Необходимо с текущей даты загружать курсы валют: доллар США (код **840**), российский рубль (код **643**), евро (код **978**) с сайта НБ РБ (реализовать в виде службы Windows двумя разными способами: используя формат XML и через API). Курсы сохранять в текстовые файлы в две отдельные папки, т.е. вариант загрузки через XML сохранять в один каталог, а через API в другой. Предусмотреть возможность менять каталоги для сохранения файлов (можно использовать конфигурационный файл службы *.ini). Файлы должны создаваться в каталоге каждый день, имя файла должно содержать текущую дату. Пример содержания файла приведен ниже.
Исходные файлы проекта, по возможности, залить на гит и написать краткую инструкцию по установке готовой службы на ПК.

---

1. После установки службу надо запустить.

2. Файлы

1. ServiceProject.exe - исполняемый файл со службой
2. deleteSrv.cmd - удаление службы
3. installSrv.cmd - установка службы
4. uninstallSrv.cmd - деинсталляция службы
5. Settings.ini - файл настроек путей, по которым будут записываться файлы курсов валют.
6. Logger.log - журнал работы сервиса.

Как только служба запустится, она проверит наличие файла Setting.ini. Если данного файла нет, она создаст его и заполнит значениями по умолчанию.

```
[XML]
folderPath=D:\Temp\XML
[JSON]
folderPath=D:\Temp\JSON
```

Все нужные папки на старте службы тоже проверяются. Если чего-то не хватает - все будет создано.

Запущенная служба создаст отдельный поток.
В этом потоке решаются все вопросы с получением курсов валют.
Если полученные файлы с курсами удалить, то примерно через 4 минуты они будут сформированы вновь.

Если файлы уже сформированы на текущую дату, они не будут обновляться.
На следующие сутки сформируются новые файлы.

Большая часть из происходящего внутри сервиса и потока логируется в файл **Logger.log**.

Проект собирался на **Delphi 10.3**. В проекте используются методы классов `TFile` и `TDirectory` - проект в Delphi XE из-за этого не соберется, так как данные классы появились в версии, начиная с Delphi XE2.

Исходники (на всякий случай):

