Web Server

User's Manual

Reference: TLT-0640-MAN-HOKA
Product Version: 1.0
Revision: A

Confidentiality & Intellectual Property

Table of Contents

Index of Tables

Illustration Index

1 Bibliography

[EDC] Embedded Device Configuration Specification 1.2
http://www.e-s-r.net/javadocs/edc-1.2-api/index.html

[CLDC] Connected Limited Device Configuration Specification 1.1
http://docs.oracle.com/javame/config/cldc/ref-impl/cldc1.1/cldc11api.pdf

[BON] Beyond Profile Specification 1.2
http://e-s-r.net/en/ESR/ESR-SPE-0001-BON-1.2-D.pdf

[MICROUI] Micro User Interface Profile Specification 1.4
http://e-s-r.net/en/ESR/ESR-SPE-0002-MICROUI-1.4-I.pdf

[MWT] Micro Widget Toolkit Profile Specification 1.0
http://e-s-r.net/en/ESR/ESR-SPE-0011-MWT-1.0-D.pdf

[RFC2616] Hypertext Transfer Protocol – HTTP/1.1
http://tools.ietf.org/pdf/rfc2616.pdf

[RFC2046] MIME Media Types
http://tools.ietf.org/pdf/rfc2046.pdf

[WCTE] Wikipedia: Chunked Transfer Encoding
http://en.wikipedia.org/wiki/Chunked_transfer_encoding

2 Introduction

2.1 Scope

This document explains how the HOKA HTTP Server can be facilitated to create web interfaces or M2M capabilities for embedded applications.

2.2 Intended audience

The intended audience for this document are *Java developers* who are familiar with Socket communication, the HTTP 1.1 protocol and web server concepts.

2.3 Document Organisation

The document is divided into the following parts:

- Introduction to Networking with Client and Server Sockets. (see: *3 Basic Networking Concepts*)
- The HOKA-1.0 Web Server. (see: *4 HOKA Web Server*)
- API documentation. (see: *5 API Documentation*)

2.4 Libraries Overview

In the following diagram the *libraries* reviewed in this user manual are displayed. The arrows mark the *dependencies* between the libraries.

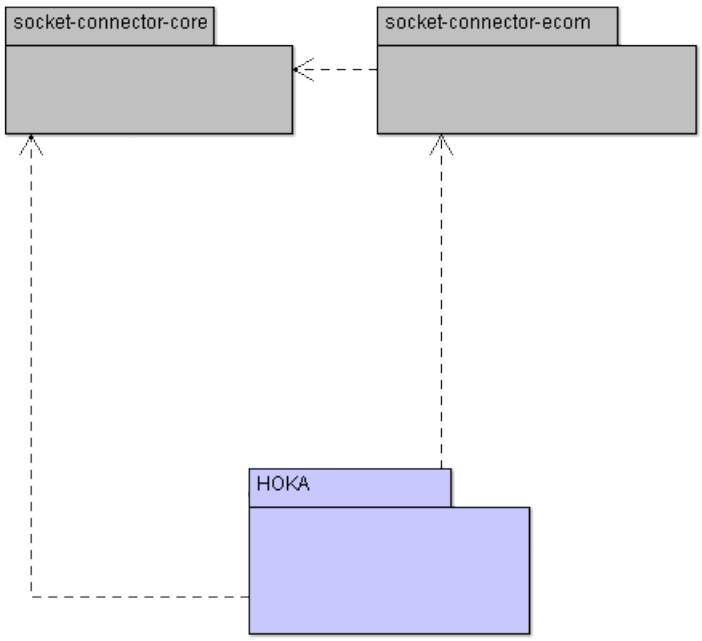

Illustration 2.1:HOKA libraries and dependencies

2.4.1 Available Libraries

The following table lists the available libraries and their descriptions:

Library name	Description
SOCKET-CONNECTOR-CORE	Generic interfaces for socket connections, see: *3.1.1.1 Socket Connector Core library*
SOCKET-CONNECTOR-NETEMBEDDED	Implementation of the generic socket connection interfaces using the NET EMBEDDED SOCKET library, see: *3.1.1.2 Socket Connector Net Embedded Library*
HOKA	The library for the HOKA Web Server, see: *4 HOKA Web Server*

Table 2-1: Available Libraries

3 Basic Networking Concepts

MicroEJ platform implements specifications for TCP/IP networking. In general, *sockets* are used to send and receive data between two parties, the *client* and *server.*

3.1.1 Generic Socket libraries

The HOKA Web Server doesn't use directly the **`NetEmbedded`** classes for socket and server socket communication. It uses the *Socket Connector Core* and the *Socket Connector Net Embedded* library as a more generic approach.

3.1.1.1 Socket Connector Core library

The *Socket Connector Core* library contains four generic interfaces (in the package **`com.is2t.connector.net`**) for socket communication:

- **`IClientSocketConnection`**: Interface for *socket* connection.
- **`IClientSocketConnectionFactory`**: Interface for *factory classes* creating **`IClientSocketConnection`** instances.
- **`IServerSocketConnection`**: Generic interface for Server socket connection.
- **`ISocketConnection`**: Generic interface for Socket connection.

3.1.1.2 Socket Connector Net Embedded Library

The *Socket Connector Net Embedded* library contains the concrete implementations for the interfaces defined in the Socket Connector Core library. There are two packages in this library:

- **`com.is2t.connector.net`**: This package contains only one abstract class, the **`SocketConnectionFactory`**, for creating *client* and *server* socket connections.
- **`com.is2t.connector.net.netembedded`**: This package contains the *concrete* implementation classes for creating *client* and *server* socket connections using net embedded.
  - **`ClientSocketConnection`**: *Client* socket connection implementation.
  - **`ClientSocketConnectionFactory`**: *Factory* class for creating ClientSocketConnection instances.
  - **`ServerSocketConnection`**: *Server* Socket Connection implementation.
  - **`SocketConnection`**: Socket Connection *implementation*.
  - **`SocketConnector`**: *Factory* class for creating socket connections.

4 HOKA Web Server

4.1 Overview

The HOKA Web Server is a tiny footprint, yet extensible web server implementation, which allows the developers to add web interface to an embedded product, and also serves as the base for the REST server implementation.

4.2 HTTPServer

The main entry point of the web server is the abstract class `com.is2t.server.http.HTTPServer`. When instantiated, the abstract method `HTTPSession newHTTPSession()` should be implemented to provide a new `com.is2t.server.http.HTTPSession` object. Next is an example creating a new `HTTPServer`. In the following example, `MySession` is the subclass of `HTTPSession` for generating the response for the HTTP request.

```
HTTPServer server = new HTTPServer(serverSocket, 10, 1) {
      protected HTTPSession newHTTPSession() {
            return new MySession(this);
      }
};
```

Illustration 4.1: Instantiating the HTTPServer

The definition of the constructor is: `public HTTPServer(IServerSocketConnection connection,int maxSimultaneousConnection, int jobCountBySession)`. The parameters of the constructor are the followings:

- **`connection`**: The `IServerSocketConnection` instance, can be created by the following code snippet, where the `PORT` constant is the number of the port the `HTTPServer` will listen to incoming request:

```
// retrieve the socket connector implementation of the platform
SocketConnectionFactory factory = SocketConnectionFactory.getImpl();
// and create a server socket with the factory
IServerSocketConnection serverSocket =
factory.newServerSocketConnection(PORT);
```

Illustration 4.2: Creating an IServerSocketConnection instance

- **`maxSimultaneousConnection`**: The maximum number of *concurrent* connections, the `HTTPServer` could handle. Internally, the `HTTPServer` will allocate an array of `IsocketConnections` with the size of `maxSimultaneousConnection`.
- **`jobCountBySession`**: the number of Threads the `HTTPServer` will utilise to process incoming requests. If this number equals to 1, only one request will be processed at a time.

4.2.1 HTTP Server API Methods

Method	Description
`public void start()`	The HTTPServer can be *started* with the `start()` method in its dedicated thread.
`public void stop()`	The `stop()` method *stops* the HTTPServer. This method will *block* the current thread until all session jobs are stopped.
`public void setLogger(Logger logger)`	Sets the *logger* for the `HTTPServer`, see chapter: *4.7 Logging*
`public Logger getLogger()`	Returns the *logger instance* of the `HTTPServer`, see: *4.7 Logging*
`public void registerEncodingHandler (IHTTPEncodingHandler handler)`	Registers a *new encoding handler* for the HTTPServer, see: *4.8 Encoding Handlers*
`public void registerTransferCodingHandler (IHTTPTransferCodingHandler handler)`	Registers a *new transfer coding handler* for the HTTPServer, see: *4.9 Transfer Coding Handlers*

Table 4-1: HTTPServer API methods

4.3 HTTPSession

The `com.is2t.server.http.HTTPSession` is an *abstract* class. Subclasses should implement the `protected abstract HTTPResponse answer(HTTPRequest request)` method to add functionality to the HTTP Server. This method receives a `com.is2t.server.http.HTTPRequest` object containing the HTTP Request data and should return an instance of `com.is2t.server.http.HTTPResponse` containing the data for HTTP Response.

4.4 HTTPRequest

The `HTTPResponse answer(HTTPRequest request)` method receives the *parsed request data* as an `HTTPRequest` instance. The public methods of the `HTTPRequest` are the followings:

- `public int getMethod()`: returns the HTTP request method. The returned value is one of the following constants defined in the `HTTPRequest` class:
  - `POST = 1`
  - `GET = 2`
  - `PUT = 3`
  - `DELETE = 4`

  The other HTTP Request methods (`HEAD`, `TRACE`, `OPTIONS`, `CONNECT`, `PATCH`) are not supported. The `HTTPServer` will reply with a status code of “400 Bad Request” in case the request method is not one of the supported types.

- `public String getURI()`: returns the *URI* part from the request.
- `public String getVersion()` returns the HTTP *version* string which is usually "HTTP/1.1"
- `public Hashtable getParameters()`: returns the `Hashtable` containing the request *parameters*.
- `public String getHeaderField(String key)`: returns the HTTP request header field given in key. The header field name is converted to *lowercase*.
- `public Hashtable getHeader()`: returns the `Hashtable` containing all of the HTTP request header fields. The header field names are converted to *lowercase*.

4.5 HTTPResponse

The class `com.is2t.server.http.HTTPResponse` is created by the `HTTPResponse answer(HTTPRequest request)` method and contains all of the data needed for the `HTTPServer` to generate the HTTP response. Internally the `HTTPResponse` stores the data to produce the HTTP response body either as a *byte array* or an *InputStream*.

There are *four* public constructors available to instantiate the `HTTPResponse`:

- `public HTTPResponse()`: Creates a new instance of `HTTPResponse` with a zero-length byte array for the response body.
- `public HTTPResponse(InputStream data)`: Creates a new instance of `HTTPResponse` using the given `InputStream` to generate the *response body*. The `HTTPServer` is using "chunked" transfer encoding[1], if this constructor is used. Please consult [WCTE] or [RFC2616] on "chunked" transfer encoding. On transfer coding *handlers*, see chapter: *4.9 Transfer Coding Handlers*.
- `public HTTPResponse(String data)`: Creates a new instance of `HTTPResponse` using the `data` parameter as the *response body*. Internally converts the String into byte array using the platform's *default* encoding which is the ISO-8859-1 (Latin-1).
- `public HTTPResponse(String data, String encoding)`: Creates a new instance of `HTTPResponse` using the data parameter as the response *body* and the specified `encoding` parameter to convert the String into array of bytes. The following *encodings* can be used:
  - ISO-8859-1: this is the platform's default encoding, but not every character can be transformed to byte using ISO-8859-1 encoding.
  - US-ASCII: US-ASCII encoding, practically this is the same as ISO-8859-1
  - UTF-8: this encoding can be used to transform *any* Java character to an array of bytes, but the support for UTF-8 should be enabled in the Run configurations, see: *Error: Reference source not found Error: Reference source not found*
- `public HTTPResponse(byte[] data)`: Creates an `HTTPResponse` using the `data`

1 The "Content-Length" header field is replaced with "Transfer-Coding" header field. The content is transmitted in one chunk.

parameter to generate the *response body*.

