[home](#) [articles](#) [quick answers](#) [discussions](#) [features](#) [community](#) [help](#)

Search for articles, questions, tips

[Articles](#) » [General Programming](#) » [Uncategorised Tips and Tricks](#) » [General](#)

Writing Email to the File of the PST Format

apriorit **Apriorit Inc, Alexander Maximov**, 6 Aug 2010

4.67 (6 votes)

Rate this:

The aim of this article is to create the library for writing e-mail messages in the *.pst format (used by Microsoft Outlook) to the file.

[Download source - 14.93 KB](#)

Table of Contents

- **Introduction**
 - The Structure of the Article
 - Used Technologies
 - The Structure of the Project
- **Brief Survey of MAPI**
- **Implementation**
 - MAPI Initialization
 - The Opening of the File
- **Data Writing**

- [The Creation of the Folders Hierarchy](#)
- [Writing the Message](#)
- [The Creation of the Message](#)
- [Writing the Message Properties](#)
- [Writing the Message Recipients](#)
- [Writing the Message Body](#)
- **How to Build and Run this Solution**
- **Testing**
- **Supported Windows Versions**

Introduction

The aim of this article is to create the library for writing e-mail messages in the *.pst format (used by Microsoft Outlook) to the file. This article will be interesting for those developers who develop various mail applications, which include the export to *.pst format functionality. This article is divided into two parts in order to make it easier for understanding and for better structuring. The first part introduces the common notions such as initialization, writing the message properties (sender, recipient, etc.); writing the text body of the message, and creation of the folders structure. The second part deals with more complicated notions such as writing the message attachment, RTF and HTML bodies of the message. The developed library is a sample and is not intended for usage in real applications. Its aim is to demonstrate the common principles of working with MAPI.

The Structure of the Article

The **Extended MAPI** part is devoted to the common information about the interface, its features and implementation.

The **Implementation** part deals with the key moments of implementation and the common architecture of the sample project.

The **Example** part describes how to build the project, use the library, and also provides the example of usage and the results of testing.

Technologies Used

- **C++**. The library and the example are written in C++ language using exceptions.
- **STLPort**. A standard library (for more information, see <http://www.stlport.org/>).
- **Extended MAPI**. MAPI (Messaging Application Programming Interface) is a messaging architecture and a Component Object Model based API for Microsoft Windows.

The Structure of the Project

- **bin** - Binary files
- **lib** - Library files
- **obj** - Object files
- **src** - Source files
- **| - PstWriter** – Main library that provides the export functionality

- | - **TestApp** – Test application that writes test message to the test PST file

Brief Survey of MAPI

Messaging Application Programming Interface (MAPI) is a messaging architecture and a Component Object Model based API for Microsoft Windows. MAPI allows client programs to become (e-mail) messaging-enabled, -aware, or -based by calling MAPI subsystem routines that interface with certain messaging servers. While MAPI is designed to be independent of the protocol, it is usually used with **MAPI/RPC**, the proprietary protocol that Microsoft Outlook uses to communicate with Microsoft Exchange.

Simple MAPI is a subset of 12 functions which enable developers to add basic messaging functionality. Extended MAPI allows complete control over the messaging system on the client computer, creation and management of messages, management of the client mailbox, service providers, and so forth. Simple MAPI ships with Microsoft Windows as part of Outlook Express/Windows Mail while the full Extended MAPI ships with Office Outlook and Exchange.

Implementation

This part describes in detail the work of the library with MAPI. The description and code pieces are also provided. The full version of the code is attached to the article. The code in the article does not include the check for errors and other parts, which are easy for understanding and do not play a significant role while working with MAPI.