uMain.pas

```
{.$DEFINE CHECK_FOR_EXECUTE}
unit uMain;

interface

uses

  Winapi.Windows,
  Winapi.Messages,
  System.SysUtils,
  System.Classes,
  Vcl.Graphics,
  Vcl.Controls,
  Vcl.SvcMgr,
  Vcl.Dialogs,
  WorkerThreadU;

  const c_ServiceName = 'The_simplest_currency_service';
```

```
type

    TSampleService = class(TService)
    procedure ServiceExecute(Sender: TService);
    procedure ServiceStart(Sender: TService; var Started: Boolean);
    procedure ServiceStop(Sender: TService; var Stopped: Boolean);
    procedure ServicePause(Sender: TService; var Paused: Boolean);
    procedure ServiceContinue(Sender: TService; var Continued: Boolean);
  private
    FWorkerThread: TWorkerThread;
  public
    function GetServiceController: TServiceController; override;
  end;

  var
  SampleService: TSampleService;

implementation
{$R *.dfm}

uses System.IOUtils, IniFiles, uUtilities;

procedure ServiceController(CtrlCode: DWord); stdcall;
begin
  SampleService.Controller(CtrlCode);
end;

function TSampleService.GetServiceController: TServiceController;
begin
  Result := ServiceController;
end;

procedure TSampleService.ServiceContinue(Sender: TService; var Continued:
Boolean);

begin
  FWorkerThread.Continue;
  Continued := True;
end;

procedure TSampleService.ServicePause(Sender: TService; var Paused:
Boolean);

begin
  FWorkerThread.Pause;
  Paused := True;
end;

procedure TSampleService.ServiceStart(Sender: TService; var Started:
Boolean);
var
```

```
  IniFileName: TFileName;
  ini : TIniFile;
  folderXML, folderJSON: TFileName;
  ExePath: TFileName;
begin

  ExePath := TPath.GetDirectoryName(GetModuleName(HInstance));
  IniFileName := TPath.Combine(ExePath, 'Settings') + '.ini';
  ini := TIniFile.Create(IniFileName);

  try
    folderXML := ini.ReadString('XML', 'folderPath', '');
    folderJSON := ini.ReadString('JSON', 'folderPath', '');

    if folderXML = '' then
    begin
      ini.WriteString('XML', 'folderPath', ExePath + '\XML');
      folderXML := ini.ReadString('XML', 'folderPath', '');
    end;

    if folderJSON = '' then
    begin
      ini.WriteString('JSON', 'folderPath', ExePath + '\JSON');
      folderJSON := ini.ReadString('JSON', 'folderPath', '');
    end;

    if not TDirectory.Exists(folderXML) then
      TDirectory.CreateDirectory(folderXML);

    if not TDirectory.Exists(folderJSON) then
      TDirectory.CreateDirectory(folderJSON);

  finally
    ini.Free;
  end;


  logString('TSampleService.ServiceStart');
  logString(Format('folder for XML: %s; folder for JSON: %s', [folderXML,
folderJSON]));

  FWorkerThread := TWorkerThread.Create(True);

  FWorkerThread.setFoldersPath(folderXML, folderJSON);
  FWorkerThread.Start;
  Started := True;
end;

procedure TSampleService.ServiceStop(Sender: TService; var Stopped:
Boolean);
begin
```

```
  logString('TSampleService.Stop');
  FWorkerThread.Terminate;
  FWorkerThread.WaitFor;
  FreeAndNil(FWorkerThread);
  Stopped := True;
end;

procedure TSampleService.ServiceExecute(Sender: TService);
begin
  while not Terminated do
  begin
    {$IFDEF CHECK_FOR_EXECUTE}logString('TSampleService.Execute');{$ENDIF}
    ServiceThread.ProcessRequests(false);
    TThread.Sleep(1000);
  end;
end;

end.
```

uHTTPQuery.pas

```
unit uHTTPQuery;

interface

uses
  SysUtils, HTTPApp, IdHTTP, XMLDoc, XMLIntf, ActiveX, System.IOUtils,
System.Classes;

const
  SERVER_ADDRESS_XML = 'https://www.nbrb.by/Services/XmlExRates.aspx?
ondate=';
  SERVER_ADDRESS_JSON = 'https://www.nbrb.by/api/exrates/rates/%d?
parammode=1';
  USD = 840;
  RUB = 643;
  EUR = 978;


function getDataXML(): string;
function getDataJSONByCurrency(const aCurrenyCode: word): string;

implementation

uses
  uUtilities;
```

```
function getDataXML: string;
var
  CoResult: Integer;
  HTTP: TIdHTTP;
  Request: String;

begin

  logString('...function getData...');

  try
    CoResult := CoInitializeEx(nil, COINIT_MULTITHREADED);

    if not((CoResult = S_OK) or (CoResult = S_FALSE)) then
    begin
      logString(Format('XML - getData result - fail (#%d)', [CoResult]));
      Exit;
    end;

    HTTP := TIdHTTP.Create;
    Request := SERVER_ADDRESS_XML;
    Result := HTTP.Get(Request);
    HTTP.Destroy;

  except
    on E: Exception do
    begin
      logString(Format('XML - getData - (%s) message: %s', [E.ClassName,
E.Message]));
      result := '';
    end;
  end;
end;

function getDataJSONByCurrency(const aCurrenyCode: word): string;
var
  CoResult: Integer;
  HTTP: TIdHTTP;
  Request: String;
  stream : TStringStream;

begin

  logString('...function getData...');

  HTTP := TIdHTTP.Create;

  try
    try
      CoResult := CoInitializeEx(nil, COINIT_MULTITHREADED);
```

```
      if not((CoResult = S_OK) or (CoResult = S_FALSE)) then
      begin
        logString(Format('JSON - getData result - fail (#%d)',
[CoResult]));
        Exit;
      end;

      request := Format(SERVER_ADDRESS_JSON, [aCurrenyCode]);
      stream := TStringStream.Create(Result);
      HTTP.Get(Request, stream);
      stream.Position := 0;
      Result := stream.ReadString(stream.Size);

      // Display document content
      logString(Result);

    finally
      FreeAndNil(HTTP);
      FreeAndNil(stream);
      // //HTTP.Destroy;
    end;

  except
    on E: Exception do
    begin
      logString(Format('JSON - getData - (%s) message: %s', [E.ClassName,
E.Message]));
      result := '';
    end;
  end;
end;

end.
```