The public methods of `HTTPResponse` are the followings:

- **`public void setStatus`**`(String status)`: Sets the HTTP response *status*. The valid response statuses are defined in the [RFC2616]. Some common response statuses are defined in the interface `com.is2t.server.http.`**`HTTPConstants`**:
  - *`HTTP_STATUS_OK`* `= "200 OK"`
  - *`HTTP_STATUS_REDIRECT`* `= "301 Moved Permanently"`
  - *`HTTP_STATUS_NOTMODIFIED`* `= "304 Not Modified"`
  - *`HTTP_STATUS_FORBIDDEN`* `= "403 Forbidden"`
  - *`HTTP_STATUS_NOTFOUND`* `= "404 Not Found"`
  - *`HTTP_STATUS_METHOD`* `= "405 Method Not Allowed"`
  - *`HTTP_STATUS_NOTACCEPTABLE`* `= "406 Not Acceptable"`
  - *`HTTP_STATUS_BADREQUEST`* `= "400 Bad Request"`
  - *`HTTP_STATUS_MEDIA_TYPE`* `= "415 Unsupported Media Type"`
  - *`HTTP_STATUS_INTERNALERROR`* `= "500 Internal Server Error"`
  - *`HTTP_STATUS_NOTIMPLEMENTED`* `= "501 Not Implemented"`

- **`public`** `String` **`getStatus`**`()`: returns the previously set response status String.
- **`public void setMimeType`**`(String mimeType)`: sets the MIME type of the HTTP response. The *MIME type* will be used to generate the value of the response header `"Content-Type"`. Valid MIME types are defined in [RFC2046]. Common MIME types and utility methods are defined in the class `com.is2t.server.http.support.`**`MIMEUtils`**, see: *4.6 MIMEUtils*. Equivalent of setting the MIME type is to add the response header "Content-Type:" with the value of the required MIME type to the response headers. So **`setMimeType`**`("text/plain");` is equivalent to **`addHeaderField`**`("Content-Type", "text/plain");`
- **`public`** `String` **`getMimeType`**`()`: returns the previously set *MIME type* of the response.
- **`public void addHeaderField`**`(String name, String value)`: adds the *header field* given in `name` parameter with the *value* given in `value` parameter to the `Hashtable` of the response header fields.
- **`public`** `Hashtable` **`getHeader`**`()`: returns the `Hashtable` containing all response header fields.

The following example demonstrates how to create the `HTTPResponse` in the `answer(HTTPRequest request)` method to return the current *date/time* of the server:

```
public HTTPResponse answer(HTTPRequest request) {

      String body = "<h1>Server date is: <b>" + new Date() +
"</b></h1>";

      HTTPResponse response = new HTTPResponse(body);

      // Set content type as text/html
```

```
        response.setMimeType(MIMEUtils.MIME_HTML);

        // Status is "200 OK"
        response.setStatus(HTTP_STATUS_OK);

        return response;
}
```

Illustration 4.3: Creating an HTTPResponse to return the current date/time of the server

4.6 MIMEUtils

The `com.is2t.server.http.support.`**`MIMEUtils`** class provides *constant* values for commonly used *MIME types* and utility methods to return the MIME type of a resource name based on *file extensions*.

These are the predefined MIME types:

- *MIME_PLAINTEXT* = "text/plain"
- *MIME_HTML* = "text/html"
- *MIME_XML* = "text/xml"
- *MIME_DEFAULT_BINARY* = "application/octet-stream"
- *MIME_CSS* = "text/css"
- *MIME_PNG* = "image/png"
- *MIME_JPEG* = "image/jpeg"
- *MIME_GIF* = "image/gif"
- *MIME_JS* = "application/x-javascript"
- *MIME_FORM_ENCODED_DATA* = "application/x-www-form-urlencoded"

The method **`public static`** `String getMIMEType(String uri)` returns the MIME type of the given `uri`, assuming that the file extension in the `uri` was previously registered with the `mapFileExtensionToMIMEType(...)`. Only lower case file extensions are recognised.

For example calling **`getMIMEType`**`(“/images/logo.png”)` will return the String `“image/png”`.

The following table shows the predefined assignments between *file extensions* and *MIME types*:

Extension	MIME TYPE
".png"	*MIME_PNG*
".css"	*MIME_CSS*
".gif"	*MIME_GIF*
".jpeg"	*MIME_JPEG*
".jpg"	*MIME_JPEG*
".html"	*MIME_HTML*
".htm"	*MIME_HTML*
".js"	*MIME_JS*

Extension	MIME TYPE
".txt"	*MIME_PLAINTEXT*
".xml"	*MIME_XML*

Table 4-2: Predefined assignments between extensions and MIME types

The method `public static boolean mapFileExtensionToMIMEType(String fileExtension, String mimeType)`can be used to add further *file extension-MIME type* assignments. The MIME type given in the parameter `mimeType` will be assigned to the extension `fileExtension`.

4.7 Logging

The `com.is2t.server.log.Logger` interface defines the methods that a non-abstract logging class should implement to process the different log events. There are two different implementation class provided:

- `com.is2t.server.log.impl.NullLogger`: When this logger is set, nothing is logged.
- `com.is2t.server.log.impl.DefaultLogger`: This logger is set when the HTTPServer is instantiated. The log is written to the standard output.

```
Fri May 17 17:13:47 GMT 2013 | main | Server started
Fri May 17 17:16:37 GMT 2013 | HTTP-JOB-0 | 89740 | Process connection
Fri May 17 17:16:37 GMT 2013 | TCPServer | 89740 | New connection from 0.0.0.0
Fri May 17 17:16:37 GMT 2013 | HTTP-JOB-0 | 89740 | Connection closed
Fri May 17 17:16:37 GMT 2013 | HTTP-JOB-0 | 100540 | Process connection
Fri May 17 17:16:37 GMT 2013 | TCPServer | 100540 | New connection from 0.0.0.0
Fri May 17 17:16:37 GMT 2013 | TCPServer | 139104 | New connection from 0.0.0.0
Fri May 17 17:16:37 GMT 2013 | HTTP-JOB-0 | 100540 | Connection closed
Fri May 17 17:16:37 GMT 2013 | HTTP-JOB-0 | 139104 | Process connection
Fri May 17 17:16:43 GMT 2013 | HTTP-JOB-0 | 139104 | 400 Bad Request
Fri May 17 17:16:43 GMT 2013 | HTTP-JOB-0 | 139104 | Connection closed
```

Illustration 4.4: Typical output of the Default Logger

4.8 Encoding Handlers

The [RFC2616] defines the HTTP header field “Content-Encoding”. Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type[2]. For example, by adding the header field: “Content-Encoding: gzip” means that the request body is *compressed* using the “gzip” encoding.

The “Accept-Encoding” header field is used in HTTP request headers to list the acceptable encodings the web client (browser) understands. Next is an example for the Accept-Encoding header:

```
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

2 http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11

This can be interpreted as:

- The most suitable encoding is “gzip” encoding (quality argument is 1.0)
- The second best option is “identity” encoding (quality argument is 0.5)
- Any other encoding is not acceptable (quality argument is 0)

The `HTTPServer` selects the most suitable *encoding* for the response depending on what encoding handlers are *registered*. The “identity” encoding handler is registered by default, any other encoding handlers should be implemented and registered by the *application developers*.

`HTTPServer` defines the interface: `com.is2t.server.http.encoding.`**`IHTTPEncodingHandler`** for *encoding* handlers. The “identity” encoding handler is implemented and registered by default.

When creating a new *encoding handler,* the following methods should be implemented:

Method	Description
`public String getId()`	Returns the *name* of the encoding. The name of the encoding is compared with the value of the “Content-Encoding” header field value, and if matches, the encoding handler will be used for encoding/decoding the content.
`public InputStream open(InputStream original) throws IOException`	Returns an `InputStream` to *decode* the *content* of the HTTP request.
`public OutputStream open(OutputStream original) throws IOException`	Returns an `OutputStream` to *encode* the *content*. This method will be used to encode the body of the HTTP response

Table 4-3: `IHTTPEncodingHandler` methods

4.9 Transfer Coding Handlers

The HTTP header field name “Transfer-Coding” defines how the body of the HTTP request or response is *encoded*. The [RFC2616] defines the following *valid* transfer codings:

Transfer coding name	Description
chunked	Transfer in a series of chunks
compress	UNIX "compress" program method
deflate	"zlib" format with "deflate" compression
gzip	Same as GNU zip
identity	(withdrawn in errata to [RFC2616])

Table 4-4: Valid Transfer Codings according to [RFC2616]

The interface for defining transfer coding handlers for the HTTPServer is `com.is2t.server.http.encoding.`**`IHTTPTransferCodingHandler`**.

The following methods should be implemented for the **`IHTTPTransferCodingHandler`** interface:

Method	Description
`public String` **`getId`**`()`	Returns the *name* of the transfer coding handler. This should corresponds to a valid transfer coding name, e.g.: “gzip”
`public InputStream` **`open`**`(HTTPRequest request, InputStream input)` **`throws`** `IOException`	Returns an InputStream to *decode* the encoded body of the HTTP request.
`public OutputStream` **`open`**`(HTTPResponse response, OutputStream output)` **`throws`** `IOException`	Returns an OutputStream to *encode* the body of the HTTP response.

Table 4-5: `IHTTPTransferCodingHandler` methods

The HTTPServer implements the “chunked” and “identity” transfer coding handlers, they are also registered by default.

- The class `com.is2t.server.http.encoding.impl.`**`ChunkedTransferCodingHandler`** handles the “chunked” encoding, with the help of an input and output stream implementation:
  - `com.is2t.server.http.encoding.impl.`**`ChunkedMessageBodyInputStream`** for reading request bodies in “chunked” encoding.
  - `com.is2t.server.http.encoding.impl.`**`ChunkedMessageBodyOutputStream`** for writing response bodies in “chunked” encoding.
- `The class` `com.is2t.server.http.encoding.impl.`**`IdentityTransferCodingHandler`** is for handling the “identity” encoding.
  - `com.is2t.server.http.encoding.impl.`**`IdentityMessageBodyInputStream`** for handling the request body
  - `com.is2t.server.http.encoding.impl.`**`IdentityMessageBodyOutputStream`** for handling the response body

4.10 Caching

The `HTTPServer` checks for the presence of the “IF-NONE-MATCH” header field, and if this header field is present in the request header, it will return instantly a “304 Not Modified” response. This mechanism could be used to prevent static resources (e.g. images) to be retrieved *multiple times* by browsers implementing an internal *cache*.

To employ this mechanism, the embedded application should put an “ETAG” header field in the responses returning a *static resource*, which should be cached by a *browser*. The

value of the “ETAG” header field is usually a “fingerprint” generated from the content of the resource.

The standard handling of the “IF-NONE-MATCH” request header field includes comparing the value of the “IF-NONE-MATCH” header field with the current “fingerprint” of the resource, but the `HTTPServer`'s mechanism is much simpler: if the “IF-NONE-MATCH” header field is present, it simply returns the “304 Not Modified” response.

The above mechanism has the following consequences:

- Applications should include the “ETAG” response header field, if they want the resource to be cached by browsers.
- Since the “IF-NONE-MATCH” request header field value (the “fingerprint”) is not compared with the resource's actual “fingerprint”, applications should avoid using this mechanism for any resource which could change (e.g.: HTML files).

4.11 Character Encodings

The default encoding of the MicroEJ platform is ISO-8859-1, likewise the default encoding of a HTTP response is ISO-8859-1. So if no encoding specified in the HTTP response headers, it is assumed that the response is encoded in ISO-8859-1.

What to do, when the response contains characters, which cannot be represented in ISO-8859-1? The UTF-8 can be used, since all browsers supports this encoding. To enable UTF-8 encoding, two steps are required:

- The appropriate constructor should be used to create a new HTTPResponse: `public HTTPResponse(String data, String encoding)`, where the encoding should be "`UTF-8`"
- The "Content-Type" response header should be supplemented by the specification of the used character encoding: Content-type: text/plain**; charset=utf-8**

In the following example, the use of the UTF-8 encoding is presented[3]:

```
String responseBody = "πάντα χωρεῖ καὶ οὐδὲν μένει καὶ δὶς ἐς τὸν αὐτὸν
      ποταμὸν οὐκ ἂν ἐμβαίης – All things move and nothing remains still,
      and you cannot step twice into the same stream";
HTTPResponse response;
try {
      response = new HTTPResponse(responseBody, "UTF-8");
} catch (UnsupportedEncodingException e) {
      e.printStackTrace();
}
```

Illustration 4.5: Creating a `HTTPResponse` using UTF-8 encoding

When using `byte[]` or `InputStream` source data for the `HTTPResponse`, the bytes should be already encoded in UTF-8 when passing to the `HTTPResponse`'s constructor. In this case only the "Content-Type" response header should be supplemented with the "`;charset=utf-8`" definition for the UTF-8 encoding.

3 Although it is legal for a Java class file to contain Unicode characters, it is best to avoid placing special characters into the source files. The best is to use "Unicode escaping". For example the Thai word: "□□□" (egg) can be escaped as: "`\u0e44\u0e02\u0e48`"

5 API Documentation

Package com.is2t.connector.net

Interface Summary		Page
IClientSocketConnection	Generic Client Socket connection interface.	*21*
IClientSocketConnectionFactory	Factory for creating new instances of `IClientSocketConnection`.	*24*
IServerSocketConnection	Generic Server Socket connection interface.	*25*
ISocketConnection	Generic Socket connection interface.	*27*

Class Summary		Page
SocketConnectionFactory	Provide an entry point to create abstract socket connectors (client and server).	*29*

Interface IClientSocketConnection

com.is2t.connector.net

All Known Implementing Classes:

ClientSocketConnection

