CoatiFrame 1.12 *example*

User Manual

(c) 2017 Mellspa, All rights reserved

Content

Preface

This is the documentation of the Mellspa library package CoatiFrame 1.12, the C++11 HTTP-server framework and of the CoatiFrame example program.

The example program is a small game which will demonstrate some of the very basic features of the HTTP-server or HTTPS-server framework or platform. The game runs as server which uses the framework typical combination of HTML5 content, images and programmed interactions.

The game is made in the GET, POST, and Widget scheme. All these schemes are served by the CoatiFrame 1.12 framework. The differences might be observed in the source listings and, of course, by playing the game online in a browser. Though the game is always the same one may observe the differences in the browser interaction.

The HTML sources are left in an instructive state in order to remain readable – this is different to the framework. The CoatiFrame 1.12 framework organizes the data slightly different: It will pack the graphical content into binaries which will become hidden to the end user. Therefore a framework based solution remains reliable once delivered to the customer.

As the game – as any framework based program – is a HTTP-server one needs a browser to get into interaction with the program. The program itself starts therefore a browser and initiates the game as well.

Enjoy!

CoatiFrame 1.12 / Twymt++ 1.1.4 example program

The example program offers a browser game as well as an instructive code example. Possible source code reductions by inheritance of classes are avoided in order to maintain a one by one experience of code and interaction. All HTML5 and image content and the basic C++ code are delivered as source files.

The game is an implementation of the well known Tic-Tac-Toe.

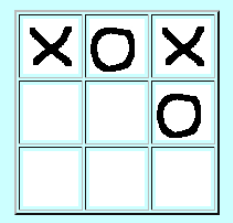

Image 1: The Tic-Tac-Toe example of the CoatiFrame 1.12 / Twymt++ 1.1 framework.

Prerequisites

- Linux 32bit (ARM) or 64-bit (x86) (standard libraries, glibc, glibc++, X11)
- Windows (Visual C++ redistributable packages for Visual Studio 2015)
- Android-x86
- A WEB-browser (probably any that supports JavaScript)

Directory structure

Linux and Windows

The directory structure looks as follows:

`Spiel1.12/`

`Spiel1.12/bin`	`(Linux) XTheGame / (Windows) XTheGame.exe`, the program
`Spiel1.12/documentation`	this documentation
`Spiel1.12/src`	`spiel.h`, `spiel.cpp`, the basic source files
`Spiel1.12/html`	All HTML5 pages and image content used by the game.

Android-x86

apk-package	coatigame.apk

How to run the game

The most simple way is just click on the program `XTheGame` which is to be found in the `Spiel`1.12 example `bin` directory.

The program opens a browser and connect it to the game on the right TCP/IP port. The game will use a free port number (it gets the number from the operating system).

Once started follow the instructions shown on it's HTML5-page.

How it works

The GET scheme

The basic framework related part of the program is to be found in `spiel.h` and `spiel.cpp`. Both files are listed.

What does it mean: “GET scheme”? Everybody knows these underlined blue links in a HTML page. Normally just a click on it links to an other page which will be loaded and shown. If you will investigate the HTML code of page you will find HTML elements like `<a href=”your_link_destination”>this is your link</a>`. Technically a click on the blue underlined text “this is your link” will send an HTTP GET to the server named by `”your_link_destination”` and the response of it is your new page.

Doing that simple scheme in an interactive document like our game you might observe that some browsers are doing progressive scans of links found in a page. Just to accelerate the loading of the next page. But this would play our game – just played by the browser.

To avoid this, the page is using some JavaScript-events (`onclick`) which initiates the reloading of the new page just by assigning the new address to `window.location.href = URI+...` . Try to localize this in spiele.html.

A disadvantage of that solution is that the URL – always shown by the browser – might be 'hijacked' and misused. Though quite a lot of information is shown there this scheme might be instructive, but it is not really that safe as it could be.

Observe the code example for the GET scheme in the source code section.