uUtilities.pas

```
unit uUtilities;

interface
uses
  SysUtils
    , XMLDoc
    , XMLIntf
    , ActiveX
    , System.IOUtils
    , System.JSON
    , System.SyncObjs
    , System.Classes;
```

```
  const

    TITLE = $01;
    FOOTER = $02;
    NOTHING = $00;
    OVERWRITE = true;
    RESULT_OK = true;
    RESULT_FAILED = false;

  function parseXML(const sXML: string; const aFileName: string): boolean;
  function parseJSON(const sJSON: String; const aFileName: string;
aAnyThing: byte = NOTHING): boolean;

  procedure logString(aMessage: String);

implementation

  procedure appendToFile(const aFileName: Tfilename; const aData: string;
aOverwrite: boolean = false);
  var
    section: TCriticalSection;

  begin
    section := TCriticalSection.Create;
    section.Enter;

    try
      if aOverwrite then
        TFile.WriteAllText(aFileName, aData + #$0D#$0A, TEncoding.ANSI)
      else
        TFile.AppendAllText(aFileName, aData + #$0D#$0A, TEncoding.ANSI);
    except on ex: EInOutError do
      begin
        logString('append to file error ' + ex.Message);
        raise Exception.Create('File writing failed');
      end;
    end;

   section.Leave;

  end;

  procedure logString(aMessage: string);
  var
    ExePath, LogFileName: TFileName;
    section: TCriticalSection;

  begin

    section := TCriticalSection.Create;
```

```
    section.Enter;
    try
      ExePath := TPath.GetDirectoryName(GetModuleName(HInstance));
      LogFileName := TPath.Combine(ExePath, 'Logger.log');
      TFile.AppendAllText(LogFileName, Format('%s [%s] > %s %s',
        [DateToStr(now), TimeToStr(now), aMessage, #$0D#$0A]),
TEncoding.ANSI);
    except on ex: EInOutError do
       //log('Error writing log...');
    end;
    section.Leave;

  end;

  function parseXML(const sXML: string; const aFileName: string): Boolean;
  var
    Doc: IXMLDocument;
    Node: IXMLNode;
    BufXMLNodeList : IXMLNodeList;
    i: integer;
    NumCode, CharCode, Rate: String;

  begin
    try
      Doc := TXMLDocument.Create(nil);
      Doc.LoadFromXML(sXML);

      if Doc.ChildNodes[1].HasChildNodes then
      begin
        BufXMLNodeList := Doc.ChildNodes[1].ChildNodes;

        AppendToFile(aFileName, 'Курсы валют на ' + DateToStr(now),
OVERWRITE);

        for i := 0 to Pred(BufXMLNodeList.Count) do
        begin
          NumCode := BufXMLNodeList[i].ChildNodes['NumCode'].NodeValue;
          if (NumCode = '840') or (NumCode = '643') or (NumCode = '978')
then
          begin
            Rate := BufXMLNodeList[i].ChildNodes['Rate'].NodeValue;
            CharCode :=
BufXMLNodeList[i].ChildNodes['CharCode'].NodeValue;
            AppendToFile(aFileName, Format('Валюта: %s; Курс: %s',
[CharCode, Rate]));
          end;
        end;
        AppendToFile(aFileName, 'Загрузка выполнена в: ' +
DateTimeToStr(now));
        result := RESULT_OK;
```

```
      end;
    except on ex:Exception do
      begin
        logString('inner error ' + ex.Message);
        result := RESULT_FAILED;
      end;
    end;
  end;

  function parseJSON(const sJSON: String; const aFileName: string;
aAnyThing: byte = NOTHING): boolean;
  var
    JSON: TJSONObject;
    CharCode, Rate: String;

  begin
    try

      if aAnyThing = TITLE then
        AppendToFile(aFileName, 'Курсы валют на ' + DateToStr(now),
OVERWRITE);

      JSON := TJSONObject.ParseJSONValue(sJSON, False, True) as
TJSONObject;
      try
        CharCode := JSON.Values['Cur_Abbreviation'].Value;
        Rate := JSON.Values['Cur_OfficialRate'].Value;
        AppendToFile(aFileName, Format('Валюта: %s; Курс: %s', [CharCode,
Rate]));
      finally
        JSON.Free;
      end;

      if aAnyThing = FOOTER then
        AppendToFile(aFileName, 'Загрузка выполнена в: ' +
DateTimeToStr(now));
        result := RESULT_OK;

    except on ex:Exception do
      begin
        logString('inner error' + ex.Message);
        result := RESULT_FAILED;
      end;
    end;

  end;

end.
```

