes());
        os.flush();//data written will be flushed to server.
        resultForm.deleteAll();
        resultForm.append("\nResponse Code returned during POST is " +
            ""+httpConn.getResponseCode());
        display.setCurrent(resultForm);
    } catch(Throwable t){
        System.out.println("Exception occured "+t.toString());
        resultForm.deleteAll();
        resultForm.append("\nNetwork failure in retrieving data from " +
            "server. Try Again!");
        display.setCurrent(resultForm);
    }
}
//Since only limited number of network objects can be in open state
//it is necessary to clean them up as soon as we are done with them.
finally{
    try{
        if(os != null)
            os.close();
    } catch(Throwable t){
```

Using_Http_and_Https_in_Java_ME

```
        System.out.println("Exception occured "+t.toString());
    }
    try{
        if(httpConn != null)
            httpConn.close();
    } catch(Throwable t){
        System.out.println("Exception occured "+t.toString());
    }
}
}
```