```
public interface IClientSocketConnection
```

Generic Client Socket connection interface. (J2ME/J2SE independent)

Method Summary		Page
void	**close**() Closes the socket connection.	23
ISocketConnection	**connect**(String uri, int port) Connects to a remote socket using the address specified by `port` and `uri`.	22
String	**getAddress**() Returns the remote address of the connection.	23
int	**getPort**() Returns the local port number for the socket connection.	23

Method Detail

connect

```
ISocketConnection connect(String uri,
                          int port)
                   throws IOException
```

Connects to a remote socket using the address specified by `port` and `uri`.

Parameters:

`uri` - the IP address

`port` - the port number for the connection

Returns:

the opened `ISocketConnection`

Throws:

`IOException` - if an I/O error occurs during opening the connection

---

close

```
void close()
    throws IOException
```

Closes the socket connection.

Throws:

`IOException` - if an I/O error occurs during closing the connection

---

getAddress

```
String getAddress()
           throws IOException
```

Returns the remote address of the connection.

Returns:

the local address of the connection

Throws:

`IOException` - if the connection is closed

---

getPort

```
int getPort()
     throws IOException
```

Returns the local port number for the socket connection.

Returns:

the local port number for the socket connection

Throws:

`IOException` - if the connection is closed

Interface IClientSocketConnectionFactory

com.is2t.connector.net

All Known Implementing Classes:

ClientSocketConnectionFactory

```
public interface IClientSocketConnectionFactory
```

Factory for creating new instances of `IClientSocketConnection`.

Method Summary		Page
`IClientSocketConnection`	**`getNewClientSocketConnection`**`(String host, int port)` Returns a new instance of `IClientSocketConnection`.	*24*

Method Detail

getNewClientSocketConnection

```
IClientSocketConnection getNewClientSocketConnection(String host,
                                                     int port)
```

Returns a new instance of `IClientSocketConnection`.

Parameters:

`host` - the IP address

`port` - the port number

Returns:

new instance of `IClientSocketConnection`

Interface IServerSocketConnection

com.is2t.connector.net

All Known Implementing Classes:

ServerSocketConnection

```
public interface IServerSocketConnection
```

Generic Server Socket connection interface. (J2ME/J2SE independent)

Method Summary		Page
ISocketConnection	**accept**() Returns a ISocketConnection object that represents a server side socket connection.	*25*
void	**close**() Closes the server connection.	*26*
String	**getAddress**() Returns the local address of this connection.	*26*
int	**getPort**() Returns the local port number of this connection.	*26*

Method Detail

accept

```
ISocketConnection accept()
                  throws IOException
```

Returns a ISocketConnection object that represents a server side socket connection. The method blocks until a connection is made.

Returns:

an instance of ISocketConnection

Throws:

IOException - when opening the connection is not possible

---

close

```
void close()
    throws IOException
```

Closes the server connection.

Throws:

`IOException` - when the connection is already closed

---

getAddress

```
String getAddress()
           throws IOException
```

Returns the local address of this connection.

Returns:

the local address of this connection

Throws:

`IOException` - when the connection is already closed

---

getPort

```
int getPort()
     throws IOException
```

Returns the local port number of this connection.

Returns:

the local port number of this connection

Throws:

`IOException` - when the connection is already closed

Interface ISocketConnection

com.is2t.connector.net

All Known Implementing Classes:

SocketConnection

```
public interface ISocketConnection
```

Generic Socket connection interface. (J2ME/J2SE independent)

Method Summary		Page
void	**close**() Closes the current connection.	*28*
String	**getAddress**() Returns the address of this connection.	*28*
InputStream	**getInputStream**() Opens an `InputStream` on this connection.	*27*
OutputStream	**getOutputStream**() Opens an `OutputStream` on this connection.	*28*

Method Detail

getInputStream

```
InputStream getInputStream()
                    throws IOException
```

Opens an `InputStream` on this connection.

Returns:

the opened `InputStream`

Throws:

`IOException` - if an I/O error occurs when creating the input stream, the socket is closed or the socket is not connected

---

getOutputStream

```
OutputStream getOutputStream()
                      throws IOException
```

Opens an `OutputStream` on this connection.

Returns:
the opened `OutputStream`

Throws:
`IOException` - if an I/O error occurs when creating the `OutputStream` or if the socket is not connected.

---

close

```
void close()
    throws IOException
```

Closes the current connection.

Throws:
`IOException` - if an I/O error occurs when closing this socket.

---

getAddress

```
String getAddress()
           throws IOException
```

Returns the address of this connection.

Returns:
the address

Throws:
`IOException` - if the socket is not connected.

Class SocketConnectionFactory

com.is2t.connector.net

```
java.lang.Object
  └ com.is2t.connector.net.SocketConnectionFactory
```

Direct Known Subclasses:

SocketConnector

```
abstract public class SocketConnectionFactory
extends Object
```

Provide an entry point to create abstract socket connectors (client and server).

Constructor Summary	Page
SocketConnectionFactory()	*30*

Method Summary		Page
static SocketConnectionFactory	**getImpl**() Returns the SocketConnectionFactory implementation of the platform.	*30*
abstract IClientSocketConnection	**newClientSocketConnection**(String host, int port) According to platform, creates a new implementation of IClientSocketConnection.	*30*
abstract IClientSocketConnectionFactory	**newClientSocketConnectionFactory**() According to platform, creates a new instance of IClientSocketConnectionFactory.	*30*
abstract IServerSocketConnection	**newServerSocketConnection**(int port) According to platform, creates a new implementation of IServerSocketConnection.	*31*
abstract ISocketConnection	**newSocketConnection**(String host, int port) According to platform, creates a new implementation of ISocketConnection.	*31*

Constructor Detail

SocketConnectionFactory

```
public SocketConnectionFactory()
```

Method Detail

getImpl

```
public static SocketConnectionFactory getImpl()
```

Returns the `SocketConnectionFactory` implementation of the platform.

Returns:
new instance of the `SocketConnectionFactory`.

---

newClientSocketConnection

```
public abstract IClientSocketConnection newClientSocketConnection(String host,
                                                                  int port)
                                                           throws IOException
```

According to platform, creates a new implementation of `IClientSocketConnection`.

Parameters:
`host` - the host's IP address
`port` - the port number
Returns:
a new instance of `IClientSocketConnection`
Throws:
`IOException` - if I/O error occurs during the creation of the `IClientSocketConnection`

---

newClientSocketConnectionFactory

```
public abstract IClientSocketConnectionFactory newClientSocketConnectionFactory(
)
                                                                        throws
IOException
```

According to platform, creates a new instance of `IClientSocketConnectionFactory`.

Returns:
a new instance of `IClientSocketConnectionFactory`

Throws:

`IOException` - if I/O error occurs during the creation of the `IClientSocketConnection`

---

newServerSocketConnection

```
public abstract IServerSocketConnection newServerSocketConnection(int port)
                                                           throws IOException
```

According to platform, creates a new implementation of `IServerSocketConnection`.

Parameters:

`port` - the local port number for the server socket

Returns:

a new instance of `IServerSocketConnection`

Throws:

`IOException` - if I/O error occurs during the creation of the `IServerSocketConnection`

---

newSocketConnection

```
public abstract ISocketConnection newSocketConnection(String host,
                                                      int port)
                                               throws IOException
```

According to platform, creates a new implementation of `ISocketConnection`.

Parameters:

`host` - the host's IP address

`port` - the host's port number

Returns:

a new instance of `ISocketConnection`

Throws:

`IOException` - `ISocketConnection`

Package com.is2t.connector.net.netembedded

Class Summary		Page
ClientSocketConnection	Net Embedded Client socket connection implementation.	*32*
ClientSocketConnectionFactory	Factory for creating `ISocketConnection` instances.	*36*
ServerSocketConnection	Net Embedded connection.	*38*
SocketConnection	Net Embedded Socket connection implementation.	*41*
SocketConnector	Net Embedded socket connector implementation.	*44*

Class ClientSocketConnection

com.is2t.connector.net.netembedded

```
java.lang.Object
  └ com.is2t.connector.net.netembedded.ClientSocketConnection
```

All Implemented Interfaces:

IClientSocketConnection

```
public class ClientSocketConnection
extends Object
implements IClientSocketConnection
```

Net Embedded Client socket connection implementation.

Constructor Summary	Page
ClientSocketConnection(String host, int port) Constructs a new instance of ClientSocketConnection with the specified port and host.	33

Method Summary		Page
void	**close**() Closes the ClientSocketConnection.	34
ISocketConnection	**connect**(String uri, int port) Creates a new ISocketConnection using the uri and port.	34
String	**getAddress**() Returns the address of this connection.	34
int	**getPort**() Returns the port of this connection.	35

Constructor Detail

ClientSocketConnection

```
public ClientSocketConnection(String host,
                              int port)
                       throws IOException
```

Constructs a new instance of `ClientSocketConnection` with the specified `port` and `host`.

Parameters:

`host` - the remote IP address

`port` - the remote port number

Throws:

`IOException` - when I/O error occurs during connecting to the remote host

Method Detail

close

```
public void close()
           throws IOException
```

Closes the `ClientSocketConnection`.

Specified by:

`close` in interface `IClientSocketConnection`

Throws:

`IOException` - if I/O error occurs when closing the connection

---

connect

```
public ISocketConnection connect(String uri,
                                 int port)
                          throws IOException
```

Creates a new `ISocketConnection` using the `uri` and `port`.

Specified by:

`connect` in interface `IClientSocketConnection`

Parameters:

`uri` - the IP address

`port` - the remote port number

Returns:

the `ISocketConnection`

Throws:

`IOException` - if I/O error occurs during connecting to the remote host.

---

getAddress

```
public String getAddress()
                  throws IOException
```

Returns the address of this connection.

Specified by:
`getAddress` in interface `IClientSocketConnection`

Returns:
the address of this connection

Throws:
`IOException` - if the connection is closed

---

getPort

```
public int getPort()
            throws IOException
```

Returns the port of this connection.

Specified by:
`getPort` in interface `IClientSocketConnection`

Returns:
the port of this connection

Throws:
`IOException` - when the connection is closed

Class ClientSocketConnectionFactory

com.is2t.connector.net.netembedded

```
java.lang.Object
  └ com.is2t.connector.net.netembedded.ClientSocketConnectionFactory
```

All Implemented Interfaces:

IClientSocketConnectionFactory

```
public class ClientSocketConnectionFactory
extends Object
implements IClientSocketConnectionFactory
```

Factory for creating `ISocketConnection` instances.

Constructor Summary	Page
`ClientSocketConnectionFactory()`	*36*

Method Summary		Page
`IClientSocketConnection`	`getNewClientSocketConnection(String host, int port)` Creates a new instance of `IClientSocketConnection` using the `host` and `port`.	*36*

Constructor Detail

ClientSocketConnectionFactory

```
public ClientSocketConnectionFactory()
```

Method Detail

getNewClientSocketConnection

```
public IClientSocketConnection getNewClientSocketConnection(String host,
                                                            int port)
```

Creates a new instance of `IClientSocketConnection` using the `host` and `port`. A RuntimeException is thrown if I/O error occurs during the creation of this connection.

Specified by:

`getNewClientSocketConnection` in interface `IClientSocketConnectionFactory`

Parameters:

`host` - the remote host's IP address

`port` - the remote port number

Returns:

a new instance of `IClientSocketConnection`

Class ServerSocketConnection

com.is2t.connector.net.netembedded

```
java.lang.Object
  └ com.is2t.connector.net.netembedded.ServerSocketConnection
```

All Implemented Interfaces:

IServerSocketConnection

```
public class ServerSocketConnection
extends Object
implements IServerSocketConnection
```

Net Embedded connection.

Constructor Summary	Page
ServerSocketConnection(int port) Creates a new connection wrapper on the given port.	*38*

Method Summary		Page
ISocketConnection	**accept**() Returns a ISocketConnection object that represents a server side socket connection.	*39*
void	**close**() Closes the connection.	*39*
String	**getAddress**() Returns the IP address of this connection.	*39*
int	**getPort**() Returns the port number for this connection.	*40*

Constructor Detail

ServerSocketConnection

```
public ServerSocketConnection(int port)
                       throws IOException
```

Creates a new connection wrapper on the given port.

Parameters:
`port` - the local port number
Throws:
`IOException` - if I/O error occurs creating the server socket connection

Method Detail

accept

```
public ISocketConnection accept()
                        throws IOException
```

Returns a `ISocketConnection` object that represents a server side socket connection. The method blocks until a connection is made.

Specified by:
`accept` in interface `IServerSocketConnection`
Returns:
the `ISocketConnection` instance
Throws:
`IOException` - if I/O error occurs when opening the connection

---

close

