/Net/samples/HTTPFormServer/src/HTTPFormServer.cpp

https://github.com/mkrivos/poco

C++ | 304 lines | 230 code | 41 blank | 33 comment | 10 complexity | 90f9575f4e26b0fa730f0c72ee79dd65 MD5 | raw file

```
1  //
2  // HTTPFormServer.cpp
3  //
4  // This sample demonstrates the HTTPServer and HTMLForm classes.
5  //
6  // Copyright (c) 2006, Applied Informatics Software Engineering GmbH.
7  // and Contributors.
8  //
9  // SPDX-License-Identifier:        BSL-1.0
10 //
11
12
13 #include "Poco/Net/HTTPServer.h"
14 #include "Poco/Net/HTTPRequestHandler.h"
15 #include "Poco/Net/HTTPRequestHandlerFactory.h"
16 #include "Poco/Net/HTTPServerParams.h"
17 #include "Poco/Net/HTTPServerRequest.h"
18 #include "Poco/Net/HTTPServerResponse.h"
19 #include "Poco/Net/HTTPServerParams.h"
20 #include "Poco/Net/HTMLForm.h"
21 #include "Poco/Net/PartHandler.h"
22 #include "Poco/Net/MessageHeader.h"
23 #include "Poco/Net/ServerSocket.h"
24 #include "Poco/CountingStream.h"
25 #include "Poco/NullStream.h"
26 #include "Poco/StreamCopier.h"
27 #include "Poco/Exception.h"
28 #include "Poco/Util/ServerApplication.h"
29 #include "Poco/Util/Option.h"
30 #include "Poco/Util/OptionSet.h"
31 #include "Poco/Util/HelpFormatter.h"
32 #include <iostream>
33
34
35 using Poco::Net::ServerSocket;
36 using Poco::Net::HTTPRequestHandler;
37 using Poco::Net::HTTPRequestHandlerFactory;
38 using Poco::Net::HTTPServer;
39 using Poco::Net::HTTPServerRequest;
40 using Poco::Net::HTTPServerResponse;
41 using Poco::Net::HTTPServerParams;
42 using Poco::Net::MessageHeader;
43 using Poco::Net::HTMLForm;
44 using Poco::Net::NameValueCollection;
45 using Poco::Util::ServerApplication;
46 using Poco::Util::Application;
47 using Poco::Util::Option;
48 using Poco::Util::OptionSet;
49 using Poco::Util::HelpFormatter;
50 using Poco::CountingInputStream;
51 using Poco::NullOutputStream;
```

```
52 using Poco::StreamCopier;
53
54
55 class MyPartHandler: public Poco::Net::PartHandler
56 {
57 public:
58    MyPartHandler():
59            _length(0)
60    {
61    }
62
63    void handlePart(const MessageHeader& header, std::istream& stream)
64    {
65            _type = header.get("Content-Type", "(unspecified)");
66            if (header.has("Content-Disposition"))
67            {
68                    std::string disp;
69                    NameValueCollection params;
70                    MessageHeader::splitParameters(header["Content-Disposition"], disp, params);
71                    _name = params.get("name", "(unnamed)");
72                    _fileName = params.get("filename", "(unnamed)");
73            }
74
75            CountingInputStream istr(stream);
76            NullOutputStream ostr;
77            StreamCopier::copyStream(istr, ostr);
78            _length = istr.chars();
79    }
80
81    int length() const
82    {
83            return _length;
84    }
85
86    const std::string& name() const
87    {
88            return _name;
89    }
90
91    const std::string& fileName() const
92    {
93            return _fileName;
94    }
95
96    const std::string& contentType() const
97    {
98            return _type;
99    }
100
101 private:
102    int _length;
103    std::string _type;
104    std::string _name;
105    std::string _fileName;
106 };
107
108
109 class FormRequestHandler: public HTTPRequestHandler
110    /// Return a HTML document with the current date and time.
