# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>0</td>
</tr>
<tr>
<td><strong>Part I  Introduction</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>Part II  Getting started</strong></td>
<td>5</td>
</tr>
<tr>
<td>1 Email Loader</td>
<td>6</td>
</tr>
<tr>
<td>2 Class Factory</td>
<td>7</td>
</tr>
<tr>
<td>3 POP3/IMAP adapter</td>
<td>8</td>
</tr>
<tr>
<td>4 Database adapter</td>
<td>8</td>
</tr>
<tr>
<td>5 Account class</td>
<td>9</td>
</tr>
<tr>
<td><strong>Part III  Full text search</strong></td>
<td>12</td>
</tr>
<tr>
<td><strong>Part IV  Customization</strong></td>
<td>13</td>
</tr>
<tr>
<td>1 Changing database SQL script</td>
<td>13</td>
</tr>
<tr>
<td>2 Adding email message fields</td>
<td>13</td>
</tr>
<tr>
<td>3 Registering document (attachment) parser</td>
<td>13</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>0</td>
</tr>
</tbody>
</table>
1 Introduction

HTML Email Library is a cross-platform framework, designed for simplifying adding email client features to Delphi applications. It has no GUI dependencies, so can be used for service application as well as for desktop clients. Framework doesn't contain own classes for POP3/IMAP/SMTP protocols or database access but instead can use any existing internet/DB library which can be easily replaced without rewriting main application.

Features:
- Background loading (each account is checked in a separate thread).
- Detection of main (body) part.
- SSL/StarTLS support.
- Automatic DB tables/triggers creation (current version has SQL scripts for Firebird and Oracle).
- Creation of email summary (annotation).
- Automatic creation of full text search index (for any SQL database).
- Documents in MS Word DOCX and RTF formats can be included in full text search index.
- Any internet library can be used for POP3/IMAP protocols (current version contains adapters for Indy and ICS).
- Any DB access library can be used to store emails (current version include FireDAC adapter).
2 Getting started

To start using library it is necessary to create `THtMailFactory` class descendant which will define classes used for internet and database access.

Example of MailFactory implementation:

```pascal
uses htMailIndy, HtMailFireDac;

TMyFactory = class(THtMailFactory)
public
  function CreateMailReceiveAdapter(const Account: THtMailAccount): IHtMailReceiveAdapter; override;
  function CreateMailDBAdapter: IHtMailDBAdapter; override;
end;

begin
  Result := TIndyMailboxAdapter.Create;;
end;

function TMyFactory.CreateMailDBAdapter: IHtMailDBAdapter;
begin
  Result := TFDMailAdapter.Create(MainForm.FDConnection1, 'MAILSCRIPT_FB', true);
end;

To receive mail message use `THtMailLoader` class.

```pascal
public
..
  Loader: THtMailLoader;
end;

// Create Loader
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Loader := THtMailLoader.Create(Factory);
  Loader.IndexedParts := [mpFromName, mpSubject, mpBody, mpAttachment];
end;

// Check mail on all accounts
procedure TMainForm.CheckMailBtnClick(Sender: TObject);
begin
  Loader.CheckMail
end;
```

Note that `CheckMail` will only start mail checking and execution of main thread will continue immediately.
To check certain account use `Loader.CheckAccount` method:

```pascal
Loader.CheckAccount(Loader.AccountById(ID));
```

To stop receiving new mail call `Loader.Terminate`.

## 2.1 Email Loader

`THtMailLoader` is a top level class used for receiving new emails and storing in database.

```pascal
constructor Create(AMailFactory: THtMailFactory);
```

Creates new mail loader instance and set factory. At this stage account list is created but not loaded from database.

```pascal
procedure LoadAccounts;
```

Loads accounts from database using DB adapter created via Factory. LoadAccounts is automatically called by CheckMail and AccountById if account list is empty.

```pascal
procedure CheckMail;
```

Start checking for new mail on all accounts. Each account is started in a separate thread.

```pascal
procedure CheckAccount(const Account: THtMailAccount);
```

Check one account for new mail.

```pascal
function Checking: boolean;
```

Is mail checking in progress (on any account).

```pascal
procedure Terminate;
```

Stop mail checking on all accounts

```pascal
function AccountById(const ID: string): THtMailAccount;
```

Find account by its ID. If no account is found, returns nil.

```pascal
property ExtractZIP: boolean read fExtractZIP write fExtractZIP;
```

Determine zip attachments handling. When set to true, zip attachments will be unpacked.
property IndexedParts: THTMailParts read fIndexedParts write fIndexedParts;

Determine which message parts should be added to text index

property Accounts: THTMailAccounts read fAccounts;

Access to accounts list.

2.2 Class Factory

THTMailFactory is used to define internet/DB access classes, document parsers and receive events. Application should not use THTMailFactory class directly but define own descendant.

Required methods (should be implemented in descendant).

function CreateMailReceiveAdapter(const Account: THTMailAccount): IHtMailReceiveAdapter;

Creates mailbox (POP3/IMAP) adapter for receiving mail.

function CreateMailDBAdapter: IHtMailDBAdapter;

Create database adapter for storing received messages and loading account list.

Optional methods (can be overriden in descendant)

function CreateMailAccount: THTMailAccount;

Creates Mail Account instance. Can be overridden for using custom account class.

function CreateMailMessage: THTMailMessage;

Creates Mail Message instance. Can be overridden for using custom mail message class.


Creates class for extracting plain text from message part (for indexing).

Events

procedure Log(const s: string; const Params: array of const;
              EventType: THTLogEventType);
Log message or error.