```
public void close()
            throws IOException
```

Closes the connection.

Specified by:
`close` in interface `IServerSocketConnection`
Throws:
`IOException` - if the connection is closed

---

getAddress

```
public String getAddress()
                    throws IOException
```

Returns the IP address of this connection.

Specified by:
`getAddress` in interface `IServerSocketConnection`

Returns:
the IP address of this connection
Throws:
`IOException` - if the connection is closed

---

getPort

```
public int getPort()
            throws IOException
```

Returns the port number for this connection.

Specified by:
`getPort` in interface `IServerSocketConnection`
Returns:
the port number for this connection
Throws:
`IOException` - if the connection is closed.

Class SocketConnection

com.is2t.connector.net.netembedded

```
java.lang.Object
  └ com.is2t.connector.net.netembedded.SocketConnection
```

All Implemented Interfaces:

ISocketConnection

```
public class SocketConnection
extends Object
implements ISocketConnection
```

Net Embedded Socket connection implementation.

Constructor Summary	Page
SocketConnection(java.net.Socket socket) Creates a new connection wrapper for the given SocketConnection.	*41*

Method Summary		Page
void	**close**() Closes the connection.	*42*
String	**getAddress**() Returns the IP address of the SocketConnection.	*42*
InputStream	**getInputStream**() Opens an InputStream on this socket connection.	*42*
OutputStream	**getOutputStream**() Opens an OutputStream on this connection.	*43*

Constructor Detail

SocketConnection

```
public SocketConnection(java.net.Socket socket)
```

Creates a new connection wrapper for the given SocketConnection.

Parameters:
`socket` - the `SocketConnection`

Method Detail

close

```
public void close()
           throws IOException
```

Closes the connection.

Specified by:
`close` in interface `ISocketConnection`
Throws:
`IOException` - when I/O error occurs when closing the connection

---

getAddress

```
public String getAddress()
                  throws IOException
```

Returns the IP address of the `SocketConnection`.

Specified by:
`getAddress` in interface `ISocketConnection`
Returns:
the IP address of the `SocketConnection`.
Throws:
`IOException` - if the connection is closed.

---

getInputStream

```
public InputStream getInputStream()
                           throws IOException
```

Opens an `InputStream` on this socket connection.

Specified by:
`getInputStream` in interface `ISocketConnection`
Returns:
the opened `InputStream`
Throws:
`IOException` - if an I/O error occurs when creating the input stream, the socket is closed, the socket is not connected

getOutputStream

```
public OutputStream getOutputStream()
                            throws IOException
```

Opens an `OutputStream` on this connection.

Specified by:

`getOutputStream` in interface `ISocketConnection`

Returns:

the opened `OutputStream`

Throws:

`IOException` - if an I/O error occurs when creating the output stream or if the socket is not connected.

Class SocketConnector

com.is2t.connector.net.netembedded

```
java.lang.Object
  └ com.is2t.connector.net.SocketConnectionFactory
      └ com.is2t.connector.net.netembedded.SocketConnector
```

```
public class SocketConnector
extends SocketConnectionFactory
```

Net Embedded socket connector implementation.

Constructor Summary	*Page*
SocketConnector()	*44*

Method Summary		*Page*
IClientSocketConnection	**newClientSocketConnection**(String host, int port) Creates a new instance of IClientSocketConnection.	*45*
IClientSocketConnectionFactory	**newClientSocketConnectionFactory**() Creates a new instance of IClientSocketConnectionFactory.	*45*
IServerSocketConnection	**newServerSocketConnection**(int port) Creates a new instance of IServerSocketConnection.	*45*
ISocketConnection	**newSocketConnection**(String host, int port) According to platform, creates a new implementation of ISocketConnection.	*46*

Methods inherited from class com.is2t.connector.net.SocketConnectionFactory
getImpl

Constructor Detail

SocketConnector

```
public SocketConnector()
```

Method Detail

newClientSocketConnection

```
public IClientSocketConnection newClientSocketConnection(String host,
                                                         int port)
```

Creates a new instance of `IClientSocketConnection`.

Overrides:
`newClientSocketConnection` in class `SocketConnectionFactory`
Parameters:
`host` - the host name to reach
`port` - the port of the connection to establish
Returns:
a new instance of `IClientSocketConnection`

---

newClientSocketConnectionFactory

```
public IClientSocketConnectionFactory newClientSocketConnectionFactory()
```

Creates a new instance of `IClientSocketConnectionFactory`.

Overrides:
`newClientSocketConnectionFactory` in class `SocketConnectionFactory`
Returns:
a new instance of `IClientSocketConnectionFactory`

---

newServerSocketConnection

```
public IServerSocketConnection newServerSocketConnection(int port)
                                                  throws IOException
```

Creates a new instance of `IServerSocketConnection`.

Overrides:
`newServerSocketConnection` in class `SocketConnectionFactory`
Parameters:
`port` - the local port number for the server socket
Returns:
a new instance of `IServerSocketConnection`
Throws:
`IOException` - if I/O error occurs during the creation of the `IServerSocketConnection`

---

newSocketConnection

```
public ISocketConnection newSocketConnection(String host,
                                             int port)
                                      throws IOException
```

According to platform, creates a new implementation of `ISocketConnection`.

Overrides:

`newSocketConnection` in class `SocketConnectionFactory`

Parameters:

`host` - the host's IP address

`port` - the host's port number

Returns:

a new instance of `ISocketConnection`

Throws:

`IOException` - `ISocketConnection`

Package com.is2t.server.http

Interface Summary		Page
HTTPConstants	Constants for HTTP statuses and header fields.	*51*

Class Summary		Page
CalibrationConstants	Configuration for `HTTPSession`.	*47*
DefaultHTTPSession	Default HTTP Session implementation.	*49*
HTTPRequest	Represents a HTTP Request.	*57*
HTTPResponse	Represents a HTTP Response.	*61*
HTTPServer	Abstract HTTP Server.	*65*
HTTPSession	Abstract HTTP Session.	*69*

Class CalibrationConstants

com.is2t.server.http

```
java.lang.Object
  └ com.is2t.server.http.CalibrationConstants
```

```
public class CalibrationConstants
extends Object
```

Configuration for `HTTPSession`.

Field Summary		Page
static boolean	**STRICT_ACCEPT_ENCODING_COMPLIANCE** if `true` the server should send a `HTTPConstants.HTTP_STATUS_NOTACCEPTABLE` if there is no `IHTTPEncodingHandler` registered for to handle the encoding specified in the HTTP request.	*48*

Constructor Summary	Page
CalibrationConstants()	*48*

Field Detail

STRICT_ACCEPT_ENCODING_COMPLIANCE

```
public static final boolean STRICT_ACCEPT_ENCODING_COMPLIANCE
```

if `true` the server should send a `HTTPConstants.HTTP_STATUS_NOTACCEPTABLE` if there is no `IHTTPEncodingHandler` registered for to handle the encoding specified in the HTTP request.

See Also:
`HTTPServer.registerEncodingHandler(com.is2t.server.http.encoding.IHTTPEncodingHandler)`

Constructor Detail

CalibrationConstants

```
public CalibrationConstants()
```

Class DefaultHTTPSession

com.is2t.server.http

```
java.lang.Object
  └ com.is2t.server.http.HTTPSession
      └ com.is2t.server.http.DefaultHTTPSession
```

```
public class DefaultHTTPSession
extends HTTPSession
```

Default HTTP Session implementation.

Retrieves the URI of the request and tries to find a matching resource.

Example:

Given the URI `http://192.168.1.1/my/wonderful/resource.html`, the Default HTTP Session will try to find the resource `/my/wonderful/resource.html` in the application's classpath (using `Class.getResourceAsStream(String)`).

Constructor Summary	Page
DefaultHTTPSession(HTTPServer server) Default constructor.	49

Method Summary		Page
HTTPResponse	**answer**(HTTPRequest request) The generic behaviour of this session implementation is to find a resource matching the given URI in the classpath.	50

Methods inherited from class com.is2t.server.http.HTTPSession
createErrorResponse

Constructor Detail

DefaultHTTPSession

```
public DefaultHTTPSession(HTTPServer server)
```

Default constructor.

Parameters:

`server` - the `HTTPServer` to associate with this session.

Method Detail

answer

```
public HTTPResponse answer(HTTPRequest request)
```

The generic behaviour of this session implementation is to find a resource matching the given URI in the classpath. The resource is included in the HTTP Response with the proper MIME-Type and HTTP Status (200 OK).

If no resource found, a HTTP 404 error response is returned.

Parameters:

`request` - the `HTTPRequest`

Returns:

the `HTTPResponse` containing the resource

Interface HTTPConstants

com.is2t.server.http

All Known Implementing Classes:

RestResourceHTML, RestResourceRawText, RestSession

`public interface` **`HTTPConstants`**

Constants for HTTP statuses and header fields.

Field Summary		Page
`String`	**`CONNECTION_FIELD_VALUE_CLOSE`** Value for HTTP header field "Connection" (`close`).	*56*
`String`	**`CONNECTION_FIELD_VALUE_KEEP_ALIVE`** Value for HTTP header field "Connection" (`keep-alive`).	*56*
`String`	**`FIELD_ACCEPT_ENCODING`** HTTP header field (in lower case) `content-encoding`.	*56*
`String`	**`FIELD_CONNECTION`** HTTP header field (in lower case) `connection`.	*56*
`String`	**`FIELD_CONTENT_ENCODING`** HTTP header field (in lower case) `content-encoding`.	*55*
`String`	**`FIELD_CONTENT_LENGTH`** HTTP header field (in lower case) `content-length`.	*56*
`String`	**`FIELD_CONTENT_TYPE`** HTTP header field (in lower case) `content-type`.	*55*
`String`	**`FIELD_IF_NONE_MATCH`** HTTP header field (in lower case) `if-none-match`.	*56*
`String`	**`FIELD_TRANSFER_ENCODING`** HTTP header field (in lower case) `transfer-encoding`.	*55*

String	**HTTP_METHOD_DELETE** HTTP `DELETE` method token as String.	*55*
String	**HTTP_METHOD_GET** HTTP `GET` method token as String.	*55*
String	**HTTP_METHOD_POST** HTTP `POST` method token as String.	*55*
String	**HTTP_METHOD_PUT** HTTP `PUT` method token as String.	*55*
String	**HTTP_STATUS_BADREQUEST** HTTP code 400: the request is not valid.	*54*
String	**HTTP_STATUS_FORBIDDEN** HTTP code 403: the client doesn't have the permission to access the requested URL.	*53*
String	**HTTP_STATUS_INTERNALERROR** HTTP code 500: the server has encountered an error while generating the response.	*54*
String	**HTTP_STATUS_MEDIA_TYPE** HTTP code 415: the requested resource type is not supported.	*54*
String	**HTTP_STATUS_METHOD** HTTP code 405: the HTTP request method (GET/POST/PUT/DELETE) is not allowed on the server for the requested URI.	*54*
String	**HTTP_STATUS_NOTACCEPTABLE** HTTP code 406: the client cannot handle the data returned in the HTTP response.	*54*
String	**HTTP_STATUS_NOTFOUND** HTTP code 404: the requested URL has not been found.	*54*

String	**HTTP_STATUS_NOTIMPLEMENTED** HTTP code 501: the HTTP request method is not implemented.	*54*
String	**HTTP_STATUS_NOTMODIFIED** HTTP code 304: the requested resources hasn't been modified since the last time.	*53*
String	**HTTP_STATUS_OK** HTTP code 200: the response has been found and correctly sent.	*53*
String	**HTTP_STATUS_REDIRECT** HTTP code 301: the requested URL redirected to another URL.	*53*

Field Detail

HTTP_STATUS_OK

```
public static final String HTTP_STATUS_OK
```

HTTP code 200: the response has been found and correctly sent.

HTTP_STATUS_REDIRECT

```
public static final String HTTP_STATUS_REDIRECT
```

HTTP code 301: the requested URL redirected to another URL.

HTTP_STATUS_NOTMODIFIED

```
public static final String HTTP_STATUS_NOTMODIFIED
```

HTTP code 304: the requested resources hasn't been modified since the last time.

HTTP_STATUS_FORBIDDEN

```
public static final String HTTP_STATUS_FORBIDDEN
```

HTTP code 403: the client doesn't have the permission to access the requested URL.

HTTP_STATUS_NOTFOUND

```
public static final String HTTP_STATUS_NOTFOUND
```

HTTP code 404: the requested URL has not been found.

HTTP_STATUS_METHOD

```
public static final String HTTP_STATUS_METHOD
```

HTTP code 405: the HTTP request method (GET/POST/PUT/DELETE) is not allowed on the server for the requested URI.

HTTP_STATUS_NOTACCEPTABLE

```
public static final String HTTP_STATUS_NOTACCEPTABLE
```

HTTP code 406: the client cannot handle the data returned in the HTTP response.

HTTP_STATUS_BADREQUEST