The POST scheme

The POST scheme is naturally some type of form based communication. Somebody is filling out some inputs, check-boxes etc. and is finally clicking some button like 'Send' or 'Ok'. This is the event when all content of the named fields is collected and packed into a URL coded question (GET again), or into some message block which is sent to the server (POST).
The content of the message block can be encrypted. But at least it is not shown as an URL on the top line of the browser. The POST scheme is therefore something safer and might contain quite a lot of additional structured information.
These forms can be programmed in the HTML hidden from the user and 'clicked' by JavaScript events (like `onclick, onchange`, etc.).
The functions `_connect_element_method_POST("fxx", onclick);` are introducing some hidden forms and connect them by the event `onclick` to the images. The Coatiframe 1.12 framework is doing the whole work in the background and connects the click the command handler functions. These

command handlers are the same as for the GET scheme. As in the GET scheme the browser always reloads the whole page as a reaction of a form request in the POST scheme.

Observe the source code for the POST-scheme version in the sources section.

The Widget scheme

The GET and the POST scheme in the CoatiFrame 1.12 framework are used for schemes where the browser is always reloading *the whole* page on one HTTP request. The Widget scheme stands for browser background request, that normally expects some data that might be evaluated and *modify parts of the page currently shown*. In case of the CoatiFrame 1.12 framework the response is a JavaScript block which is evaluated and modifies the currently shown page. The commands used in the handler functions modifies the local objects as well as support the JavaScript code to modify the remote shown content.

The sources

The cpp header

```
/**************************************************************************
*
*
*   "CoatiFrame" Release 1.12
*
*   Portions Copyright (c) 2014-2017
*   by Mellspa.
*   All rights reserved.
*   Copyrighted as an unpublished work.
*
**************************************************************************/

// this include file is needed for the framework
#include <mstwymtbase.h>

// defines the logic of the game
#include <dreikreuze.h>


//! The definition of the context class of the GET scheme which will be connected to the MSCoatiApp.
class MSHTMLKontext_Game_Get : public MSHTMLKontextN
{
protected:

  // the game representation as a class member
  MSDreikreuze dreikreuze;

  // two function to interact with the game
  void KommandoFeld(string feld);
  void checkwin();


  // the class handler functions
  void CommandF11(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF12(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF13(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF21(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF22(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF23(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF31(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF32(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF33(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);

  void CommandRunWidgetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
```

```
htmlcommand);

private:
  void CommandNeuStarten(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandGame(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandRunPostScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);

public:

  //! contructor
  MSHTMLKontext_Game_Get(unsigned int stack_level, map<string,string>& extended_params);

  //! destructor
  virtual ~MSHTMLKontext_Game_Get();

};


//! The definition of the context class of the POST scheme which will be connected to the
MSCoatiApp.
//! most parts are equally the GET scheme
class MSHTMLKontext_Game_Post : public MSHTMLKontextN
{
protected:

  // the game representation as a class member
  MSDreikreuze dreikreuze;

  // two function to interact with the game
  void KommandoFeld(string feld);
  void checkwin();


  // the class handler functions
  void CommandF11(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF12(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF13(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF21(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF22(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF23(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF31(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF32(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandF33(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);

  void CommandNeuStarten(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandGame(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandRunGetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandRunWidgetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand);


public:

  //! contructor
  MSHTMLKontext_Game_Post(unsigned int stack_level, map<string,string>& extended_params);

  //! destructor
  virtual ~MSHTMLKontext_Game_Post();

};



//! The definition of the context class of the Widget scheme which will be connected to the
MSCoatiApp.
class MSHTMLKontext_GameWidget : public MSHTMLKontextN
{
protected:

  // the game representation as a class member
  MSDreikreuze dreikreuze;



  // the class handler function
```

```
  void CommandGame(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandRunGetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);
  void CommandRunPostScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string htmlcommand);

public:

  // constructor an destructor are formally needed but does not define any extra content for the
game.

  //! contructor
  MSHTMLKontext_GameWidget(unsigned int stack_level, map<string,string>& extended_params);

  //! destructor
  virtual ~MSHTMLKontext_GameWidget();

};
```

