Contents

0.1 $_{c}all.cpp$

```
#include "__call.h"
#include "Exception.h"
#include "MString.h"
#include <ostream>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <filesystem>

namespace jet {

   __call::__call(coreutils::ZString &in, coreutils::MString &parentOut, Global &
       global, Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, local) {
      if(hasContainer && keywordDefined("input"))
        throw coreutils::Exception("call␣tag␣cannot␣have␣both␣input␣keyword␣and␣have␣
            a␣container.");
      if(!keywordDefined("pgm"))
        throw coreutils::Exception("pgm␣keyword␣must␣be␣specified.");
      for(ix = 0; ix <= 50; ++ix)
        argv[ix] = NULL;
      argv[0] = resolveKeyword("pgm").c_str(); // TODO: Need to peel off the program
          name only and pass as argv[0].
      for(ix = 1; ix <= 50; ++ix) {
         coreutils::MString arg("arg");
         arg << ix;
         if(keywordDefined(arg)) {
           argv[ix] = resolveKeyword(arg).c_str();
         } else
           break;
      }
      pipe(fdo);
      pid = fork();
      if(pid == 0) {
         close(fdo[0]);
         dup2(fdo[1], 1);
         if(hasContainer) {
            pipe(fdi);
            if(fork() == 0) {
               close(fdi[0]);
               write(fdi[1], container.getData(), container.getLength());
               close(fdi[1]);
               exit(0);
            }
            close(fdi[1]);
            dup2(fdi[0], 0);
         } else if(keywordDefined("input")) {
            coreutils::MString input(resolveKeyword("input"));
            pipe(fdi);
            if(fork() == 0) {
               close(fdi[0]);
               write(fdi[1], input.getData(), input.getLength());
               close(fdi[1]);
```

```
            exit(0);
         }
         close(fdi[1]);
         dup2(fdi[0], 0);
      }
      rc = execvpe(resolveKeyword("pgm").c_str(), argv, global.envp);
      close(fdo[1]);
      exit(errno);
   }
   close(fdo[1]);
   if(keywordDefined("name")) {
      coreutils::MString value;
      value.read(fdo[0]);
      storeVariable(resolveKeyword("name"), value, resolveKeyword("scope"));
   } else
     out.read(fdo[0]);
   waitpid(pid, &status, 0);
   if(keywordDefined("error")) {
      global.variables[resolveKeyword("error")] = (status >> 8 & 255);
   }
}

}
```

0.2 $_{c}all.h$

```
#ifndef ____call_h__
#define ____call_h__

#include "Tag.h"

namespace jet {

   class __call : public Tag {

    public:
      __call(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
          Tag *parent, Tag *local);

    private:
      int pid;
      int status;
      int ix;
      int fdi[2];
      int fdo[2];
      int rc;
      char *argv[50];

   };

}

#endif
```

0.3 _comment.cpp

```
#include "__comment.h"
#include "Exception.h"

namespace jet {

   __comment::__comment(coreutils::ZString &in, coreutils::MString &parentOut, Global
        &global, Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, this)
       {
      if(!hasContainer)
        throw coreutils::Exception("comment must have a container.");
      output = false;
   }

}
```

0.4 $_{c}omment.h$

```
#ifndef ____comment_h__
#define ____comment_h__

#include "Tag.h"

namespace jet {

   class __comment : public Tag {

    public:
      __comment(coreutils::ZString &in, coreutils::MString &parentOut, Global &global
          , Tag *parent, Tag *local);

   };

}

#endif
```

0.5 $_{d}otag.cpp$

```
#include "__dotag.h"
#include "Exception.h"

namespace jet {

   __dotag::__dotag(coreutils::ZString &in, coreutils::MString &parentOut, Global &
       global, Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, local) {
      coreutils::ZString container3 = global.tags[name];
      processContainer(container3, container);
   }
}
```

0.6 $_{d}otag.h$

```
#ifndef ____dotag_h__
#define ____dotag_h__

#include "Tag.h"
#include "ZString.h"

namespace jet {

   class __dotag : public Tag {

    public:
      __dotag(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
          Tag *parent, Tag *local);

   };

}

#endif
```