```
public static final String HTTP_STATUS_BADREQUEST
```

HTTP code 400: the request is not valid.

HTTP_STATUS_MEDIA_TYPE

```
public static final String HTTP_STATUS_MEDIA_TYPE
```

HTTP code 415: the requested resource type is not supported.

HTTP_STATUS_INTERNALERROR

```
public static final String HTTP_STATUS_INTERNALERROR
```

HTTP code 500: the server has encountered an error while generating the response.

HTTP_STATUS_NOTIMPLEMENTED

```
public static final String HTTP_STATUS_NOTIMPLEMENTED
```

HTTP code 501: the HTTP request method is not implemented.

HTTP_METHOD_POST

```
public static final String HTTP_METHOD_POST
```

HTTP `POST` method token as String.

HTTP_METHOD_GET

```
public static final String HTTP_METHOD_GET
```

HTTP `GET` method token as String.

HTTP_METHOD_PUT

```
public static final String HTTP_METHOD_PUT
```

HTTP `PUT` method token as String.

HTTP_METHOD_DELETE

```
public static final String HTTP_METHOD_DELETE
```

HTTP `DELETE` method token as String.

FIELD_CONTENT_TYPE

```
public static final String FIELD_CONTENT_TYPE
```

HTTP header field (in lower case) `content-type`.

FIELD_CONTENT_ENCODING

```
public static final String FIELD_CONTENT_ENCODING
```

HTTP header field (in lower case) `content-encoding`.

FIELD_TRANSFER_ENCODING

```
public static final String FIELD_TRANSFER_ENCODING
```

HTTP header field (in lower case) `transfer-encoding`. See RFC HTTP/1.1 RFC2616 3.6.

FIELD_ACCEPT_ENCODING

`public static final String` **`FIELD_ACCEPT_ENCODING`**

HTTP header field (in lower case) `content-encoding`.

FIELD_CONTENT_LENGTH

`public static final String` **`FIELD_CONTENT_LENGTH`**

HTTP header field (in lower case) `content-length`.

FIELD_IF_NONE_MATCH

`public static final String` **`FIELD_IF_NONE_MATCH`**

HTTP header field (in lower case) `if-none-match`.

FIELD_CONNECTION

`public static final String` **`FIELD_CONNECTION`**

HTTP header field (in lower case) `connection`.

CONNECTION_FIELD_VALUE_KEEP_ALIVE

`public static final String` **`CONNECTION_FIELD_VALUE_KEEP_ALIVE`**

Value for HTTP header field "Connection" (`keep-alive`).

CONNECTION_FIELD_VALUE_CLOSE

`public static final String` **`CONNECTION_FIELD_VALUE_CLOSE`**

Value for HTTP header field "Connection" (`close`).

Class HTTPRequest

com.is2t.server.http

```
java.lang.Object
  └ com.is2t.server.http.HTTPRequest
```

```
final public class HTTPRequest
extends Object
```

Represents a HTTP Request.

Field Summary		Page
static int	**DELETE** Value returned by getMethod() if the request method is DELETE.	*58*
static int	**GET** Value returned by getMethod() if the request method is GET.	*58*
static int	**POST** Value returned by getMethod() if the request method is POST.	*58*
static int	**PUT** Value returned by getMethod() if the request method is PUT.	*58*

Method Summary		Page
Hashtable	**getHeader**() Returns all HTTP Header fields of the request.	*60*
String	**getHeaderField**(String key) Returns the header field value associated to the given header field key.	*59*
int	**getMethod**() Returns the request method as an integer value which is one of POST, GET, PUT or DELETE.	*58*

`Hashtable`	**`getParameters`**`()` Returns the query parameters as `Hashtable`.	*59*
`String`	**`getURI`**`()` Returns the request URI.	*59*
`String`	**`getVersion`**`()` Returns the HTTP version request.	*59*

Field Detail

POST

```
public static final int POST
```

Value returned by `getMethod()` if the request method is `POST`.

GET

```
public static final int GET
```

Value returned by `getMethod()` if the request method is `GET`.

PUT

```
public static final int PUT
```

Value returned by `getMethod()` if the request method is `PUT`.

DELETE

```
public static final int DELETE
```

Value returned by `getMethod()` if the request method is `DELETE`.

Method Detail

getMethod

```
public int getMethod()
```

Returns the request method as an integer value which is one of `POST`, `GET`, `PUT` or `DELETE`.

Returns:
the request method (one of `POST`, `GET`, `PUT` or `DELETE`).

---

getURI

```
public String getURI()
```

Returns the request URI.

Returns:
the request URI string.

---

getVersion

```
public String getVersion()
```

Returns the HTTP version request.

Returns:
the HTTP version request string.

---

getParameters

```
public Hashtable getParameters()
```

Returns the query parameters as `Hashtable`.

Returns:
a `Hashtable` of (String,String) representing the HTTP Query Parameters.

---

getHeaderField

```
public String getHeaderField(String key)
```

Returns the header field value associated to the given header field `key`.

Parameters:
`key` - a header field name (if `null`, `null` is returned).
Returns:
the requested header field value, `null` if the header field is not found.

---

getHeader

```
public Hashtable getHeader()
```

Returns all HTTP Header fields of the request.

Returns:

a `Hashtable` of (String,String) representing the HTTP Header Fields (may be empty).

Class HTTPResponse

com.is2t.server.http

```
java.lang.Object
  └ com.is2t.server.http.HTTPResponse
```

Direct Known Subclasses:

RestResponse

```
public class HTTPResponse
extends Object
```

Represents a HTTP Response.

Constructor Summary	Page
HTTPResponse() Creates an empty HTTPResponse.	*62*
HTTPResponse(byte[] data) Creates a new HTTPResponse using the given byte array as response data.	*63*
HTTPResponse(InputStream data) Creates a new HTTPResponse using the given InputStream as the response data.	*62*
HTTPResponse(String data) Creates a new HTTPResponse using the given String as response data.	*62*
HTTPResponse(String data, String encoding) Creates a new HTTPResponse using the String data as response data and the encoding.	*63*

Method Summary		Page
void	**addHeaderField**(String name, String value) Adds a response header field.	*64*
Hashtable	**getHeader**() Returns the response header.	*64*

`String`	**`getMimeType`**`()` Returns the MIME-TYPE of the response.	*64*
`String`	**`getStatus`**`()` Returns the response status.	*63*
`void`	**`setMimeType`**`(String mimeType)` Set the response MIME-TYPE.	*64*
`void`	**`setStatus`**`(String status)` Set the response status.	*63*

Constructor Detail

HTTPResponse

`public` **`HTTPResponse`**`()`

Creates an empty `HTTPResponse`.

HTTPResponse

`public` **`HTTPResponse`**`(InputStream data)`

Creates a new `HTTPResponse` using the given `InputStream` as the response data.

Parameters:

`data` - the data to send through response (as a stream)

HTTPResponse

`public` **`HTTPResponse`**`(String data)`

Creates a new `HTTPResponse` using the given `String` as response data. The `data` is transformed into bytes using the `ISO-8859-1` encoding.

Parameters:

`data` - the data to send through response (as a raw string)

HTTPResponse

```
public HTTPResponse(String data,
                    String encoding)
             throws UnsupportedEncodingException
```

Creates a new `HTTPResponse` using the `String` `data` as response data and the `encoding`.

Parameters:

`data` - the `String` to be used as response body.
`encoding` - the encoding used to transform the `String` `data` to bytes. The following encodings can be used:

- `ISO-8859-1` ISO-8859-1 encoding, always supported by the platform
- `UTF-8` UTF-8 encoding, only supported if the "Embed UTF-8 encoding" option is enabled in the Run Configurations. If this option is not set, an `UnsupportedEncodingException` is thrown.
- `US-ASCII` US-ASCII encoding

Throws:

`UnsupportedEncodingException` - when the specified encoding is not supported.

HTTPResponse

```
public HTTPResponse(byte[] data)
```

Creates a new `HTTPResponse` using the given byte array as response data.

Parameters:

`data` - the data to send through response (as a raw byte array)

Method Detail

getStatus

```
public String getStatus()
```

Returns the response status.

Returns:

the response status.

---

setStatus

```
public void setStatus(String status)
```

Set the response status.

Parameters:

`status` - the response status to set. Should be one of the `HTTP_STATUS_*` constants defined in `HTTPConstants`

---

getMimeType

```
public String getMimeType()
```

Returns the MIME-TYPE of the response.

Returns:

the response MIME-TYPE.

---

setMimeType

```
public void setMimeType(String mimeType)
```

Set the response MIME-TYPE.

Parameters:

`mimeType` - the response MIME-TYPE to set.

---

addHeaderField

```
public void addHeaderField(String name,
                           String value)
```

Adds a response header field.

Parameters:

`name` - name of the header field to set.
`value` - value of the header filed.

---

getHeader

```
public Hashtable getHeader()
```

Returns the response header.

Returns:

a `Hashtable` of (String,String) representing the HTTP Header Fields (may be empty).

Class HTTPServer

com.is2t.server.http

```
java.lang.Object
  └ com.is2t.server.ip.Server
      └ com.is2t.server.tcp.TCPServer
          └ com.is2t.server.http.HTTPServer
```

Direct Known Subclasses:

RestServer

```
abstract public class HTTPServer
extends TCPServer
```

Abstract HTTP Server. Subclasses should override the `newHTTPSession()` method to add specific session handling behaviour.

Features + limitations:

- CLDC 1.1
- No fixed configuration files, logging, authorization, encryption.
- Supports parameter parsing of GET and POST methods
- Supports both dynamic content and file serving
- Never caches anything
- Doesn't limit bandwidth, request time or simultaneous connections
- Contains a built-in list of most common MIME types
- All header names are converted to lower case

Override `HTTPSession.answer(HTTPRequest)` and redefine the server behaviour for your own application

Example:

```
// get a new server which handle a Default HTTP Session
HTTPServer server = new HTTPServer(serverSocket, 10, 1) {
     protected HTTPSession newHTTPSession() {
           return new DefaultHTTPSession(this);
     }
};

// start the server
server.start();
```

Constructor Summary	*Page*

HTTPServer(IServerSocketConnection connection, int maxSimultaneousConnection, int jobCountBySession) Creates a HTTPServer using the given IServerSocketConnection .	*66*

Method Summary		*Page*
void	**registerEncodingHandler**(IHTTPEncodingHandler handler) Registers a new HTTP content encoding handler.	*67*
void	**registerTransferCodingHandler**(IHTTPTransferCodingHandler handler) Registers a new HTTP transfer coding handler.	*67*
void	**start**() Start the HTTPServer (in a dedicated thread): start listening for connections and start session jobs.	*67*
void	**stop**() Stops the HTTPServer.	*68*

Methods inherited from class com.is2t.server.tcp.TCPServer
isStopped

Methods inherited from class com.is2t.server.ip.Server
getLogger, setLogger

Constructor Detail

HTTPServer

```
public HTTPServer(IServerSocketConnection connection,
                  int maxSimultaneousConnection,
                  int jobCountBySession)
```

Creates a HTTPServer using the given IServerSocketConnection .

The default encoding to be used is the identity encoding. Further encodings may be registered using registerEncodingHandler(IHTTPEncodingHandler).

Server is not started until start() is called.

Parameters:

`connection` - the `IServerSocketConnection` connection used by the server
`maxSimultaneousConnection` - the maximal number of simultaneously opened connections.
`jobCountBySession` - the number of parallel jobs to process by opened sessions. if `jobCountBySession` == 1, the jobs are processed sequentially.

Method Detail

registerEncodingHandler

```
public void registerEncodingHandler(IHTTPEncodingHandler handler)
```

Registers a new HTTP content encoding handler.

Should be called before `start()`, otherwise a `RuntimeException` is thrown.

Parameters:

`handler` - the `IHTTPEncodingHandler` to register

---

registerTransferCodingHandler

```
public void registerTransferCodingHandler(IHTTPTransferCodingHandler handler)
```

Registers a new HTTP transfer coding handler.

Should be called before `start()`, otherwise a `RuntimeException` is raised.

Parameters:

`handler` - the `IHTTPTransferCodingHandler` to register

---

start

```
public void start()
```

Start the `HTTPServer` (in a dedicated thread): start listening for connections and start session jobs.
Multiple start is not allowed.

Overrides:

`start` in class `TCPServer`

---

stop

```
public void stop()
```

Stops the `HTTPServer`. Stops listening for connections. This method blocks until all session jobs are stopped.

Overrides:

`stop` in class `TCPServer`

Class HTTPSession

com.is2t.server.http

```
java.lang.Object
  └ com.is2t.server.http.HTTPSession
```

Direct Known Subclasses:
DefaultHTTPSession, RestSession