```delphi
data type Account = THtMailAccount;
data type Message = THtMailMessage;

procedure MessageReceived(const Account: THtMailAccount;
                         const Msg: THtMailMessage);
```

Called after message is received and parsed but before message is stored into database.

```delphi
procedure LoadingStarted(const Account: THtMailAccount);
```

Called for each mailbox on start.

```delphi
procedure LoadingCompleted(const Account: THtMailAccount);
```

Called for each mailbox on end

### 2.3 POP3/IMAP adapter

**IHtMailReceive** interface is used to work with POP3/IMAP protocols and has the following methods:

```delphi
procedure Connect(const AMailAccount: THtMailAccount; ALogEvent: THtLogEvent);
```

Connect to mail server.

```delphi
procedure LoadMessageList(const MessageList: THtMailMessages);
```

Load emails IDs from server.

```delphi
procedure LoadMessage(ServerNum: integer; const Msg: THtMailMessage);
```

Load single email from server.

Library contains **IHtMailReceive** implementations for **Indy** (htMailIndy unit) and **ICS** (htMailICS unit) libraries. Indy implementation supports both POP3 and IMAP protocols, ICS supports POP3 protocol only.

### 2.4 Database adapter

**IHtMailDBAdapter** interface is used to work with SQL or non-SQL databases.

```delphi
procedure GetAccountList(const Accounts: THtMailAccounts);
```

Load account list from database.
function SaveMessage(const AccountID, FolderID: string;
const Msg: THTMailMessage): string;

Save new message into database.

function FindMessage(const AccountID, UID: string): boolean;

Check if message is already present in database.

procedure UpdateLastUID(const AccountID, LastUID: string);

Update 'Last processed UID' field for the account.

procedure Commit;

Commit changes.

procedure Complete;

Called after all new messages are saved.

function AddAccount(const A: THTMailAccount): integer;

Add new account to database.

procedure UpdateAccount(const A: THTMailAccount);

Update account properties in database.

procedure CheckMailTables;

Check database for email tables and create them if necessary.

Library contains IHTMailDBAdapter implementation for FireDAC library (htMailFireDAC unit) and SQL scripts for Oracle and Firebird.

2.5 Account class

THTMailAccount class contains account settings and has following methods and properties:

procedure Check(const MailFactory: THTMailFactory);

Check account for new messages and get new messages.
procedure Terminate;

Stop loading messages from server.

property Host: string;

POP3 or IMAP Host and (optional) port. Example: imap.gmail.com:993

property User: string;

POP3 or IMAP user name

property Password: string;

POP3 or IMAP password

property FromName: string;

Sender name. Example: John Doe

property FromEmail: string;

Sender email address - admin@site.com

property SMTPHost: string;

SMTP Host and (optional) port. Example: smtp.gmail.com:443

property SMTPUser: string;

SMTP User name.

property SMTPPassword: string;

SMTP Password.

property LastAccountUID: string;

UID of last message successfully loaded from server

property IndexedParts: THtMailParts;

Which message parts should be added to text index.
property ExtractZIP: Boolean;

Extract files from .zip attachments.

property Added: integer;

Count of added messages on last check.
3 Full text search

Full text search index allows to implement fast searching for messages containing set of words in some part (subject, body, attachment). Indexed parts are defined by THTMailLoader\texttt{IndexedParts} property.

Index is composed by two tables

\textbf{Email\_Words} (Word\_ID, Word)
\textbf{Email\_Word\_Index} (Word\_ID, Msg\_Id, Weight, Part)

first table contains global set of words and second connects words to email messages parts.

Weight is number of word occurrences in a message, Part is set of email parts (first bit - From Name, 2nd - Subject, 3rd - Body, 4th- Attachment).

Sample SQL for searching emails containing both words Delphi and Library:

```sql
select m.* from email_messages m
  join (select wi.msg_id from email_words w left join email_word_index wi on w.word_id = wi.word_id on t0.msg_id = m.msg_id
  join (select wi.msg_id from email_words w left join email_word_index wi on w.word_id = wi.word_id on t1.msg_id = m.msg_id
order by sent desc
```

Sample SQL for searching emails containing any of the words Delphi and Library:

```sql
select m.* from email_messages m
  join (select wi.msg_id from email_words w left join email_word_index wi on w.word_id = wi.word_id on t0.msg_id = m.msg_id
order by sent desc
```
4 Customization

Enter topic text here.

4.1 Changing database SQL script

Database adapter can create necessary tables and triggers automatically by executing SQL script. SQL scripts are compiled into resource (.res) file and included in project. Library contains SQL scripts for Oracle (MAILSCRIPT_ORA) and Firebird (MAILSCRIPT_FB resource). To change existing script edit mailscript_fb.sql or mailscript.ora_sql and recompile mailscript.rc file.
To add new script, create sql file and add it into mailscript.rc. Script resource name should be passed to database adapter in THtMailFactory.CreateMailDBAdapter method.

4.2 Adding email message fields

THtEmailMessage class can be extended with custom fields.
1. Define THtEmailMessage class descendant with additional fields.
2. Use this class in THtMailFactory.CreateMailMessage method.
4. Add fields to database EMAIL_MESSAGES table.
5. Override IHtMailDBAdapter.SaveMessage method to store new fields in a database.

4.3 Registering document (attachment) parser

Document parsers are used in full text search index creation to include attachments of some type in index. Document parser should implement IHtDocumentParser interface with single method:

function DocumenttoText(var Part: THtMessagePart): string;

This method converts document stored in Part variable and returns plain text. To use new parser override THtMailFactory.CreateDocumentParser method. Example:

begin
  if SameText(ExtractFileExt(Part.FileName), '.xls') then
    Result := TXLSDocumentParser.Create
  else
    Result := inherited CreateDocumentParser(Part)
end;