The cpp source

```
/*****************************************************************************
*
*
*   "CoatiFrame" Release 1.12
*
*   Portions Copyright (c) 2014-2017
*   by Mellspa.
*   All rights reserved.
*   Copyrighted as an unpublished work.
*
*****************************************************************************/
#include <stdafx.h>

#include <spiel.h>

// The definitions for the GET scheme

MSHTMLKontext_Game_Get::MSHTMLKontext_Game_Get(unsigned int stack_level, map<string,string>&
extended_params)
: MSHTMLKontextN(stack_level,"game_get.html?", extended_params, nullptr) //The ? marks the GET
scheme.
{


  // now loading the game view from the HTML5, store it in thedocumentdisplayed
  _loadDocumentTemplate("spiele.html");


  // declare all page handlers
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF11,"/f11");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF12,"/f12");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF13,"/f13");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF21,"/f21");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF22,"/f22");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF23,"/f23");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF31,"/f31");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF32,"/f32");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF33,"/f33");

  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandNeuStarten,"/NeuStarten");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandGame,"/game_get.html");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandRunPostScheme,"/playpost");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandRunWidgetScheme,"/playwidget");

  //thats it
}

MSHTMLKontext_Game_Get::~MSHTMLKontext_Game_Get()
{
  // nothing to do, formally only
}
```

```
// the command handlers

void MSHTMLKontext_Game_Get::CommandF11(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("11");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF12(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("12");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF13(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("13");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF21(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("21");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF22(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("22");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF23(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("23");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF31(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("31");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF32(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("32");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandF33(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("33");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}
```

```
void MSHTMLKontext_Game_Get::CommandNeuStarten(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh,
string htmlcommand)
{
   // The user wants the game to be initialized again.
  dreikreuze.NeuInitialisieren();

  _loadDocumentTemplate("spiele.html");

  // the ID must be set again to the document
  setIDScript = true;

  // declare all page handlers
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF11,"/f11");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF12,"/f12");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF13,"/f13");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF21,"/f21");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF22,"/f22");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF23,"/f23");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF31,"/f31");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF32,"/f32");
  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandF33,"/f33");

  MS_InsertHandler(MSHTMLKontext_Game_Get,CommandNeuStarten,"/NeuStarten");

  // send the new set page data...
  html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content

}

void MSHTMLKontext_Game_Get::CommandGame(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
  // just show the page. It is not a real command. Useful if the browser is doing a 'reload'
  html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Get::CommandRunPostScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh,
string htmlcommand)
{
  // user wants to play the get scheme
  auto nc = new MSHTMLKontext_Game_Post(1, _extended_params);

  // Set the new context independetly (->not modal) and call the initial page of it
  SetNewContextNotModal(nc,nc->DPN(),html_oh);

  // delete this context as soon as possible
  html_oh.RemoveContext(GetMainContextID());
}

void MSHTMLKontext_Game_Get::CommandRunWidgetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader&
html_oh, string htmlcommand)
{
  // user wants to play the get scheme
  auto nc = new MSHTMLKontext_GameWidget(1, _extended_params);

  // Set the new context independetly (->not modal) and call the initial page of it
  SetNewContextNotModal(nc,nc->DPN(),html_oh);

  // delete this context as soon as possible
  html_oh.RemoveContext(GetMainContextID());
}



void MSHTMLKontext_Game_Get::checkwin()
{
  if(dreikreuze.Gewonnen()==1)
  {
    // The user won the game. Change the element content in order to explain the new situation to
the user.
    auto tmp = thedocumentdisplayed->FindObject("id","restart");
    if(tmp)
    {