```
abstract public class HTTPSession
extends Object
```

Abstract HTTP Session. Subclasses implements the `answer(HTTPRequest)` method to generate `HTTPResponse` to a `HTTPRequest`.

Constructor Summary	Page
`HTTPSession``(HTTPServer server)` Creates a new HTTP Session in the given `HTTPServer`.	*69*

Method Summary		Page
static `HTTPResponse`	**`createErrorResponse`**`(String status, String msg)` Create a `HTTPResponse` to write the `msg` for the given `status`.	*69*

Constructor Detail

HTTPSession

```
public HTTPSession(HTTPServer server)
```

Creates a new HTTP Session in the given `HTTPServer`.

Parameters:
`server` - a `HTTPServer`

Method Detail

createErrorResponse

```
public static HTTPResponse createErrorResponse(String status,
                                               String msg)
```

Create a `HTTPResponse` to write the `msg` for the given `status`.

Parameters:

`status` - the error status. One of `HTTP_STATUS_*` constant of the `HTTPConstants` interface.

`msg` - an optional error message to add in response.

Returns:

a `HTTPResponse` that represent the error.

See Also:

`HTTPConstants.HTTP_STATUS_BADREQUEST`,
`HTTPConstants.HTTP_STATUS_FORBIDDEN`,
`HTTPConstants.HTTP_STATUS_INTERNALERROR`,
`HTTPConstants.HTTP_STATUS_MEDIA_TYPE`,
`HTTPConstants.HTTP_STATUS_METHOD`,
`HTTPConstants.HTTP_STATUS_NOTACCEPTABLE`,
`HTTPConstants.HTTP_STATUS_NOTFOUND`,
`HTTPConstants.HTTP_STATUS_NOTIMPLEMENTED`,
`HTTPConstants.HTTP_STATUS_NOTMODIFIED`,
`HTTPConstants.HTTP_STATUS_OK`,
`HTTPConstants.HTTP_STATUS_REDIRECT`

Package com.is2t.server.http.encoding

Interface Summary		Page
IHTTPEncodingHandler	Interface for defining HTTP encoding handlers.	*71*
IHTTPTransferCodingHandler	Interface for defining HTTP transfer coding handlers.	*74*

Exception Summary		Page
UnsupportedHTTPEncodingException	This exception is thrown when `HTTPConstants.FIELD_TRANSFER_ENCODING` or `HTTPConstants.FIELD_CONTENT_ENCODING` cannot be handled by the server.	*76*

Interface IHTTPEncodingHandler

com.is2t.server.http.encoding
All Known Implementing Classes:
IdentityEncodingHandler

`public interface` **`IHTTPEncodingHandler`**

Interface for defining HTTP encoding handlers. A HTTP encoding handler is able to decode data from an `InputStream` and encode data to an `OutputStream`.

Encodings are specified in HTTP headers such as `content-encoding`, `transfer-encoding`, `accept-encoding`.

Encoding handlers should be registered in the `HTTPServer` in order to use them.

See Also:
`HTTPServer.registerEncodingHandler(IHTTPEncodingHandler)`

Method Summary		Page
`String`	**`getId`**`()` Returns the name of the supported encoding.	*72*
`InputStream`	**`open`**`(InputStream original)` Returns an `InputStream` to read the decoded data from the `originalInputStream`.	*73*
`OutputStream`	**`open`**`(OutputStream original)` Wraps the `originalOutputStream` with a special `OutputStream` which performs the encoding.	*73*

Method Detail

getId

`String` **`getId`**`()`

Returns the name of the supported encoding.

Returns:
an internal `String` in lower case format.

---

open

```
InputStream open(InputStream original)
          throws IOException
```

Returns an `InputStream` to read the decoded data from the `originalInputStream`.

Parameters:
`original` - the `InputStream` to read the encoded data.
Returns:
the `InputStream` to read the decoded data.
Throws:
`IOException` - if any I/O error occurs

---

open

```
OutputStream open(OutputStream original)
           throws IOException
```

Wraps the `originalOutputStream` with a special `OutputStream` which performs the encoding. Returns an `OutputStream` to encode the data from the `originalOutputStream`.

Parameters:
`original` - the output stream to wrap
Returns:
the `OutputStream` to encode the data.
Throws:
`IOException` - if any I/O error occurs

Interface IHTTPTransferCodingHandler

com.is2t.server.http.encoding
All Known Implementing Classes:
ChunkedTransferCodingHandler, IdentityTransferCodingHandler

```
public interface IHTTPTransferCodingHandler
```

Interface for defining HTTP transfer coding handlers.

The HTTP transfer coding handler decodes data from the body of a `HTTPRequest` and encodes the body of a `HTTPResponse`.

Transfer coding is specified in `transfer-encoding` HTTP header.

Encoding handlers should be registered in the `HTTPServer` in order to use them.

See Also:
`HTTPServer.registerTransferCodingHandler(IHTTPTransferCodingHandler)`

Method Summary		Page
`String`	**`getId`**`()` Returns the supported encoding id.	*74*
`InputStream`	**`open`**`(HTTPRequest request, InputStream input)` Opens an `InputStream` that can be used to decode message body of the given request.	*75*
`OutputStream`	**`open`**`(HTTPResponse response, OutputStream output)` Opens an `OutputStream` that can be used to encode the message body of the `HTTPResponse`.	*75*

Method Detail

getId

```
String getId()
```

Returns the supported encoding id.

Returns:
an internal `String` in lower case format.

---

open

```
InputStream open(HTTPRequest request,
                 InputStream input)
          throws IOException
```

Opens an `InputStream` that can be used to decode message body of the given request. The returned `InputStream` MUST conforms to the followings:

- The `InputStream` MUST reach the EOF (i.e. read methods returns `-1`) when the request body has been completely read.
- The `InputStream.close()` method MUST read any remaining bytes from the message body (if any) and MUST NOT close the underlying stream.

Parameters:

`request` - the `HTTPRequest` to be decoded by this transfer coding handler.
`input` - the `InputStream` from which encoded message body can be read.

Returns:

the `InputStream` used to decode message body of the given request

Throws:

`IOException` - if any I/O Error occurs

---

open

```
OutputStream open(HTTPResponse response,
                  OutputStream output)
           throws IOException
```

Opens an `OutputStream` that can be used to encode the message body of the `HTTPResponse`.

Parameters:

`response` - the `HTTPResponse` to be encoded by this transfer coding handler.
`output` - the `OutputStream` where the encoded message body is written.

Returns:

the output stream used to encode message body of the given response

Throws:

`IOException` - if any I/O Error occurs

Class UnsupportedHTTPEncodingException

com.is2t.server.http.encoding

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └java.io.IOException
              └java.io.UnsupportedEncodingException
                  └com.is2t.server.http.encoding.UnsupportedHTTPEncodingExcepti
on
```

All Implemented Interfaces:

Serializable

```
public class UnsupportedHTTPEncodingException
extends UnsupportedEncodingException
```

This exception is thrown when `HTTPConstants.FIELD_TRANSFER_ENCODING` or `HTTPConstants.FIELD_CONTENT_ENCODING` cannot be handled by the server.

Field Summary		Page
String	**encoding** The encoding which is not supported.	*77*
String	**field** The HTTP Header field causing the error.	*76*

Constructor Summary	Page
UnsupportedHTTPEncodingException(String field, String encoding) Creates a new UnsupportedHTTPEncodingException with the given parameters.	*77*

Field Detail

field

```
public final String field
```

The HTTP Header field causing the error.

---

encoding

```
public final String encoding
```

The encoding which is not supported.

Constructor Detail

UnsupportedHTTPEncodingException

```
public UnsupportedHTTPEncodingException(String field,
                                        String encoding)
```

Creates a new `UnsupportedHTTPEncodingException` with the given parameters.

Parameters:

`field` - the HTTP Header field causing the error.
`encoding` - the encoding which is not supported.

Package com.is2t.server.http.encoding.impl

Class Summary		Page
ChunkedMessageBodyInputStream	Input Stream for reading HTTP 1.1 Chunked transfer encoding data.	78
ChunkedMessageBodyOutputStream	Output Stream for writing HTTP 1.1 Chunked transfer encoding data.	82
ChunkedTransferCodingHandler	HTTP-1.1 chunked transfer encoding handler to read and write data in chunked encoding.	85
IdentityEncodingHandler	HTTP-1.1 Identity encoding handler.	88
IdentityMessageBodyInputStream	Identity input stream.	90
IdentityMessageBodyOutputStream	Identity output stream.	94
IdentityTransferCodingHandler	Identity transfer coding handler.	98

Class ChunkedMessageBodyInputStream

com.is2t.server.http.encoding.impl

```
java.lang.Object
  └java.io.InputStream
      └com.is2t.server.http.encoding.impl.ChunkedMessageBodyInputStream
```

All Implemented Interfaces:

Closeable

```
public class ChunkedMessageBodyInputStream
extends InputStream
```

Input Stream for reading HTTP 1.1 Chunked transfer encoding data.

Each chunk starts with the number of octets of the data it embeds, expressed as a hexadecimal numbers in ASCII, followed by optional parameters (chunk extension) and a terminating CRLF sequence, followed by the chunk data. The chunk is terminated by CRLF. If chunk extensions are provided, the chunk size is terminated by a semicolon followed with the extension name and an optional equal sign and value. (chunk extensions are skipped).

Constructor Summary	Page
ChunkedMessageBodyInputStream(InputStream is) Creates a new ChunkedMessageBodyInputStream using the InputStream is as the underlying data source.	*80*

	Method Summary	Page
int	**available**() Returns the number of available bytes to read.	*80*
void	**close**() Reads all remaining message body data and closes this input stream.	*80*
int	**read**() Reads the next byte from the InputStream used in the constructor.	*81*
int	**read**(byte[] data, int offset, int length) Reads up to length bytes to the byte array data starting with offset.	*81*

Constructor Detail

ChunkedMessageBodyInputStream

```
public ChunkedMessageBodyInputStream(InputStream is)
```

Creates a new `ChunkedMessageBodyInputStream` using the InputStream `is` as the underlying data source.

Parameters:
`is` - InputStream to read from.

Method Detail

available

```
public int available()
             throws IOException
```

Returns the number of available bytes to read.

Overrides:
`available` in class `InputStream`
Returns:
the number of available bytes to read.
Throws:
`IOException` - if the `ChunkedMessageBodyInputStream` is already closed.

---

close

```
public void close()
           throws IOException
```

Reads all remaining message body data and closes this input stream. This method DOES NOT close the underlying stream (i.e. the TCP connection stream). It is the responsibility of the HTTPSession to close the underlying stream.

Specified by:
`close` in interface `Closeable`
Overrides:
`close` in class `InputStream`
Throws:
`IOException` - when an error occurs while closing the stream

---

read

```
public int read()
         throws IOException
```

Reads the next byte from the InputStream used in the constructor.

Overrides:
`read` in class `InputStream`

Returns:
the next byte value (0-255) or -1 if the end of the InputStream has been reached.

Throws:
`IOException` - when the InputStream has already been closed.

---

read

```
public int read(byte[] data,
                int offset,
                int length)
           throws IOException
```

Reads up to `length` bytes to the byte array `data` starting with `offset`. The method tries to read `length` bytes (if there are at least `length` bytes in the InputStream used in the constructor. Otherwise reads just the available number of bytes).

Overrides:
`read` in class `InputStream`

Parameters:
`data` - the byte array to store the bytes read from the underlying InputStream
`offset` - the position in the byte array `data` to store the bytes read from the InputStream.
`length` - number of bytes to store in the byte array `data` (if the number of bytes available in the InputStream are at least `length`) If less than `length` bytes are left in the InputStream, only the remaining bytes are stored in the byte array `data`.

Returns:
the number of bytes stored in the byte array `data`, or -1 if no bytes can be stored in the byte array `data`.

Throws:
`IOException` - when the end of InputStream has already been reached.

Class ChunkedMessageBodyOutputStream

com.is2t.server.http.encoding.impl

```
java.lang.Object
  └java.io.OutputStream
      └com.is2t.server.http.encoding.impl.ChunkedMessageBodyOutputStream
```

All Implemented Interfaces:

Closeable, Flushable

```
public class ChunkedMessageBodyOutputStream
extends OutputStream
```

Output Stream for writing HTTP 1.1 Chunked transfer encoding data.

Each chunk starts with the number of octets of the data it embeds, expressed as a hexadecimal numbers in ASCII and a terminating CRLF sequence, followed by the chunk data. The chunk is terminated by CRLF.

Constructor Summary	Page
`ChunkedMessageBodyOutputStream(OutputStream os)` Creates a new instance of `ChunkedMessageBodyOutputStream` using the specified `OutputStream` as the underlying OutputStream.	*83*

	Method Summary	Page
void	`close()` Close this output stream.	*83*
void	`flush()` Writes the pending data and flush underlying stream.	*83*
void	`write(byte[] b, int off, int len)` Writes the content of the byte array `b` from the offset `off` in length `len` in chunked encoding using the underlying `OutputStream`.	*84*
void	`write(int b)` Write one byte of data.	*84*

Constructor Detail

ChunkedMessageBodyOutputStream

```
public ChunkedMessageBodyOutputStream(OutputStream os)
```

Creates a new instance of `ChunkedMessageBodyOutputStream` using the specified `OutputStream` as the underlying OutputStream.