MAPI Initialization

First, what you should do while working with MAPI is to initialize it. If you do not do this, all calls will return the error. To initialize MAPI, call the function:

[Hide](#) [Copy Code](#)

```
HRESULT MAPIInitialize(LPVOID lpMapiInit);
```

In the example, it is performed in the following way:

[Hide](#) [Copy Code](#)

```
MAPIINIT init = { 0, MAPI_MULTITHREAD_NOTIFICATIONS};  
HRESULT hr = MAPIInitialize(&init);
```

The **MAPI_MULTITHREAD_NOTIFICATIONS** parameter means that MAPI should generate notifications using a thread dedicated to notification handling instead of the first thread used to call **MAPIInitialize**. It's advisable to always define this parameter in order to avoid problems with threading. Also the **MAPIUninitialize** call should correspond to every **MAPIInitialize** call. The first should be called after the work with MAPI is finished. And this pair of methods should be called for each thread that uses MAPI.

The Opening of the File

The PST file is opened in several stages. All the work with the file is performed through MAPI and the application does not have direct access to it.

First, you should get an administration object for the service access:

[Hide](#) [Copy Code](#)

```
// Get a profile administration object.
hr = MAPIAdminProfiles(0, &m_pIProfAdmin);
...
DWORD dwStart = GetTickCount();
sprintf_s( m_profileName, 100, "Temp%lx", dwStart );
// Creating profile
(void) m_pIProfAdmin->DeleteProfile( (LPTSTR)&m_profileName, 0 );
hr = m_pIProfAdmin->CreateProfile( (LPTSTR)&m_profileName, NULL, 0, 0 );
...
// Get a service administration object.
hr = m_pIProfAdmin->AdminServices((LPTSTR)&m_profileName, 0, 0, 0, &m_pSvcAdmin);
```

The `m_profileName` defines the name of the profile in this case. In order not to open the existing profile, the profile name is generated in a random way and the attempt to delete it is performed.

Then it is necessary to create and configure the service, which will work with our file:

[Hide](#) [Shrink ▲](#) [Copy Code](#)

```
//Creating service
hr = m_pSvcAdmin->CreateMsgService( "MSUPST MS", "MAPI Pst Msg Store", 0, 0 )

// Configure the MS Personal Information Store per NewPST entries.
hr = m_pSvcAdmin->GetMsgServiceTable(0, &m_ptblSvc);

COM::AutoPtr<SRowSet,RowEraser> pRows;
COM::AutoPtr<SRowSet,RowEraser> pRows;

enum {iSvcName, iSvcUID, iEnrtyID, cptaSvc};

//Now we should find created service.
SizedSPropTagArray (cptaSvc, ptaSvc) = { cptaSvc,
    { PR_SERVICE_NAME, PR_SERVICE_UID, PR_ENTRYID} };
HrQueryAllRows(m_ptblSvc, (LPSPPropTagArray)&ptaSvc, NULL, NULL, 0, &pRows);

int cRows = pRows->cRows;

for(LPSRow pRow = pRows->aRow; pRow < pRows->aRow + cRows; ++pRow)
{
    //Here is our service
    m_MsgStoreUID = *((LPMAPUID)pRow->lpProps[iSvcUID].Value.bin.lpb);
    if(strcmp(pRow->lpProps[iSvcName].Value.lpszA, pszMsgService) == 0)
    {
        // Configuring PST Message Store
        ULONG count = 0;
        const ULONG nProps = 2;
        SPropValue rgval[nProps];
```

```

rgval[count].ulPropTag = PR_DISPLAY_NAME_W;
rgval[count].Value.lpszW = const_cast<wchar_t*>(storeDisplayName.c_str());

rgval[count].ulPropTag = PR_PST_PATH;
rgval[count].Value.lpszA = const_cast<char*>(path.c_str());

// Writing parameters
m_pSvcAdmin->ConfigureMsgService( &m_MsgStoreUID, 0, 0, nProps, rgval);
}
}

```

If everything is fine, the new PST file is created. It has the **path** path and the **storeDisplayName** display name (the Microsoft Outlook property).

The last thing that you should do is to get the object, which will be used for access to the so called Message Store.