0.7 _dump.cpp

```
#include "__dump.h"
#include "Exception.h"
#include <iostream>
#include <fstream>

namespace jet {

   __dump::__dump(coreutils::ZString &in, coreutils::MString &parentOut, Global &
       global, Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, local) {
      if(!keywordDefined("file"))
        throw coreutils::Exception("file must be sppecified for dump tag.");

      std::ofstream outFile(keywords["file"].str());

      if(global.cgi) {
         outFile << "*** CGI VARIABLES ***" << std::endl;
         for (auto i = global.cgiVariables.begin(); i != global.cgiVariables.end(); i
             ++)
           outFile << i->first << "=[" << i->second << "]" << std::endl;
      }

      outFile << "*** GLOBAL VARIABLES ***" << std::endl;
      for (auto i = global.variables.begin(); i != global.variables.end(); i++)
        outFile << i->first << "=[" << i->second << "]" << std::endl;

      outFile << "*** LOCAL VARIABLES ***" << std::endl;
      for (auto i = local->variables.begin(); i != local->variables.end(); i++)
        outFile << i->first << "=[" << i->second << "]" << std::endl;

      outFile << "*** KEYWORD VALUES ***" << std::endl;
      for (auto i = parent->keywords.begin(); i != parent->keywords.end(); i++)
        outFile << i->first << "=[" << i->second << "]" << std::endl;

      outFile << "*** COOKIES ***" << std::endl;
      for (auto i = global.cookies.data.begin(); i != global.cookies.data.end(); i++)
        outFile << i->first << "=[" << i->second << "]" << std::endl;

      outFile.close();

   }

}
```

0.8 $_{d}ump.h$

```
#ifndef ____dump_h__
#define ____dump_h__

#include "Tag.h"
#include "MString.h"
#include "Global.h"

namespace jet {

   class __dump : public Tag {

    public:
      __dump(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
          Tag *parent, Tag *local);

   };

}

#endif
```

0.9 for.cpp

```
#include "__for.h"
#include "Exception.h"
#include <iostream>

namespace jet {

   __for::__for(coreutils::ZString &in, coreutils::MString &parentOut, Global &global
       , Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, this) {
      double counter = 0.0f;
      bool nameDefined = keywordDefined("name");
      if(keywordDefined("start")) {
         counter = resolveKeyword("start").asDouble();
         keywords["start"].reset();
      }
      if(!keywordDefined("end"))
        throw coreutils::Exception("for␣tag␣requires␣end␣keyword.");
      if(!keywordDefined("step"))
        throw coreutils::Exception("for␣tag␣requires␣step␣keyword.");
      for(double ix = counter; ix <= resolveKeyword("end").asDouble(); ix +=
          resolveKeyword("step").asDouble()) {
         resolveKeyword("end").reset();
         resolveKeyword("step").reset();
         if(nameDefined) {
            if(!keywordDefined("scope") || (resolveKeyword("scope") == "global"))
              global.variables[resolveKeyword("name")] = ix;
            else if(resolveKeyword("scope") == "local")
              this->local->variables[resolveKeyword("name")] = ix;
            else if(resolveKeyword("scope") == "parent")
              parent->local->variables[resolveKeyword("name")] = ix;
            else
              throw coreutils::Exception("scope␣value␣is␣not␣valid.");
         }
         processContainer(container);
         container.reset();
      }
   }

}
```

0.10 $_{for.cpp}$

```
#include "__for.h"
#include "Exception.h"
#include <iostream>

namespace jet {

   __for::__for(coreutils::ZString &in, coreutils::MString &parentOut, Global &global
       , Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, this) {
      double counter = 0.0f;
      bool nameDefined = keywordDefined("name");
      if(keywordDefined("start")) {
         counter = resolveKeyword("start").asDouble();
         keywords["start"].reset();
      }
      if(!keywordDefined("end"))
        throw coreutils::Exception("for␣tag␣requires␣end␣keyword.");
      if(!keywordDefined("step"))
        throw coreutils::Exception("for␣tag␣requires␣step␣keyword.");
      for(double ix = counter; ix <= resolveKeyword("end").asDouble(); ix +=
          resolveKeyword("step").asDouble()) {
         resolveKeyword("end").reset();
         resolveKeyword("step").reset();
         if(nameDefined) {
            if(!keywordDefined("scope") || (resolveKeyword("scope") == "global"))
              global.variables[resolveKeyword("name")] = ix;
            else if(resolveKeyword("scope") == "local")
              this->local->variables[resolveKeyword("name")] = ix;
            else if(resolveKeyword("scope") == "parent")
              parent->local->variables[resolveKeyword("name")] = ix;
            else
              throw coreutils::Exception("scope␣value␣is␣not␣valid.");
         }
         processContainer(container);
         container.reset();
      }
   }

}
```