Parameters:

`os` - the underlying `OutputStream` to use

Method Detail

close

```
public final void close()
                  throws IOException
```

Close this output stream. This method DOES NOT close the underlying stream (i.e. the TCP connection stream). It is the responsibility of the HTTPSession to close the underlying stream.

Specified by:

`close` in interface `Closeable`

Overrides:

`close` in class `OutputStream`

Throws:

`IOException` - when an error occurs while closing the stream

---

flush

```
public final void flush()
                  throws IOException
```

Writes the pending data and flush underlying stream.

Specified by:

`flush` in interface `Flushable`

Overrides:

`flush` in class `OutputStream`

Throws:

`IOException` - when the connection is closed.

---

write

```
public final void write(byte[] b,
                        int off,
                        int len)
                 throws IOException
```

Writes the content of the byte array `b` from the offset `off` in length `len` in chunked encoding using the underlying `OutputStream`.

Overrides:

`write` in class `OutputStream`

Parameters:

`b` - the byte array

`off` - the starting index in byte array `b`

`len` - the number of bytes written to the underlying Output stream in chunked encoding.

Throws:

`IOException` - if the ChunkedMessageBodyOutputStream is already closed.

---

write

```
public final void write(int b)
                 throws IOException
```

Write one byte of data. The byte is not sent immediately, it is stored in a buffer and will be sent in a chunk when the buffer is full.

Overrides:

`write` in class `OutputStream`

Parameters:

`b` - the byte to write to the underlying `OutputStream`

Throws:

`IOException` - if the ChunkedMessageBodyOutputStream is already closed.

See Also:

`OutputStream.write(int)`

Class ChunkedTransferCodingHandler

com.is2t.server.http.encoding.impl

```
java.lang.Object
  └ com.is2t.server.http.encoding.impl.ChunkedTransferCodingHandler
```

All Implemented Interfaces:

IHTTPTransferCodingHandler

```
public class ChunkedTransferCodingHandler
extends Object
implements IHTTPTransferCodingHandler
```

HTTP-1.1 chunked transfer encoding handler to read and write data in chunked encoding.

Method Summary		Page
String	**getId**() Returns the internal ID of the ChunkedTransferCodingHandler.	*86*
static ChunkedTransferCodingHandler	**getInstance**() Factory method to create an instance of ChunkedTransferCodingHandler.	*85*
InputStream	**open**(HTTPRequest request, InputStream input) Creates a ChunkedMessageBodyInputStream to read the body of the HTTP request in "chunked" encoding from the HTTPRequest and the InputStream.	*86*
OutputStream	**open**(HTTPResponse response, OutputStream output) Creates an OutputStream to write the body of the HTTP response in "chunked" encoding using the HTTPResponse and the OutputStream.	*86*

Method Detail

getInstance

```
public static ChunkedTransferCodingHandler getInstance()
```

Factory method to create an instance of ChunkedTransferCodingHandler.

Returns:
an instance of `ChunkedTransferCodingHandler`

---

getId

```
public String getId()
```

Returns the internal ID of the `ChunkedTransferCodingHandler`.

Specified by:
`getId` in interface `IHTTPTransferCodingHandler`
Returns:
the String "chunked".

---

open

```
public InputStream open(HTTPRequest request,
                        InputStream input)
                 throws IOException
```

Creates a `ChunkedMessageBodyInputStream` to read the body of the HTTP request in "chunked" encoding from the `HTTPRequest` and the `InputStream`.

Specified by:
`open` in interface `IHTTPTransferCodingHandler`
Parameters:
`request` - the `HTTPRequest`
`input` - the `InputStream`
Returns:
a new instance of `ChunkedMessageBodyInputStream`
Throws:
`IOException` - when I/O error occurs

---

open

```
public OutputStream open(HTTPResponse response,
                         OutputStream output)
                  throws IOException
```

Creates an `OutputStream` to write the body of the HTTP response in "chunked" encoding using the `HTTPResponse` and the `OutputStream`.

Specified by:

`open` in interface `IHTTPTransferCodingHandler`

Parameters:

`response` - the `HTTPResponse`

`output` - the `OutputStream`

Returns:

a new instance of `ChunkedMessageBodyOutputStream`

Throws:

`IOException` - if an I/O error occurs

Class IdentityEncodingHandler

com.is2t.server.http.encoding.impl

```
java.lang.Object
  └ com.is2t.server.http.encoding.impl.IdentityEncodingHandler
```

All Implemented Interfaces:

IHTTPEncodingHandler

```
final public class IdentityEncodingHandler
extends Object
implements IHTTPEncodingHandler
```

HTTP-1.1 Identity encoding handler.

Method Summary		Page
String	**getId**() Returns the internal ID of the IdentityEncodingHandler.	*89*
static IdentityEncodingHandler	**getInstance**() Returns an instance of IdentityEncodingHandler.	*88*
InputStream	**open**(InputStream original) Returns the originalInputStream.	*89*
OutputStream	**open**(OutputStream original) Returns the originalOutputStream.	*89*

Method Detail

getInstance

```
public static IdentityEncodingHandler getInstance()
```

Returns an instance of IdentityEncodingHandler.

Returns:

an instance of IdentityEncodingHandler

---

getId

```
public String getId()
```

Returns the internal ID of the `IdentityEncodingHandler`.

Specified by:
`getId` in interface `IHTTPEncodingHandler`
Returns:
the string "identity".

---

open

```
public InputStream open(InputStream original)
                 throws IOException
```

Returns the `originalInputStream`.

Specified by:
`open` in interface `IHTTPEncodingHandler`
Parameters:
`original` - the `InputStream` to return
Returns:
the `originalInputStream`
Throws:
`IOException` - not thrown

---

open

```
public OutputStream open(OutputStream original)
                  throws IOException
```

Returns the `originalOutputStream`.

Specified by:
`open` in interface `IHTTPEncodingHandler`
Parameters:
`original` - the `OutputStream` to return
Returns:
the `originalOutputStream`
Throws:
`IOException` - not thrown

Class IdentityMessageBodyInputStream

com.is2t.server.http.encoding.impl

```
java.lang.Object
  └java.io.InputStream
      └com.is2t.server.http.encoding.impl.IdentityMessageBodyInputStream
```

All Implemented Interfaces:

Closeable

```
public class IdentityMessageBodyInputStream
extends InputStream
```

Identity input stream. Wraps an `InputStream` and all of the operations on `IdentityMessageBodyInputStream` are delegated to this underlying `InputStream`.

Constructor Summary	Page
`IdentityMessageBodyInputStream``(InputStream is, int bodyLength)` Creates a new instance of `IdentityMessageBodyInputStream` with the `InputStream is` with the predefined length `bodyLength`.	*91*

	Method Summary	Page
`int`	**`available`**`()` Returns the number of bytes that can be read (or skipped over) from this input stream without blocking.	*91*
`void`	**`close`**`()` Reads all remaining message body data and then close this input stream.	*91*
`int`	**`read`**`()` Reads the next byte of data from the input stream.	*92*
`int`	**`read`**`(byte[] data, int offset, int length)` Reads up to `length` bytes of data from the underlying `InputStream` into an array of bytes.	*92*

Constructor Detail

IdentityMessageBodyInputStream

```
public IdentityMessageBodyInputStream(InputStream is,
                                      int bodyLength)
```

Creates a new instance of `IdentityMessageBodyInputStream` with the `InputStream` `is` with the predefined length `bodyLength`. The `bodyLength` should be the maximum number of bytes can be read from the InputStream.

Parameters:

`is` - the underlying `InputStream` to read the body content of the HTTP message body
`bodyLength` - the number of bytes can be read from the underlying InputStream.

Method Detail

available

```
public int available()
            throws IOException
```

Returns the number of bytes that can be read (or skipped over) from this input stream without blocking.

Overrides:

`available` in class `InputStream`

Returns:

the number of available bytes could be read from the stream

Throws:

`IOException` - if IO Error occurs.

---

close

```
public void close()
          throws IOException
```

Reads all remaining message body data and then close this input stream. This method DOES NOT close the underlying stream (i.e. the TCP connection stream). It is the responsibility of the HTTPSession to close the underlying stream.

Specified by:

`close` in interface `Closeable`

Overrides:

`close` in class `InputStream`

Throws:

`IOException` - when an error occurs while closing the stream

---

read

```
public int read()
         throws IOException
```

Reads the next byte of data from the input stream. The byte value is returned as an int in the range of 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

Overrides:

`read` in class `InputStream`

Returns:

the next byte of data from the input stream, or -1 if the end of the input stream has been reached.

Throws:

`IOException` - If a premature EOF is reached, this stream is closed and an IOException is thrown.

---

read

```
public int read(byte[] data,
                int offset,
                int length)
         throws IOException
```

Reads up to `length` bytes of data from the underlying `InputStream` into an array of bytes. An attempt is made to read as many as `length` bytes, but the amount of bytes can be read from the `InputStream` could be less than `length`, even `0`. The number of bytes actually read are returned as an integer.

Overrides:

`read` in class `InputStream`

Parameters:

`data` - the byte array to store the read bytes from the underlying `InputStream`
`offset` - the starting index of byte array `data` to store the bytes
`length` - the number of bytes intended to be read by the caller of this method

Returns:

the number of bytes actually read from the underlying `InputStream`. If the end of stream has been reached, returns `-1`

Throws:

`IOException` - thrown in the following cases:

- if this stream is already closed
- if an EOF is reached prematurely on the underlying InputStream

See Also:

`InputStream.read(byte[], int, int)`

Class IdentityMessageBodyOutputStream

com.is2t.server.http.encoding.impl

```
java.lang.Object
  └java.io.OutputStream
      └com.is2t.server.http.encoding.impl.IdentityMessageBodyOutputStream
```

All Implemented Interfaces:

Closeable, Flushable

```
public class IdentityMessageBodyOutputStream
extends OutputStream
```

Identity output stream. Wraps an `OutputStream` and all of the operations on `IdentityMessageBodyOutputStream` are delegated to this underlying `OutputStream`.

Constructor Summary	Page
`IdentityMessageBodyOutputStream``(OutputStream os)` Creates a new instance of `IdentityMessageBodyOutputStream` using the `OutputStream` as the underlying OutputStream to write the data to.	*95*

	Method Summary	Page
void	**`close`**`()` Closes this output stream and flushes the underlying `OutputStream` .	*95*
void	**`flush`**`()` Flushes the underlying output stream.	*95*
void	**`write`**`(byte[] b)` Writes the content of the byte array to the underlying output stream.	*96*
void	**`write`**`(byte[] b, int off, int len)` Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.	*96*
void	**`write`**`(int b)` Writes the specified byte to this output stream.	*96*

Constructor Detail

IdentityMessageBodyOutputStream

```
public IdentityMessageBodyOutputStream(OutputStream os)
```

Creates a new instance of `IdentityMessageBodyOutputStream` using the `OutputStream` as the underlying OutputStream to write the data to.

Parameters:

`os` - the underlying `OutputStream` to use

Method Detail

close

```
public final void close()
                  throws IOException
```

Closes this output stream and flushes the underlying `OutputStream` . This method DOES NOT close the underlying stream (i.e. the TCP connection stream). It is the responsibility of the `HTTPSession` to close the underlying stream.

Specified by:

`close` in interface `Closeable`

Overrides:

`close` in class `OutputStream`

Throws:

`IOException` - when an I/O error occurs while closing the stream

---

flush

```
public final void flush()
                  throws IOException
```

Flushes the underlying output stream.

Specified by:

`flush` in interface `Flushable`

Overrides:

`flush` in class `OutputStream`

Throws:

`IOException` - if this output stream has already been closed.

---

write

```
public final void write(byte[] b)
                throws IOException
```

Writes the content of the byte array to the underlying output stream.

Overrides:
`write` in class `OutputStream`
Parameters:
`b` - the byte array to read bytes from and write to the underlying output stream.
Throws:
`IOException` - if this output stream has already been closed.

---

write

```
public final void write(byte[] b,
                        int off,
                        int len)
                throws IOException
```

Writes `len` bytes from the specified byte array starting at offset `off` to this output stream. The general contract for write(b, off, len) is that some of the bytes in the array `b` are written to the output stream in order; element b[off] is the first byte written and b[off+len-1] is the last byte written by this operation. If `b` is null, a `NullPointerException` is thrown. If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

Overrides:
`write` in class `OutputStream`
Parameters:
`b` - the byte array to read bytes from
`off` - the starting index in byte array `b` to read bytes from
`len` - number of bytes to read from the byte array `b`
Throws:
`IOException` - if this output stream has already been closed

---

write

```
public final void write(int b)
                throws IOException
```

Writes the specified byte to this output stream. The general contract for write is that one byte is written to the output stream. The byte to be written is the eight low-order bits of the argument `b`. The 24 high-order bits of b are ignored.