[Hide](#) [Copy Code](#)

```

MAPILogonEx(0, m_profileName, NULL, MAPI_NEW_SESSION |
    MAPI_EXTENDED | MAPI_NO_MAIL | MAPI_TIMEOUT_SHORT, &m_pses);

MapiTable ptable;
m_pses->GetMsgStoresTable(0, &ptable);

SizedSPropTagArray(3, columns) = { 3, { PR_DEFAULT_STORE, PR_ENTRYID, PR_DISPLAY_NAME } };

HrQueryAllRows(ptable, (LPSPPropTagArray) &columns, NULL, NULL, 0, &prows);

IMsgStore * msgStore = 0;
m_pses->OpenMsgStore(0, rows->aRow[0].lpProps[1].Value.bin.cb,
    (LPENTRYID)rows->aRow[0].lpProps[1].Value.bin.lpb, NULL, MDB_WRITE |
    MAPI_DEFERRED_ERRORS | MAPI_BEST_ACCESS | MDB_NO_MAIL, &msgStore);

```

As a result, we receive the **IMsgStore * msgStore**, which will be used in practically all other MAPI calls.

Data Writing

The Creation of the Folders Hierarchy

When writing the message, you should define its path like *folder1\folder2\folder3*. Such path defines the location in the database hierarchy, where the message will be placed. To create the folder, you should open the root folder in the database and then create the required one.

So, first you should find the root folder:

[Hide](#) [Copy Code](#)

```
COM::AutoPtr<SPropValue,MapiEraser> pVal_EID;
::HrGetOneProp(store->GetMsgStore(),PR_IPM_SUBTREE_ENTRYID,&pVal_EID);

// Open the IPM subtree folder
ULONG ulObjType = 0;
MapiFolder currentFolder;
store->GetMsgStore()->OpenEntry(pVal_EID->Value.bin.cb,
    (LPENTRYID)pVal_EID->Value.bin.lpb,NULL,0x10,&ulObjType,
    reinterpret_cast<IUnknown**>(&currentFolder));
```

The `store->GetMsgStore()` returns the **Message Store**. The previous part of the article dedicated to the initialization was aimed to get it. As a result, you obtain the **currentFolder** object, which refers to the root directory of the database.

To create the subdirectory, perform the following:

[Hide](#) [Copy Code](#)

```
currentFolder->CreateFolder(FOLDER_GENERIC,
    (LPTSTR)name.c_str(), NULL, NULL, 1 | OPEN_IF_EXISTS | MAPI_UNICODE, &folder);
```

where **name** is the name of the subdirectory and **folder** is the object, which will refer to the created folder. The names of flags, which are passed to the method, are informative enough.

All created objects, which represent folders, are saved in the implementation of the library in order to improve performance. Though there is nothing to prevent you from opening them each time you write the message for memory saving.

Writing the Message

Some message abstraction is used in the library. For simplicity, it is represented as the structure with the set of properties. To simplify the representation of the message structure, it is divided into several logical blocks. These are the common properties (flags, subject, sender), recipients, dates, and message bodies (in this article, it is the text body). Other types of bodies (HTML and RTF) and the attachments are not discussed in this article.

The Creation of the Message

The creation of the message is performed by the call of the corresponding method from the object, which represents the folder, in which the message should be created. In the example, it looks like the following:

[Hide](#) [Copy Code](#)

```
fld->GetFolder()->CreateMessage(0,0,&pMessage);
```

As a result of the successful call, we get the pointer to the message object.

Writing the Message Properties

All message properties are written in the same way. First, the properties array is created, where properties are written as pairs – tag and value. Here is an example of writing several properties:

[Hide](#) [Copy Code](#)

```
std::vector<SPropValue> propArray;
SPropValue currentProp;

currentProp.ulPropTag = PR_MESSAGE_CLASS_W;
currentProp.Value.lpszW = (LPWSTR)pMsg->szMsgClass.c_str();
propArray.push_back(currentProp);
//Message flags
currentProp.ulPropTag = PR_MESSAGE_FLAGS;
currentProp.Value.ul = pMsg->flags;
propArray.push_back(currentProp);
```

After the properties array is filled, you should write it to the message. It is performed in the following way:

[Hide](#) [Copy Code](#)

```
if (propArray.size() != 0)
{
    COM::AutoPtr<SPropProblemArray, MapiEraser> Problems;
    return pMessage->SetProps((ULONG)propArray.size(), &propArray.at(0), &Problems);
}
```

All necessary properties of the message are written in the same way. For more information, see the library code. Any other non-standard properties, which are not used in the library, can be added in a similar way. You just should find its type and tag in MSDN.

