COM API для программистов

Обновлено 2 года назад

Простой пример-введение

Хитрости

- Почтовый индекс в адресной строке
- Загрузка базы данных
- Модальные всплывающие окошки
- Разные версии программы
- Собираем всё вместе

Описание функций

- Печать конвертов
- Печать уведомлений
- Печать почтовых реестров
- Печать описей вложений
- Вспомогательные методы

Используемые форматы данных

- Данные для печати конвертов и уведомлений
- Данные для печати почтовых реестров
- Данные для печати описей вложений

Примеры использования

- Примеры вызова из jscript
- Примеры вызова из excel
- Примеры вызова из 1С

Описание на языке idl

В этом разделе описано, как можно вызывать функции программы «Печать конвертов!» из других программ.

Простой пример-введение

Для начала проиллюстрируем, как можно просто вызвать печать конверта из других программ:

Создадим файл *test.js* со следующим текстовым содержимым:

```
pepro= WScript.CreateObject("PEPro.Application");

pepro.PrintEnvelope
  ("Иван",  "г Иваново, Ивановская 4", "111111",
   "Марья", "г Мариуполь, Ленина 2",   "222222");
```

Разберём, как работает этот скрипт.

Сначала необходимо создать экземпляр программы «Печать конвертов!», что выполняется в первой команде CreateObject ("PEPro.Application"). Полученный экземпляр сохраняется для дальнейшего использования.

Далее вызывается непосредственно функция PrintEnvelope печати конверта для письма от Ивана к Марье.

- Иван указывает обратный адрес "г Иваново, Ивановская 4", почтовый индекс "111111".
- Письмо пишется на адрес Марьи "г Мариуполь, Ленина 2", почтовый индекс "222222".

Просто запустив созданный файл, мы должны получить открытую форму печати конверта (возможно, в зависимости от регистрации программы, после «формы ворчания»).

Далее, того же самого результата можно добиться, собрав данные для конверта в один xml-аргумент и воспользовавшись методом PrintFromXmlEnvelopes:

```
pepro= WScript.CreateObject("PEPro.Application");

pepro.PrintFromXmlEnvelopes
(
"<EnvelopesData>" +
"  <EnvelopeData>" +
"    <From>" +
"      <Name>Иван</Name>" +
"      <Address>г Иваново, Ивановская 4</Address>" +
"      <ZipCode>111111</ZipCode>" +
"    </From>" +
```

```
"    <To>" +
"      <Name>Марья</Name>" +
"      <Address>г Мариуполь, Ленина 2</Address>" +
"      <ZipCode>CA 90048</ZipCode>" +
"    </To>" +
"  </EnvelopeData>" +
"</EnvelopesData>"
);
```

Конечно, возникает вопрос, зачем делать сложно то, что проще простого. Понять «зачем» можно из следующего примера, который иллюстрирует, как Иван может разом писать несколько писем. Например, помимо Марьи можно заодно распечатать конверт и для Анжелины:

```
pepro= WScript.CreateObject("PEPro.Application");

pepro.PrintFromXmlEnvelopes
(
"<EnvelopesData>" +
"  <EnvelopeData>" +
"    <From>" +
"      <Name>Иван</Name>" +
"      <Address>г Иваново, Ивановская 4</Address>" +
"      <ZipCode>111111</ZipCode>" +
"    </From>" +
"    <To>" +
"      <Name>Марья</Name>" +
"      <Address>г Мариуполь, Ленина 2</Address>" +
"      <ZipCode>CA 90048</ZipCode>" +
"    </To>" +
"  </EnvelopeData>" +
"  <EnvelopeData>" +
"    <From>" +
"      <Name>Иван</Name>" +
"      <Address>г Иваново, Ивановская 4</Address>" +
"      <ZipCode>111111</ZipCode>" +
"    </From>" +
"    <To>" +
"      <Name>Angelina</Name>" +
"      <Address>1901 Ave. of the Stars # 680\r\n" +
" Los Angeles, CA 90067-6008\r\n" +
" USA</Address>" +
"    </To>" +
"  </EnvelopeData>" +
"</EnvelopesData>"
);
```

С ростом количества связей Ивана может получиться так, что неудобно собирать данные для печати конвертов в виде строки в памяти. Для этого случая предусмотрена возможность использования заранее подготовленного файла, содержащего все данные.

Такой режим поддерживается при помощи метода PrintFromFileEnvelopes:

```
pepro.PrintFromFileEnvelopes(path + "\\printenvelopes.xml");
```

Хитрости

Почтовый индекс в адресной строке

Некоторые информационные системы хранят адреса одной строкой вместе с почтовым индексом (например 1С v 7.7).