      tmp->ReplaceElement(new MSHTML_a2("#", "X wins, Restart Game"),"id","NeuStarten");
    }
```

```
  }
  if(dreikreuze.Gewonnen()==2)
  {
    // The computer won the game. Change the element content in order to explain the new situation
to the user.
    auto tmp = thedocumentdisplayed->FindObject("id","restart");
    if(tmp)
    {
      tmp->ReplaceElement(new MSHTML_a2("#", "O wins, Restart Game"),"id","NeuStarten");
    }
  }
  if(dreikreuze.Gewonnen()==3)
  {
    // Nobody won the game. Change the element content in order to explain the new situation to the
user.
    auto tmp = thedocumentdisplayed->FindObject("id","restart");
    if(tmp)
    {
      tmp->ReplaceElement(new MSHTML_a2("#", "nobody wins, Restart Game"),"id","NeuStarten");
    }
  }
}


//! The user clicked on a playing field. Register it and give an reaction -> exchange the displayed
image.
void MSHTMLKontext_Game_Get::KommandoFeld(string feld)
{
  if(dreikreuze.FeldBespielbar(feld))
  {
    auto tmp = thedocumentdisplayed->FindObject("id","i"+feld);
    if(tmp)
    {
      tmp->ReplaceAttributeContent("src","kreuz.png"); // show a cross
    }
    tmp = thedocumentdisplayed->FindObject("id","i"+dreikreuze.GebeOFeld());
    if(tmp)
    {
      tmp->ReplaceAttributeContent("src","kreis.png"); // show a circle
    }
  }
}

// The definitions for the POST scheme

MSHTMLKontext_Game_Post::MSHTMLKontext_Game_Post(unsigned int stack_level, map<string,string>&
extended_params)
: MSHTMLKontextN(stack_level,"game_post.html",extended_params,nullptr)
{

  // now loading the game view from the HTML5, store it in thedocumentdisplayed
  _loadDocumentTemplate("spiele_post.html");


  // insert the right ID into the document
  InsertFormCoatiid(GetMainContextID());


  // declare all page handlers
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF11,"/f11");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF12,"/f12");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF13,"/f13");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF21,"/f21");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF22,"/f22");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF23,"/f23");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF31,"/f31");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF32,"/f32");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandF33,"/f33");

  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandNeuStarten,"/NeuStarten");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandGame,"/game_post.html");
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandRunGetScheme,"/playget");
  _connect_element_method_POST("playget", onclick);
  MS_InsertHandler(MSHTMLKontext_Game_Post,CommandRunWidgetScheme,"/playwidget");
```

```
  _connect_element_method_POST("playwidget", onclick);



  // connect the intended active elements of the document with the commands above
  _connect_element_method_POST("f11", onclick);
  _connect_element_method_POST("f12", onclick);
  _connect_element_method_POST("f13", onclick);
  _connect_element_method_POST("f21", onclick);
  _connect_element_method_POST("f22", onclick);
  _connect_element_method_POST("f23", onclick);
  _connect_element_method_POST("f31", onclick);
  _connect_element_method_POST("f32", onclick);
  _connect_element_method_POST("f33", onclick);
  _connect_element_method_POST("NeuStarten", onclick);
  //thats it

}

MSHTMLKontext_Game_Post::~MSHTMLKontext_Game_Post()
{
  // nothing to do, formally only
}


// the command handlers

void MSHTMLKontext_Game_Post::CommandF11(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("11");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandF12(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("12");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandF13(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("13");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandF21(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("21");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandF22(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("22");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandF23(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("23");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}
```

```
void MSHTMLKontext_Game_Post::CommandF31(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("31");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandF32(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("32");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandF33(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
   KommandoFeld("33");
   checkwin();
   html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}


void MSHTMLKontext_Game_Post::CommandNeuStarten(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh,
string htmlcommand)
{
   // The user wants the game to be initialized again.
  dreikreuze.NeuInitialisieren();

  // reload the HTML5 template again
  _loadDocumentTemplate("spiele_post.html");

  // insert the right ID into the document
  InsertFormCoatiid(GetMainContextID());


  // connect the intended active elements of the document with the commands above
  _connect_element_method_POST("f11", onclick);
  _connect_element_method_POST("f12", onclick);
  _connect_element_method_POST("f13", onclick);
  _connect_element_method_POST("f21", onclick);
  _connect_element_method_POST("f22", onclick);
  _connect_element_method_POST("f23", onclick);
  _connect_element_method_POST("f31", onclick);