Writing the Message Recipients

Message recipients are written in a rather different way than other properties. There is a separate **ModifyRecipients** method for writing recipients in the message object. This method is called in the following way:

[Hide](#) [Copy Code](#)

```
//Filling property array
std::vector<SPropValue> propArray;
SPropValue currentProp;
std::wstring recipientDisplayName = !itRecip->name.empty() ?
    itRecip->name : itRecip->address;
currentProp.ulPropTag = PR_DISPLAY_NAME_W;
currentProp.Value.lpszW = (LPWSTR)recipientDisplayName.c_str();
propArray.push_back(currentProp);

currentProp.ulPropTag = PR_EMAIL_ADDRESS_W;
currentProp.Value.lpszW = (LPWSTR)itRecip->address.c_str();
propArray.push_back(currentProp);

currentProp.ulPropTag = PR_RECIPIENT_TYPE;
```

```

currentProp.Value.ul = itRecip->type;
propArray.push_back(currentProp);

//Filling special structure for changing recipients by property array
ADRLIST adrList = { 0 };
adrList.cEntries = 1;
adrList.aEntries[0].rgPropVals = &propArray.front();
adrList.aEntries[0].ulReserved1 = 0;
adrList.aEntries[0].cValues = (ULONG)propArray.size();
//Making changes in message object
pMessage->ModifyRecipients( MODRECIP_ADD, &adrList );

```

Writing the Message Body

The message body is often represented as a rather fair-sized data. That is why it is written not as a property, but as a so called “**stream**”. First, you should create (open) a **stream** in order to write data to the message. This procedure is implemented in the library in the **CPstWriter::WriteMessageStream** method. Its code looks like the following:

[Hide](#) [Copy Code](#)

```

long CPstWriter::WriteMessageStream
    (IMessage * pMessage, const char * data, size_t dataSize, unsigned long streamTag)
{
    ...
    MapiStream pStream;
    pMessage->OpenProperty (streamTag, &IID_IStream, STGM_WRITE,
        MAPI_CREATE | MAPI_MODIFY, ( LPUNKNOWN* ) & pStream);

    ULONG ulWritten = 0;
    pStream->Write (data, (ULONG)dataSize, &ulWritten);

    pStream->Commit( STGC_DEFAULT );
    ...
}

```

It means that you just create a **stream** with the tie to a definite **streamTag**. After this, the writing of data to this **stream** is performed and changes are committed.