111 {
112 public:
113    FormRequestHandler()
114    {
115    }
116
117    void handleRequest(HTTPServerRequest& request, HTTPServerResponse& response)
118    {
119            Application& app = Application::instance();
120            app.logger().information("Request from " + request.clientAddress().toString());
121
122            MyPartHandler partHandler;
123            HTMLForm form(request, request.stream(), partHandler);
124
125            response.setChunkedTransferEncoding(true);
126            response.setContentType("text/html");
127
```

```
128             std::ostream& ostr = response.send();
129
130             ostr <<
131                     "<html>\n"
132                     "<head>\n"
133                     "<title>POCO Form Server Sample</title>\n"
134                     "</head>\n"
135                     "<body>\n"
136                     "<h1>POCO Form Server Sample</h1>\n"
137                     "<h2>GET Form</h2>\n"
138                     "<form method=\"GET\" action=\"/form\">\n"
139                     "<input type=\"text\" name=\"text\" size=\"31\">\n"
140                     "<input type=\"submit\" value=\"GET\">\n"
141                     "</form>\n"
142                     "<h2>POST Form</h2>\n"
143                     "<form method=\"POST\" action=\"/form\">\n"
144                     "<input type=\"text\" name=\"text\" size=\"31\">\n"
145                     "<input type=\"submit\" value=\"POST\">\n"
146                     "</form>\n"
147                     "<h2>File Upload</h2>\n"
148                     "<form method=\"POST\" action=\"/form\" enctype=\"multipart/form-data\">\n"
149                     "<input type=\"file\" name=\"file\" size=\"31\"> \n"
150                     "<input type=\"submit\" value=\"Upload\">\n"
151                     "</form>\n";
152
153             ostr << "<h2>Request</h2><p>\n";
154             ostr << "Method: " << request.getMethod() << "<br>\n";
155             ostr << "URI: " << request.getURI() << "<br>\n";
156             NameValueCollection::ConstIterator it = request.begin();
157             NameValueCollection::ConstIterator end = request.end();
158             for (; it != end; ++it)
159             {
160                     ostr << it->first << ": " << it->second << "<br>\n";
161             }
162             ostr << "</p>";
163
164             if (!form.empty())
165             {
166                     ostr << "<h2>Form</h2><p>\n";
167                     it = form.begin();
168                     end = form.end();
169                     for (; it != end; ++it)
170                     {
171                             ostr << it->first << ": " << it->second << "<br>\n";
172                     }
173                     ostr << "</p>";
174             }
175
176             if (!partHandler.name().empty())
177             {
178                     ostr << "<h2>Upload</h2><p>\n";
179                     ostr << "Name: " << partHandler.name() << "<br>\n";
180                     ostr << "File Name: " << partHandler.fileName() << "<br>\n";
181                     ostr << "Type: " << partHandler.contentType() << "<br>\n";
182                     ostr << "Size: " << partHandler.length() << "<br>\n";
183                     ostr << "</p>";
184             }
185             ostr << "</body>\n";
186     }
187 };
188
189
190 class FormRequestHandlerFactory: public HTTPRequestHandlerFactory
191 {
192 public:
193     FormRequestHandlerFactory()
194     {
195     }
196
197     HTTPRequestHandler* createRequestHandler(const HTTPServerRequest& request)
198     {
199             return new FormRequestHandler;
200     }
201 };
202
203
```

```
204 class HTTPFormServer: public Poco::Util::ServerApplication
205     /// The main application class.
206     ///
207     /// This class handles command-line arguments and
208     /// configuration files.
209     /// Start the HTTPFormServer executable with the help
210     /// option (/help on Windows, --help on Unix) for
211     /// the available command line options.
212     ///
213     /// To use the sample configuration file (HTTPFormServer.properties),
214     /// copy the file to the directory where the HTTPFormServer executable
215     /// resides. If you start the debug version of the HTTPFormServer
216     /// (HTTPFormServerd[.exe]), you must also create a copy of the configuration
217     /// file named HTTPFormServerd.properties. In the configuration file, you
218     /// can specify the port on which the server is listening (default
219     /// 9980) and the format of the date/Form string sent back to the client.
220     ///
221     /// To test the FormServer you can use any web browser (http://localhost:9980/).
222 {
223 public:
224     HTTPFormServer(): _helpRequested(false)
225     {
226     }
227
228     ~HTTPFormServer()
229     {
230     }
231
232 protected:
233     void initialize(Application& self)
234     {
235             loadConfiguration(); // load default configuration files, if present
236             ServerApplication::initialize(self);
237     }
238
239     void uninitialize()
240     {
241             ServerApplication::uninitialize();
242     }
243
244     void defineOptions(OptionSet& options)
245     {
246             ServerApplication::defineOptions(options);
247
248             options.addOption(
249                     Option("help", "h", "display help information on command line arguments")
250                             .required(false)
251                             .repeatable(false));
252     }
253
254     void handleOption(const std::string& name, const std::string& value)
255     {
256             ServerApplication::handleOption(name, value);
257
258             if (name == "help")
259                     _helpRequested = true;
260     }
261
262     void displayHelp()
263     {
264             HelpFormatter helpFormatter(options());
265             helpFormatter.setCommand(commandName());
266             helpFormatter.setUsage("OPTIONS");
267             helpFormatter.setHeader("A web server that shows how to work with HTML forms.");
268             helpFormatter.format(std::cout);
269     }
270
271     int main(const std::vector<std::string>& args)
272     {
273             if (_helpRequested)
274             {
275                     displayHelp();
276             }
277             else
278             {
279                     unsigned short port = (unsigned short) config().getInt("HTTPFormServer.port", 9980);
```

```
280
281                     // set-up a server socket
282                     ServerSocket svs(port);
283                     // set-up a HTTPServer instance
284                     HTTPServer srv(new FormRequestHandlerFactory, svs, new HTTPServerParams);
285                     // start the HTTPServer
286                     srv.start();
287                     // wait for CTRL-C or kill
288                     waitForTerminationRequest();
289                     // Stop the HTTPServer
290                     srv.stop();
291             }
292             return Application::EXIT_OK;
293     }
294
295 private:
296     bool _helpRequested;
297 };
298
299
300 int main(int argc, char** argv)
301 {
302     HTTPFormServer app;
303     return app.run(argc, argv);
304 }
```