  _connect_element_method_POST("f32", onclick);
  _connect_element_method_POST("f33", onclick);
  _connect_element_method_POST("NeuStarten", onclick);


  // send the new set page data...
  html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content

}

void MSHTMLKontext_Game_Post::CommandGame(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
  // just show the page. It is not a real command. Useful if the browser is doing a 'reload'
  html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_Game_Post::CommandRunGetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh,
string htmlcommand)
{
  // user wants to play the get scheme
  auto nc = new MSHTMLKontext_Game_Get(1, _extended_params);

  // Set the new context independetly (->not modal) and call the initial page of it
  SetNewContextNotModal(nc,nc->DPN(),html_oh);

  // delete this context as soon as possible
  html_oh.RemoveContext(GetMainContextID());
}
```

```
void MSHTMLKontext_Game_Post::CommandRunWidgetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader&
html_oh, string htmlcommand)
{
  // user wants to play the get scheme
  auto nc = new MSHTMLKontext_GameWidget(1, _extended_params);

  // Set the new context independetly (->not modal) and call the initial page of it
  SetNewContextNotModal(nc,nc->DPN(),html_oh);

  // delete this context as soon as possible
  html_oh.RemoveContext(GetMainContextID());
}


void MSHTMLKontext_Game_Post::checkwin()
{
  if(dreikreuze.Gewonnen()==1)
  {
    // The user won the game. Change the element content in order to explain the new situation to
the user.
    // The element is an underlined text.
    auto tmp = thedocumentdisplayed->FindObject("id","NeuStarten");
    if(tmp)
    {
      tmp->RemoveSubelements();
      MSHTML_u2* u = new MSHTML_u2();
      *u << new MSHTML_text2("X wins, Restart Game");
      *tmp << u;
    }

  }
  if(dreikreuze.Gewonnen()==2)
  {
    // The computer won the game. Change the element content in order to explain the new situation
to the user.
    // The element is an underlined text.
    auto tmp = thedocumentdisplayed->FindObject("id","NeuStarten");
    if(tmp)
    {
      tmp->RemoveSubelements();
      MSHTML_u2* u = new MSHTML_u2();
      *u << new MSHTML_text2("O wins, Restart Game");
      *tmp << u;
    }
  }
  if(dreikreuze.Gewonnen()==3)
  {
    // Nobody won the game. Change the element content in order to explain the new situation to the
user.
    // The element is an underlined text.
    auto tmp = thedocumentdisplayed->FindObject("id","NeuStarten");
    if(tmp)
    {
      tmp->RemoveSubelements();
      MSHTML_u2* u = new MSHTML_u2();
      *u << new MSHTML_text2("nobody wins, Restart Game");
      *tmp << u;
    }
  }
}


//! The user clicked on a playing field. Register it and give an reaction -> exchange the displayed
image.
void MSHTMLKontext_Game_Post::KommandoFeld(string feld)
{
  if(dreikreuze.FeldBespielbar(feld))
  {
    auto tmp = thedocumentdisplayed->FindObject("id","i"+feld);
    if(tmp)
    {
      tmp->ReplaceAttributeContent("src","kreuz.png"); // show a cross
    }
    tmp = thedocumentdisplayed->FindObject("id","i"+dreikreuze.GebeOFeld());
```

```
    if(tmp)
    {
      tmp->ReplaceAttributeContent("src","kreis.png"); // show a circle
    }
  }
}

// The definitions for the Widget scheme


MSHTMLKontext_GameWidget::MSHTMLKontext_GameWidget(unsigned int stack_level, map<string,string>&
extended_params)
: MSHTMLKontextN(stack_level,"game_widget.html",extended_params,nullptr)
{

  // now loading the game view from the HTML5, store it in thedocumentdisplayed
  _loadDocumentTemplate("spiele_widget.html");


  // insert the right ID into the document
  InsertFormCoatiid(GetMainContextID());


  // declare all page handlers
  MS_InsertHandler(MSHTMLKontext_GameWidget,CommandGame,"/game_widget.html");
  MS_InsertHandler(MSHTMLKontext_GameWidget,CommandRunGetScheme,"/playget");
  _connect_element_method_POST("playget", onclick);
  MS_InsertHandler(MSHTMLKontext_GameWidget,CommandRunPostScheme,"/playpost");
  _connect_element_method_POST("playpost", onclick);
  //thats it


  map<string,string> params; // formally, not used for this game

  auto f = [this](string element, MSHTMLinHeader& html_ih)->string{

    string antwort = "";
    if(dreikreuze.FeldBespielbar(element.substr(1)))
    {
       antwort += _widgetmanager.ChangeWidgetAttribute(element, "src", "kreuz.png");
       antwort += _widgetmanager.RemoveActionWidgetObject(element,"onclick");
       string reaction = "i"+dreikreuze.GebeOFeld();
       antwort += _widgetmanager.ChangeWidgetAttribute(reaction, "src", "kreis.png");
       antwort += _widgetmanager.RemoveActionWidgetObject(reaction,"onclick");
       switch(dreikreuze.Gewonnen())
       {
        case 1:
          antwort += _widgetmanager.ChangeWidgetHTML("NeuStartenTxt","X wins, Restart Game");
          break;
        case 2:
          antwort += _widgetmanager.ChangeWidgetHTML("NeuStartenTxt","O wins, Restart Game");
          break;
        case 3:
          antwort += _widgetmanager.ChangeWidgetHTML("NeuStartenTxt","nobody wins, Restart Game");
          break;
       };
    }
    return antwort;
  };

  // declare all widget handlers
  _widgetmanager.AddWidgetObject("i11", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i12", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i13", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i21", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i22", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i23", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i31", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i32", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("i33", *this, "onclick",f ,params);