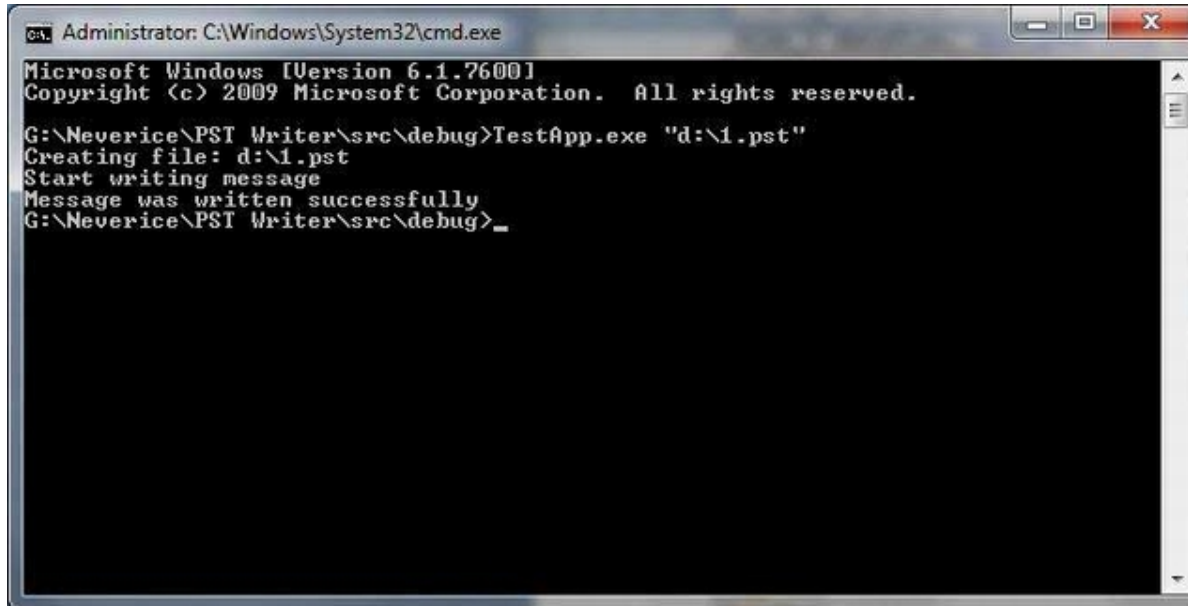
To write the text (unicode) message body, you should call the mentioned above method with the **streamTag = PR_BODY_W** parameter.

How to Build and Run this Solution

1. Use Microsoft Visual Studio 2005 or later to open *PstWriter.sln* in the src folder.
2. Build the solution.
3. Install Microsoft Outlook 2007 or later (it provides Unicode MAPI) for the proper work.

Testing

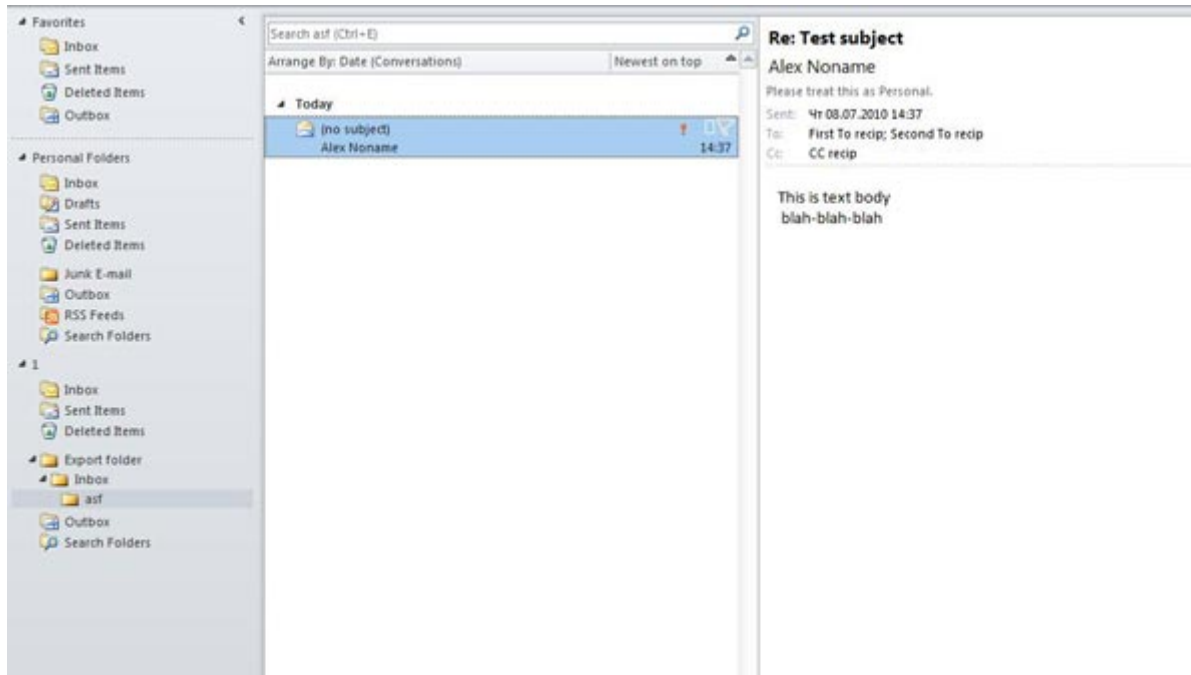
1. Build the solution using the instructions above.
2. Run *TestApp.exe* with the command line parameter, which defines the path to the result PST file. For example, *TestApp.exe "D:\1.pst"*.



```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

G:\Neverice\PST Writer\src\debug>TestApp.exe "d:\1.pst"
Creating file: d:\1.pst
Start writing message
Message was written successfully
G:\Neverice\PST Writer\src\debug>_
```