0.11 for.h

```
#ifndef ____for_h__
#define ____for_h__

#include "Tag.h"
#include <sstream>

namespace jet {

   class __for : public Tag {

    public:
      __for(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
          Tag *parent, Tag *local);

   };

}

#endif
```

0.12 Global.cpp

```
#include "Global.h"
#include "Exception.h"
#include "__mysql.h"
#include <iostream>
#include <stdlib.h>

namespace jet {

   Global::Global(char **envp) : envp(envp) {

   }

   Global::~Global() {

   }

   void Global::dump() {
      for (auto i = variables.begin(); i != variables.end(); i++)
        std::cout << i->first << "=[" << i->second << "]" << std::endl;
   }

   bool Global::sessionExists(coreutils::MString sessionId) {
      return sessions.find(sessionId) != sessions.end();
   }

   void Global::addSession(coreutils::MString sessionId, __mysql *mysql) {
      if(sessionExists(sessionId))
        coreutils::Exception("sessionid already exists.");
      sessions[sessionId] = mysql;
   }

   void Global::removeSession(coreutils::MString sessionId) {
      sessions.erase(sessionId);
   }

   __mysql * Global::getSession(coreutils::MString sessionId) {
      if(sessions.find(sessionId) == sessions.end())
        throw coreutils::Exception("requested session is not available.");
      return sessions[sessionId];
   }

   coreutils::ZString Global::getSessionVariable(coreutils::MString &splitName) {
      if(sessions.find(splitName[0]) == sessions.end())
        throw coreutils::Exception("requested session is not available in variable.")
            ;
      return sessions[splitName[0]]->getColumnValue(splitName[1]);
   }

   void Global::outputHeaders() {
      if(headers.size() > 0) {
         for(auto header = headers.begin();
             header != headers.end();
             ++header) {
            std::cout << header->first << ": " << header->second << std::endl;
```

```
      }
      std::cout << std::endl;
   }
}

void Global::setupFormData(coreutils::ZString &formdata) {
   coreutils::ZString boundary = formdata.goeol();
   while(!formdata.eod()) {
      if(formdata.ifNext("Content-Disposition:␣form-data;")) {
         formdata.skipWhitespace();
         if(formdata.ifNext("name=\"")) {
            coreutils::ZString name = formdata.getTokenInclude("
                ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789._-"
                );
            if(formdata.ifNext("\"")) {
               formdata.goeol();
               formdata.goeol();
               coreutils::ZString data = formdata.getTokenExclude("-"); // TODO:
                   Fix this parsing. Need a string exclusion method to check for '
                   boundary'.
               data.trimCRLF();
               formdata.ifNext(boundary);
               int index = 0;
               coreutils::MString namex;
               do {
                  namex = "";
                  namex << name << ":" << index++;
               } while(cgiVariables.count(namex) != 0);
               cgiVariables[namex] = data;
               if(formdata.ifNext("--"))
                 break;
               formdata.goeol();
            }
            else
              throw coreutils::Exception("expecting␣closing␣double␣quote␣on␣
                  variable␣name␣in␣received␣CGI␣data.");
         } else
           throw coreutils::Exception("expecting␣name␣subfield␣in␣received␣CGI␣
               data.");
      } else
        throw coreutils::Exception("expecting␣Content-Disposition␣header␣in␣
            received␣CGI␣data.");
   }
}

void Global::setupFormURLEncoded(coreutils::ZString &formdata) {
   while(!formdata.eod()) {
      coreutils::ZString name = formdata.getTokenInclude("
          ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789._-");
      if(formdata.ifNext("=")) {
         coreutils::MString data = formdata.getTokenExclude("&");
         formdata.ifNext("&");
         int index = 0;
         coreutils::MString namex;
         do {
            namex = "";
            namex << name << ":" << index++;
```

```
            } while(cgiVariables.count(namex) != 0);
            cgiVariables[namex] = data.fromCGI();
          } else
            throw coreutils::Exception("expecting␣=␣after␣name␣in␣received␣CGI␣data.")
                ;
      }
   }

}
```