  _widgetmanager.AddWidgetObject("NeuStartenTxt", *this, "coati",f ,params); // formally, does
nothing

  auto f1 = [f,this](string element, MSHTMLinHeader& html_ih)->string{
```

```
    string antwort = "";
    // The user wants the game to be initialized again.
    dreikreuze.NeuInitialisieren();

    map<string,string> params; // formally, not used for this game
    // declare all widget handler actions for onclick
    antwort += _widgetmanager.AddWidgetObject("i11", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i12", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i13", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i21", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i22", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i23", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i31", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i32", *this, "onclick",f ,params);
    antwort += _widgetmanager.AddWidgetObject("i33", *this, "onclick",f ,params);

    //reinitialize all widget objects
    antwort += _widgetmanager.ChangeWidgetHTML("NeuStartenTxt","Restart Game");
    antwort += _widgetmanager.ChangeWidgetAttribute("i11", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i12", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i13", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i21", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i22", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i23", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i31", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i32", "src", "leer.png");
    antwort += _widgetmanager.ChangeWidgetAttribute("i33", "src", "leer.png");

    return antwort;
  };
  _widgetmanager.AddWidgetObject("NeuStarten", *this, "onclick",f1 ,params);
}

MSHTMLKontext_GameWidget::~MSHTMLKontext_GameWidget()
{
  // nothing to do, formally only
}


// the command handlers

void MSHTMLKontext_GameWidget::CommandGame(MSHTMLinHeader& html_ih, MSHTMLoutHeader& html_oh, string
htmlcommand)
{
  // just show the page. It is not a real command. Useful if the browser is doing a 'reload'
  html_oh.SetHTMLResponse200(GetBuffer()); // set the new page content
}

void MSHTMLKontext_GameWidget::CommandRunGetScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader&
html_oh, string htmlcommand)
{
  // user wants to play the get scheme
  auto nc = new MSHTMLKontext_Game_Get(1, _extended_params);

  // Set the new context independetly (->not modal) and call the initial page of it
  SetNewContextNotModal(nc,nc->DPN(),html_oh);

  // delete this context as soon as possible
  html_oh.RemoveContext(GetMainContextID());
}

void MSHTMLKontext_GameWidget::CommandRunPostScheme(MSHTMLinHeader& html_ih, MSHTMLoutHeader&
html_oh, string htmlcommand)
{
  // user wants to play the get scheme
  auto nc = new MSHTMLKontext_Game_Post(1, _extended_params);

  // Set the new context independetly (->not modal) and call the initial page of it
  SetNewContextNotModal(nc,nc->DPN(),html_oh);

  // delete this context as soon as possible
  html_oh.RemoveContext(GetMainContextID());
}
```

```
//! the main program
MSAppViewportBase* coatimain(int argc, char* argv[], bool isX)
{

 function<MSCoatiApplication*(MSAppViewportBase*, bool)> start_local = [](MSAppViewportBase* msavb,
bool isX)
 {
   MSCoatiApplication* theapp = nullptr;

  check_program(msavb->Argv()[0], msavb->GetAppname(isX));

  cout << "Twymt++ " <<  TWYMTVERSION << " based on CoatiFrame "<< COATIFRAMEWORK << " release#: "
<< ReleaseNumber() << endl;

  string htmlroot = MSGetCurrentDirName();
#ifndef MSANDROIDVERSION
  htmlroot += DataPath("/../htmlroot");
#else
  htmlroot += DataPath("/htmlroot");
#endif

  if(!CheckInstallFiles(htmlroot ,

{"kreuz.png","leer.png","kreis.png","mellspa2.png","spiele.html","spiele_post.html","spiele_widget.h
tml","spiele.css"},
                        "http://www.mellspa.de/Downloads/CoatiGame"))
  {
    cout << "Error installing missing files." << endl;
    MS_throw("Error installing missing files.");
  }

  // The startpath parameter is used to load the HTML-templates from disk
  map<string,string> params;
  params["startpath"] = htmlroot;

  // getting an instance of the game-class-server on port 0 == "use a free port"
  // do it instructively, not as short as C++ it permits

  // Important notice: The next only 4 instructions (and already very long expressed) are needed to
implement the
  // the _whole_ Coati server into an application. All other instruction are preparing the
environment for each supported
  // environment (Linux, Raspberry Pi (command line and X), Android, and Windows (32bit command line
or graphical)).

  MSCoatiInterfaceProps http("127.0.0.1", 0, MS_PROTOCOL_HTTP);
  vector<MSCoatiInterfaceProps> v;
  v.push_back(http);
  theapp = new MSCoatiApplication(v,
                                  [](MSHTMLStartKontextN* t){return new MSHTMLKontext_GameWidget(1,t-
>GetExtendedParams());},
                                  params
                                 );

  log0_out << "going to run" << endl;

  return theapp;
 };


  // the Linux, Android, Raspberry, and Windows case
  return new MSAppViewportBase("XTheGame", "TheGame", start_local);


}

// organising the program loading stuff
#include <mscoatimain.h>
```