WorkerThreadU.pas

```
unit WorkerThreadU;

interface

uses
  System.Classes, System.SysUtils;

type
  TWorkerThread = class(TThread)
  private
    FPaused: Boolean;
    fFolderXML, fFolderJSON: TFileName;
  protected
    procedure Execute; override;
  public
    procedure Pause;
    procedure Continue;
    procedure setFoldersPath(const aFolderXML, aFolderJSON: TfileName);
  end;

implementation

uses
  System.IOUtils,
  uHTTPQuery,
  uUtilities;

procedure TWorkerThread.Continue;
begin
  FPaused := False;
end;

procedure TWorkerThread.Execute;
var
  fileName: TFileName;
  xml, json: integer;

begin
  logString('Thread Running...');
  FPaused := False;

(*************************************************************************
************************
    Что делает поток:
    1. Смотрит каждую минуту есть ли на сегодня файлы с курсами валют.
    1.1. Если ХОТЯ БЫ одного из файлов нету, поток во внутреннем цикле
пытается каждые 10 секунд
         стучаться на сервер, который ему отдаст нужные данные. Если все
ок (фа йлы сформированы),
         то будет выход из цикла и поток продолжит работу.
```

```
    1.2 Если файлы на месте за текущую дату, то на сервер из потока
запросы не полетят.

**************************************************************************
**********************)

  while not Terminated do
  begin
    fileName := '\currencyData_' + FormatDateTime('yyyy_mm_dd', now) +
'.txt';
    logString(fFolderXML + fileName);
    logString(fFolderJSON + fileName);

    if (not FileExists(fFolderXML + fileName)) or (not
FileExists(fFolderJSON + fileName)) then
    try
        xml := 0;
        json := 0;

        while true do
        begin

          logString('#');
          if parseXML(getDataXML(), fFolderXML + fileName) then inc(xml);

          if parseJSON(getDataJSONByCurrency(USD), fFolderJSON + fileName,
TITLE) then inc(json);
          if parseJSON(getDataJSONByCurrency(RUB), fFolderJSON + fileName)
then inc(json) ;
          if parseJSON(getDataJSONByCurrency(EUR), fFolderJSON + fileName,
FOOTER) then inc(json);
          sleep (10000);

          if (xml = 1) and (json = 3) then
          begin
            logString('На сегодня файлы с курсами валют сформированы');
            break;
          end
          else
          begin
            xml := 0;
            json := 0;
            logString('Сервер не отдал данныые...');
          end;
        end;

    except on E: Exception do
      begin
        logString(Format('Thread execute failed %s, %s', [E.ClassName,
E.Message]));
      end;
```

```
    end;

    sleep(600000);
  end;
end;

procedure TWorkerThread.Pause;
begin
  FPaused := True;
end;

procedure TWorkerThread.setFoldersPath(const aFolderXML, aFolderJSON:
TfileName);
begin
  fFolderXML := aFolderXML;
  fFolderJSON := aFolderJSON;
end;

end.
```