0.13 Global.h

```
#ifndef __Global_h__
# define __Global_h__

# include "MString.h"
# include "CGIFormattedData.h"
# include <map>

namespace  jet {

   class __mysql;

   class Global {

    public:
      Global(char **envp);
      virtual ~Global();

      void dump();
      bool sessionExists(coreutils::MString sessionId);
      void addSession(coreutils::MString sessionId, __mysql *mysql);
      void removeSession(coreutils::MString sessionId);
      __mysql * getSession(coreutils::MString sessionId);
      coreutils::ZString getSessionVariable(coreutils::MString &splitName);
      void outputHeaders();
      void setupFormData(coreutils::ZString &formdata);
      void setupFormURLEncoded(coreutils::ZString &formdata);
      char *errorCursor = NULL;

      coreutils::CGIFormattedData cookies;
      std::map<coreutils::MString, coreutils::MString> variables;
      std::map<coreutils::MString, coreutils::MString> cgiVariables;
      std::map<coreutils::MString, __mysql *> sessions;
      std::map<coreutils::MString, coreutils::MString> headers;
      std::map<coreutils::MString, coreutils::MString> tags;
      char **envp;
      bool cgi = false;
      bool session = false;
      coreutils::MString sessionId;

   };

}

#endif
```

0.14 $_{h}eader.cpp$

```
#include "__header.h"
#include "Exception.h"
#include "Operand.h"
#include <iostream>

namespace jet {

   __header::__header(coreutils::ZString &in, coreutils::MString &parentOut, Global &
       global, Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, local) {
      output = false;
      if(!global.cgi)
        throw coreutils::Exception("must␣enable␣cgi␣mode␣on␣jet␣tag␣to␣use␣header␣tag
            .");
      if(!keywordDefined("name"))
        throw coreutils::Exception("header␣tag␣must␣have␣name␣defined.");
      if(!keywordDefined("expr") && keywordDefined("value") && hasContainer)
        throw coreutils::Exception("header␣tag␣cannot␣have␣both␣value␣and␣a␣container
            .");
      if(keywordDefined("expr") && !keywordDefined("value") && hasContainer)
        throw coreutils::Exception("header␣tag␣cannot␣have␣both␣expr␣and␣a␣container.
            ");
      if(keywordDefined("expr") && keywordDefined("value") && !hasContainer)
        throw coreutils::Exception("header␣tag␣cannot␣have␣both␣expr␣and␣value.");
      if(!keywordDefined("expr") && !keywordDefined("value") && !hasContainer)
        throw coreutils::Exception("header␣tag␣must␣have␣a␣value,␣expr␣or␣a␣container
            .");
      if(keywordDefined("expr")) {
         if(keywordDefined("eval"))
           throw coreutils::Exception("Cannot␣use␣eval␣with␣expr.");
         global.headers[resolveKeyword("name")] = Operand(keywords["expr"], *this).
             string;
      } else if(hasContainer) {
         processContainer(container);
         if(evaluate) {
            global.headers[resolveKeyword("name")] = out;
         } else {
            global.headers[resolveKeyword("name")] = container;
         }
      } else
        global.headers[resolveKeyword("name")] = resolveKeyword("value");

   }
}
```

0.15 $_{h}eader.h$

```
#ifndef ____header_h__
#define ____header_h__

#include "Tag.h"
#include "ZString.h"
#include "MString.h"
#include <sstream>

namespace jet {

   class __header : public Tag {

    public:
      __header(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
           Tag *parent, Tag *local);

    protected:

   };

}

#endif
```