CoatiFrame 1.12

General idea

Doing projects in C++ which should offer a quick and nice graphical user interface become difficult if the code should

- support different operating systems,
- be as small as possible,
- not depend on other libraries or special third party products
- support many languages at once.
- no indirect maintenance costs or efforts due to the HTTP-server or HTTPS-server

A HTTP or HTTPS-server can interact with browsers and represent an application in a graphical way.

Twymt++ 1.1

General idea

The CoatiFrame 1.12 framework is a powerful base to write HTTP server applications with respect to many sophisticated challenges. Twymt++ sets a logical class interface on top of the CoatiFrame 1.12 framework. The main goal of the Twymt++ interface is that the application developer remains informed about http/https communication, but the interface guarantees a simple event scheme.

The application is working as single instance as well as multiple instance.

The class interface is defined by:

- an instance of MSCoatiApplication – the HTTP/HTTPS CoatiFrame 1.12 server framework and
- an own context class derived from MSHTMLKontextN – the private 'interface' base class of any Twymt++ app. The pointer of that class is returned by a lambda which is set as an argument to the MSCoatiApplication instance.

The own instance of the MSHTMLKontextN defines the behavior of your application. Furthermore a simplified start-up scheme can be used to facilitate a rapid application development.

Prerequisites

The framework is based on C++11 standard libraries and the TCP/IP functionality offered by the operating system. Libraries and data preparation programs for 32/64-bit Linux (x86_64, arm_32) and Windows (32/64bit) are available. Due to the minimalistic approach other platforms are probably supported.

There are no further library or compiler switch dependencies except that the used compiler needs to support C++11.

Scope of delivery

The licensee will get libraries for Linux (x86_64, arm32), and Windows (32/64bit) including all needed headers and a complete user manual of the framework.

Licenses

There is a single user license which is valid to be used on a single computer, a multi-user-license for the use on maximal 10 computers and a company license for an unlimited number of computers of one company. Prices are on e-mail request: *info@mellspa.de*