3. As a result, if there were no errors, a file is created in the defined folder.
4. If you open it in Microsoft Outlook, you see the following:



As it can be seen from the screenshot, the following objects and properties were successfully created:

1. The folders hierarchy (*Result\Inbox\asf*).
2. The message with To, CC, Bcc, and From fields, all dates, subject, and the text body.
3. All recipients (except the mailing address) also include the display name.
4. The message is marked as read, private, and important.

Supported Windows Versions

- Windows XP (SP3)
- Windows 7 (x86, x64)

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

Share

TWITTER

FACEBOOK

About the Authors



Apriorit Inc

Apriorit Inc.
Hungary 🇭🇺

This member doesn't quite have enough reputation to be able to display their biography and homepage.

Group type: Organisation

[32 members](#)



Alexander Maximov

Software Developer apriorit
Ukraine 🇺🇦

No Biography provided

You may also be interested in...

Pro

[Using PDFs to Manage and Share Content](#)

Pro

[The Offline-First Approach to Mobile App Development](#)



How to List your Outlook PST Details (file name/location, PST (old/new) version, size, etc.)



Outlook PST Email Extraction



The *AdES Collection, Part 3: PADES for Windows in C++



Speed up ASP.NET Core WEB API application. Part 2

Comments and Discussions






















You must **Sign In** to use this message board.



Spacing **Relaxed** Layout **Normal** Per page **25** **Update**

First Prev Next

Writing the header	wombleme	29-Oct-12 14:14
Managed C# wrapper ?	Stef Heyenrath	11-Jan-12 21:09
Re: Managed C# wrapper ?	Vladimir Dorodov	20-Dec-12 7:49
Message Closed		28-Apr-14 8:16

 MAPIInitialize fails when using Outlook 2010 64 on 64 bit Windows 7 	 Stef Heyenrath	9-Jan-12 3:25
 Re: MAPIInitialize fails when using Outlook 2010 64 on 64 bit Windows 7 	 Alexander Maximov	11-Jan-12 20:43
 Re: MAPIInitialize fails when using Outlook 2010 64 on 64 bit Windows 7 	 Stef Heyenrath	11-Jan-12 21:08
 Issue in creating .MSG file 	 Ajit41807	11-Sep-12 0:35
 ConfigureMsgService error 	 tiutababo	7-Apr-11 17:36
 Oooops.... 	 persian music	5-Aug-10 12:07
 Re: Oooops.... 	 Apriorit Inc	6-Aug-10 0:06

Last Visit: 31-Dec-99 18:00 Last Update: 15-Oct-18 2:54

[Refresh](#)**1**

 General
  News
  Suggestion
  Question
  Bug
  Answer
  Joke
  Praise
  Rant
  Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

[Permalink](#) |
 [Advertise](#) |
 [Privacy](#) |
 [Cookies](#) |
 [Terms of Use](#) |
 Mobile
 Web01-2016 | 2.8.180920.1 | Last Updated 6 Aug 2010

Select Language ▼

Layout: [fixed](#) | [fluid](#)

Article Copyright 2010 by Apriorit Inc, Alexander Maximov
 Everything else Copyright © [CodeProject](#), 1999-2018