0.16 $_{if.cpp}$

```
#include "__if.h"
#include "Exception.h"
#include <iostream>
#include "Operand.h"

namespace jet {

   __if::__if(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
       Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, this, "else") {
      coreutils::MString result;
      bool booleanResult = false;
      if(keywordDefined("value1")) {
         if(keywordDefined("expr"))
           throw coreutils::Exception("Either␣value1␣or␣expr␣can␣be␣specified␣but␣not
               ␣both.");
         if(keywordDefined("value2")) {
            if(!keywordDefined("type"))
              throw coreutils::Exception("type␣expected␣if␣value1␣and␣value2␣
                  specified.");
         } else
           throw coreutils::Exception("value2␣required␣if␣value1␣specified.");
         coreutils::MString type = resolveKeyword("type");
         if((type != "eq") &&
            (type != "ne") &&
            (type != "lt") &&
            (type != "le") &&
            (type != "gt") &&
            (type != "ge"))
           throw coreutils::Exception("type␣value␣must␣be␣'eq','ne','lt','le','gt','
               ge'.");
         int rc = resolveKeyword("value1").compare(resolveKeyword("value2"));
//       std::cout << "if: " << resolveKeyword("value1") << " " << type << " " <<
    resolveKeyword("value2") << ":" << rc << std::endl;
         if(((type == "eq") && (rc == 0)) ||
            ((type == "ne") && (rc != 0)) ||
            ((type == "lt") && (rc == -1)) ||
            ((type == "le") && (rc != 1)) ||
            ((type == "gt") && (rc == 1)) ||
            ((type == "ge") && (rc != -1)))
           booleanResult = true;
         else
           booleanResult = false;
      } else if(keywordDefined("expr")) {
         if(keywordDefined("value2"))
           throw coreutils::Exception("value2␣should␣not␣be␣specified␣with␣expr.");
         if(keywordDefined("type"))
           throw coreutils::Exception("type␣should␣not␣be␣specified␣with␣expr.");
         booleanResult = Operand(keywords["expr"], *this).boolean;
      }
      if(booleanResult)
        processContainer(container);
      else if(hasContainer2)
        processContainer(container2);
   }
```

```
}
```

0.17 $_{if.h}$

```
#ifndef ____if_h__
#define ____if_h__

#include "Tag.h"
#include "ZString.h"
#include "MString.h"
#include <sstream>

namespace jet {

   class __if : public Tag {

    public:
      __if(coreutils::ZString &in, coreutils::MString &parentOut, Global &global, Tag
           *parent, Tag *local);

   };

}

#endif
```

0.18 ifrow.cpp

```
#include "__ifrow.h"
#include "Exception.h"
#include "MString.h"
#include "__mysql.h"
#include <stdlib.h>
#include <unistd.h>

namespace jet {

   __ifrow::__ifrow(coreutils::ZString &in, coreutils::MString &parentOut, Global &
       global, Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, this, "
       else") {
      output = false;
      if(!hasContainer)
        throw coreutils::Exception("ifrow␣tag␣must␣have␣a␣container.");
      if(!global.sessionExists(keywords[resolveKeyword("sessionid")]))
        throw coreutils::Exception("sessionid␣does␣not␣exist.");
      if(global.getSession(keywords[resolveKeyword("sessionid")])->hasRow())
        processContainer(container);
      else
        processContainer(container2);
   }

}
```

0.19 $_{i}frow.h$

```
#ifndef ____ifrow_h__
#define ____ifrow_h__

#include "Tag.h"

namespace jet {

   class __ifrow : public Tag {

    public:
      __ifrow(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
          Tag *parent, Tag *local);

   };

}

#endif
```

0.20 _include.cpp

```
#include "__include.h"
#include "Exception.h"
#include "File.h"

namespace jet {

   __include::__include(coreutils::ZString &in, coreutils::MString &parentOut, Global
        &global, Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, local)
        {
      if(!keywordDefined("file"))
        throw coreutils::Exception("file␣keyword␣must␣be␣specified.");
      if(hasContainer)
        throw coreutils::Exception("include␣tag␣should␣not␣have␣a␣container.");
      hasContainer = true;
      coreutils::File file(resolveKeyword("file"));
      file.read();
      container = file.asZString();
      try {
         processContainer(container);
      }
      catch(coreutils::Exception e) {
         container.setCursor(global.errorCursor);
         container.moveBackToLineStart();
         std::cout << "----------------------------" << std::endl;
         std::cout << "Error␣in␣included␣script␣'" << variables["file"] << "'␣at␣line
             ␣" << container.getLineNumberAtCursor() << std::endl;
         std::cout << "Error␣text:␣" << e.text << std::endl;
         std::cout << "----------------------------" << std::endl;
         std::cout << container.parsed() << std::endl;
         std::cout << "********␣Error␣caught:␣" << e.text << std::endl;
         std::cout << container.unparsed() << std::endl;
         global.errorCursor = in.getCursor();
         throw coreutils::Exception("error␣in␣included␣file.");
      }

   }

}
```