Overrides:

`write` in class `OutputStream`

Parameters:

`b` - the value to be written to the underlying output stream

Throws:

`IOException` - if this output stream has already been closed

Class IdentityTransferCodingHandler

com.is2t.server.http.encoding.impl

```
java.lang.Object
  └ com.is2t.server.http.encoding.impl.IdentityTransferCodingHandler
```

All Implemented Interfaces:

IHTTPTransferCodingHandler

```
public class IdentityTransferCodingHandler
extends Object
implements IHTTPTransferCodingHandler
```

Identity transfer coding handler.

Method Summary		Page
String	**getId**() Returns an internal ID of this encoding handler.	*99*
static IdentityTransferCodingHandler	**getInstance**() Returns an instance of IdentityTransferCodingHandler.	*98*
InputStream	**open**(HTTPRequest request, InputStream input) Creates a new instance of IdentityMessageBodyInputStream to read the message body of the HTTP request.	*99*
OutputStream	**open**(HTTPResponse response, OutputStream output) Creates an IdentityMessageBodyOutputStream to write the message body of the HTTP response.	*99*

Method Detail

getInstance

```
public static IdentityTransferCodingHandler getInstance()
```

Returns an instance of IdentityTransferCodingHandler.

Returns:
an instance of `IdentityTransferCodingHandler`

---

getId

```
public String getId()
```

Returns an internal ID of this encoding handler.

Specified by:
`getId` in interface `IHTTPTransferCodingHandler`
Returns:
null

---

open

```
public InputStream open(HTTPRequest request,
                        InputStream input)
                 throws IOException
```

Creates a new instance of `IdentityMessageBodyInputStream` to read the message body of the HTTP request.

Specified by:
`open` in interface `IHTTPTransferCodingHandler`
Parameters:
`request` - the HTTP request object
`input` - the input stream to read the message body of the HTTP request
Returns:
`IdentityMessageBodyOutputStream`
Throws:
`IOException` - if an I/O error occurs
See Also:
`IdentityMessageBodyInputStream`

---

open

```
public OutputStream open(HTTPResponse response,
                         OutputStream output)
                  throws IOException
```

Creates an `IdentityMessageBodyOutputStream` to write the message body of the HTTP response.

Specified by:

`open` in interface `IHTTPTransferCodingHandler`

Parameters:

`response` - the `HTTPResponse`

`output` - the `OutputStream` to write the message body of the HTTP response

Returns:

`IdentityMessageBodyOutputStream`

Throws:

`IOException` - when I/O error occurs

See Also:

`IdentityMessageBodyOutputStream`

Package com.is2t.server.log.impl

Class Summary		Page
DefaultLogger	The default logger for `HTTPServer`.	*101*
NullLogger	The Null logger implements the `Logger` interface but does not log anything.	*107*

Class DefaultLogger

com.is2t.server.log.impl

```
java.lang.Object
  └ com.is2t.server.log.impl.DefaultLogger
```

All Implemented Interfaces:

Logger

```
public class DefaultLogger
extends Object
implements Logger
```

The default logger for `HTTPServer`.

Field Summary		Page
static String	**FIELD_SEP** Field separator character ("Pipe" character: "|").	*103*

Constructor Summary	Page
DefaultLogger(PrintStream out) Creates a new logger.	*103*

Method Summary		Page
void	**connectionClosed**(ISocketConnection c) Logs the "Connection closed" message.	*104*
void	**connectionLost**(ISocketConnection c, IOException e) Logs the "Connection lost([reason])" message.	*104*
void	**dumpEvent**(String message) Logs an event.	*104*
void	**httpError**(ISocketConnection c, String status, String message) Logs the `status` and `message`.	*104*

void	**newConnection**(ISocketConnection c) Displays the message "New connection from [remote IP address]" with the hash code of the ISocketConnection instance c.	*105*
void	**processConnection**(ISocketConnection c) Displays the message "Process connection" with the hash code of the ISocketConnection instance c.	*105*
void	**serverStarted**() Displays the message "Server started".	*105*
void	**serverStopped**() Displays the message "Server stopped".	*106*
void	**tooManyOpenConnections**(int nbOpen, ISocketConnection connectionRefused) Logs the event of refusing an incoming connection request due to too many open connections.	*106*
void	**unexpectedError**(Throwable e) Prints the stack trace of the Throwable e to the standard error stream.	*106*

Field Detail

FIELD_SEP

```
public static final String FIELD_SEP
```

Field separator character ("Pipe" character: "|").

Constructor Detail

DefaultLogger

```
public DefaultLogger(PrintStream out)
```

Creates a new logger. Logs will be sent to the given PrintStream

Parameters:

out - the print stream to use for logging

Method Detail

connectionClosed

```
public void connectionClosed(ISocketConnection c)
```

Logs the "Connection closed" message.

Specified by:
`connectionClosed` in interface `Logger`

Parameters:
`c` - the `ISocketConnection` to get the hash code to log

See Also:
`dumpConnectionEvent(ISocketConnection, String)`

---

connectionLost

```
public void connectionLost(ISocketConnection c,
                           IOException e)
```

Logs the "Connection lost([reason])" message.

Specified by:
`connectionLost` in interface `Logger`

Parameters:
`c` - the `ISocketConnection` to get the hash code to log
`e` - the `IOException` to get the reason of why the connection has been lost.

See Also:
`dumpConnectionEvent(ISocketConnection, String)`

---

dumpEvent

```
public void dumpEvent(String message)
```

Logs an event. The log entry is in the following form:
`[date] | [name of current thread] | message`

Parameters:
`message` - the message to log

---

httpError

```
public void httpError(ISocketConnection c,
                      String status,
                      String message)
```

Logs the `status` and `message`.

Specified by:
`httpError` in interface `Logger`
Parameters:
`c` - the `ISocketConnection` to get the hash code to log
`status` - textual status
`message` - optional textual message (could be `null`)
See Also:
`dumpConnectionEvent(ISocketConnection, String)`

---

newConnection

```
public void newConnection(ISocketConnection c)
```

Displays the message "New connection from [remote IP address]" with the hash code of the `ISocketConnection` instance `c`.

Specified by:
`newConnection` in interface `Logger`
Parameters:
`c` - the `ISocketConnection` to get the remote IP address
See Also:
`dumpConnectionEvent(ISocketConnection, String)`

---

processConnection

```
public void processConnection(ISocketConnection c)
```

Displays the message "Process connection" with the hash code of the `ISocketConnection` instance `c`.

Specified by:
`processConnection` in interface `Logger`
Parameters:
`c` - the `ISocketConnection`
See Also:
`dumpConnectionEvent(ISocketConnection, String)`

---

serverStarted

```
public void serverStarted()
```

Displays the message "Server started".

Specified by:
`serverStarted` in interface `Logger`
See Also:
`dumpEvent(String)`

---

serverStopped

```
public void serverStopped()
```

Displays the message "Server stopped".

Specified by:
`serverStopped` in interface `Logger`
See Also:
`dumpEvent(String)`

---

tooManyOpenConnections

```
public void tooManyOpenConnections(int nbOpen,
                                   ISocketConnection connectionRefused)
```

Logs the event of refusing an incoming connection request due to too many open connections.

Specified by:
`tooManyOpenConnections` in interface `Logger`
Parameters:
`nbOpen` - the maximum number of open connections
`connectionRefused` - the refused `ISocketConnection`
See Also:
`dumpEvent(String)`

---

unexpectedError

```
public void unexpectedError(Throwable e)
```

Prints the stack trace of the `Throwable` `e` to the standard error stream.

Specified by:
`unexpectedError` in interface `Logger`
Parameters:
`e` - the `Throwable` to log
See Also:
`Throwable.printStackTrace()`

Class NullLogger

com.is2t.server.log.impl

```
java.lang.Object
  └ com.is2t.server.log.impl.NullLogger
```

All Implemented Interfaces:

Logger

```
public class NullLogger
extends Object
implements Logger
```

The Null logger implements the `Logger` interface but does not log anything. When no logging is required this logger can be used.

Method Summary		Page
void	**connectionClosed**(ISocketConnection connection) Empty implementation (does not log anything).	*108*
void	**connectionLost**(ISocketConnection connection, IOException e) Empty implementation (does not log anything).	*108*
static Logger	**getInstance**() Returns the single instance of `NullLogger`.	*108*
void	**httpError**(ISocketConnection connection, String status, String message) Empty implementation (does not log anything).	*109*
void	**newConnection**(ISocketConnection streamConnection) Empty implementation (does not log anything).	*109*
void	**processConnection**(ISocketConnection streamConnection) Empty implementation (does not log anything).	*109*
void	**serverStarted**() Empty implementation (does not log anything).	*109*

void	**serverStopped**() Empty implementation (does not log anything).	*110*
void	**tooManyOpenConnections**(int nbOpen, ISocketConnection connectionRefused) Empty implementation (does not log anything).	*110*
void	**unexpectedError**(Throwable e) Empty implementation (does not log anything).	*110*

Method Detail

getInstance

```
public static Logger getInstance()
```

Returns the single instance of NullLogger.

Returns:
the single instance of NullLogger

---

connectionClosed

```
public void connectionClosed(ISocketConnection connection)
```

Empty implementation (does not log anything).

Specified by:
connectionClosed in interface Logger
Parameters:
connection - the ISocketConnection

---

connectionLost

```
public void connectionLost(ISocketConnection connection,
                           IOException e)
```

Empty implementation (does not log anything).

Specified by:
connectionLost in interface Logger

Parameters:
`connection` - the `ISocketConnection`
`e` - the `IOException`

---

httpError

```
public void httpError(ISocketConnection connection,
                      String status,
                      String message)
```

Empty implementation (does not log anything).

Specified by:
`httpError` in interface `Logger`
Parameters:
`connection` - the `ISocketConnection`
`status` - the status
`message` - the message

---

newConnection

```
public void newConnection(ISocketConnection streamConnection)
```

Empty implementation (does not log anything).

Specified by:
`newConnection` in interface `Logger`
Parameters:
`streamConnection` - the `ISocketConnection`

---

processConnection

```
public void processConnection(ISocketConnection streamConnection)
```

Empty implementation (does not log anything).

Specified by:
`processConnection` in interface `Logger`
Parameters:
`streamConnection` - the `ISocketConnection`

---

serverStarted

```
public void serverStarted()
```

Empty implementation (does not log anything).

Specified by:
`serverStarted` in interface `Logger`

---

serverStopped

```
public void serverStopped()
```

Empty implementation (does not log anything).

Specified by:
`serverStopped` in interface `Logger`

---

tooManyOpenConnections

```
public void tooManyOpenConnections(int nbOpen,
                                   ISocketConnection connectionRefused)
```

Empty implementation (does not log anything).

Specified by:
`tooManyOpenConnections` in interface `Logger`
Parameters:
`nbOpen` - the maximum number of connections
`connectionRefused` - `ISocketConnection` the refused connection

---

unexpectedError

```
public void unexpectedError(Throwable e)
```

Empty implementation (does not log anything).

Specified by:
`unexpectedError` in interface `Logger`
Parameters:
`e` - the `IOException`

Package com.is2t.server.tcp

Class Summary		*Page*
TCPServer	Abstract TCP/IP server.	*111*

Class TCPServer

com.is2t.server.tcp

```
java.lang.Object
  └ com.is2t.server.ip.Server
      └ com.is2t.server.tcp.TCPServer
```

Direct Known Subclasses:

HTTPServer

```
abstract public class TCPServer
extends Server
```

Abstract TCP/IP server.

Constructor Summary	Page
TCPServer(IServerSocketConnection connection) Constructs a new instance of TCPServer with IServerSocketConnection as the underlying connection.	*112*

Method Summary		Page
boolean	**isStopped**() Returns true if the TCPServer is stopped.	*113*
void	**start**() Starts the TCPServer.	*113*
void	**stop**() Stops the TCPServer and closes the connection.	*113*

Methods inherited from class com.is2t.server.ip.Server
getLogger, setLogger

Constructor Detail

TCPServer

```
public TCPServer(IServerSocketConnection connection)
```

Constructs a new instance of `TCPServer` with `IServerSocketConnection` as the underlying connection.

Parameters:

`connection` - the `IServerSocketConnection`

Method Detail

start

```
public void start()
```

Starts the `TCPServer`. The `TCPServer` can be started only once. Calling this method while the `TCPServer` is already running causes a `RuntimeException`.

---

stop

```
public void stop()
```

Stops the `TCPServer` and closes the connection.

---

isStopped

```
public boolean isStopped()
```

Returns `true` if the `TCPServer` is stopped.

Returns:

`true` if the `TCPServer` is stopped, `false` otherwise

Headquarters
11, rue du chemin Rouge
44373 Nantes Cedex 3
FRANCE
Phone: +33 2 40 18 04 96
www.is2t.com