0.21 $_{i}nclude.h$

```
#ifndef ____include_h__
#define ____include_h__

#include "Tag.h"

namespace jet {

   class __include : public Tag {

    public:
      __include(coreutils::ZString &in, coreutils::MString &parentOut, Global &global
          , Tag *parent, Tag *local);

   };

}

#endif
```

0.22 jet-2.0.cpp

```
#include <iostream>
#include <sstream>
#include "File.h"
#include "Global.h"
#include "Exception.h"
#include "__jet.h"

int main(int argc, char **argv, char **envp) {

   coreutils::File script(argv[1]);
   script.read();
   coreutils::ZString data = script.asZString();
   data.goeol();

   jet::Global global(envp);

   try {
      coreutils::MString out;
      global.errorCursor = data.getCursor();
      jet::__jet *jet = new jet::__jet(data, out, global, NULL, NULL);
      delete jet;
      global.outputHeaders();
      std::cout << out;
   }
   catch(coreutils::Exception e) {
      data.setCursor(global.errorCursor);
      data.moveBackToLineStart();
      std::cout << "------------------------------" << std::endl;
      std::cout << "Error␣in␣jet␣script␣'" << argv[1] << "'␣at␣line␣" << data.
          getLineNumberAtCursor() << std::endl;
      std::cout << "Error␣text:␣" << e.text << std::endl;
      std::cout << "------------------------------" << std::endl;
      std::cout << data.parsed() << std::endl;
      std::cout << "********␣Error␣caught:␣" << e.text << std::endl;
      std::cout << data.unparsed() << std::endl;
      global.dump();
   }

}
```

0.23 $_{j}et.cpp$

```
#include "__jet.h"
#include "Exception.h"
#include "Global.h"
#include "SessionId.h"
#include <iostream>
#include <fstream>

namespace jet {

   __jet::__jet(coreutils::ZString &in, coreutils::MString &parentOut, Global &global
       , Tag *parent, Tag *local) : Tag(in, parentOut, global, parent, this) {
      char *cookies;
      if(keywordDefined("cgi") && (resolveKeyword("cgi") == "true")) {
         global.cgi = true;

         if(keywordDefined("sessiondir")) {
            global.session = true;
            global.cookies = getenv("HTTP_COOKIE");

//          if request_has_cookie then
//            pull sessionfile from sessiondir.
//            if last activity time is expired then ignore.
//            follow sessiontimeoutredirecturl.
//          else
            SessionId sessionId;
            global.headers["Set-Cookie"] << "session=" << sessionId;
            if(keywordDefined("sessiontimeout")) {
               time_t timeout = time(0) + keywords["sessiontimeout"].asInteger();
            }

//                also save last activity time in session file.

         }

         coreutils::ZString requestMethod(getenv("REQUEST_METHOD"));
         if(requestMethod == "POST") {
            coreutils::ZString contentLength(getenv("CONTENT_LENGTH"));
            coreutils::ZString contentType(getenv("CONTENT_TYPE"));

            if(contentType == "multipart/form-data")
               coreutils::IMFMultipart postdata(0, "");

//          else if(contentType == "application/x-www-form-urlencoded")
//            global.setupFormURLEncoded(postdata);
         }
      }
      processContainer(container, NULL, true);
   }
}
```

0.24 $_{j}et.h$

```
#ifndef ____jet_h__
#define ____jet_h__

#include "Tag.h"
#include "ZString.h"
#include "IMFRequest.h"
#include "IMFMessage.h"
#include <sstream>

namespace jet {

   class __jet : public Tag {

    public:
      __jet(coreutils::ZString &in, coreutils::MString &parentOut, Global &global,
          Tag *parent, Tag *local);

   };

}

#endif
```