В то же время, описываемое API предполагает передачу почтового индекса отдельным полем.

Чтобы разрешить это противоречие и упростить интеграцию, в API добавлен специальный логический флажок «Выкусывать почтовый индекс прямо из строки адреса». Для управления этим флажком предназначены методы GetFlagParseZipIndexFromAddress и SetFlagParseZipIndexFromAddress.

Таким образом, всё, что нужно сделать, для того чтобы распечатать почтовый документ, указав почтовый индекс прямо в строке адреса — это установить описываемый флажок непосредственно перед вызовом соответствующей функции.

```
pepro= WScript.CreateObject("PEPro.Application");

pepro.SetFlagParseZipIndexFromAddress(1);
pepro.PrintEnvelope
  ("Иван",  "111111, г Иваново, Ивановская 4", "",
   "Марья", "222222, г Мариуполь, Ленина 2",   "");
```

Загрузка базы данных

В базе данных хранятся все данные о контрагентах, рассылках, письмах, реестрах и пользовательских бланках почтовых документов (конвертов, уведомлений, реестров и описей вложений). Соединение с базой данных занимает определённое время. Кроме того, при соединении могут обнаруживаться проблемы (база отсутствует или имеет неправильную версию).

В некоторых случаях, для некоторых функций база данных не нужна (например, печать на стандартных бланках, без использования пользовательских шаблонов). В таких случаях можно ускорить работу, не загружая базу данных. Поэтому при инициализации API, база данных по умолчанию не загружается.

Для загрузки базы данных и проверки того, что она загружена, в API добавлены методы DatabaseIsLoaded и SafeLoadDatabase. В частности, если необходимо использовать

пользовательские шаблоны документов, рекомендуется вызывать просто метод SafeLoadDatabase перед соответствующей функцией.

```
pepro= WScript.CreateObject("PEPro.Application");

pepro.SafeLoadDatabase();
pepro.PrintEnvelope
  ("Иван",  "г Иваново, Ивановская 4", "111111",
   "Марья", "г Мариуполь, Ленина 2",   "222222");
```

Модальные всплывающие окошки

Программа «Печать конвертов!» предоставляет свой COM API как «out-of-process OLE Server» (т.е. COM объект реально располагается в адресном пространстве своего собственного процесса). Многие методы API предполагают открытие диалоговых окошек в «модальном режиме». В связи с этим возникает 2 проблемы:

1. «всплывание» окошек COM объекта поверх окошек вызывающего процесса.
2. «модальный» режим работы окошек для вызывающего процесса.

Активация выскакивающих окошек

Компания Microsoft неустанно работает над совершенствованием своих операционных систем семейства Windows и в частности стремится сделать их более безопасными, понятными и предсказуемыми для пользователей.

С этим стремлением связаны регламенты оконной системы по работе со «всплывающими» окошками. Очевидно, приложение, с которым работает пользователь, вольно само решать, какие окошки открывать. А вот если окошко захочет открыть приложение, с которым пользователь в настоящее время не работает, да ещё и открыть его поверх других окошек, тут могут быть проблемы. В частности, такое поведение может быть характерно для всяких вредоносных и навязчивых программ. Поэтому, в разных версиях ОС семейства Windows всплывающие окошки могут «всплывать» по-разному. И как мы помним, в нашем случае именно такая ситуация (COM объект пытается открывать новые окошки из отдельного процесса).

За открытие окошка поверх всех других окошек других приложений отвечает функция Win API SetForegroundWindow. Документация для неё гласит:

The system restricts which processes can set the foreground window. A process can set the foreground window only if one of the following conditions is true:

- The process is the foreground process.
- The process was started by the foreground process.
- The process received the last input event.
- There is no foreground process.
- The foreground process is being debugged.
- The foreground is not locked (see LockSetForegroundWindow).
- The foreground lock time-out has expired (see SPI_GETFOREGROUNDLOCKTIMEOUT in SystemParametersInfo).
- No menus are active.

An application cannot force a window to the foreground while the user is working with another window. Instead, Foreground and Background Windows will activate the window (see SetActiveWindow) and call the function to notify the user.
A process that can set the foreground window can enable another process to set the foreground window by calling the AllowSetForegroundWindow function. The process specified by dwProcessId loses the ability to set the foreground window t he next time the user generates input, unless the input is directed at that process, or the next time a process calls AllowSetForegroundWindow, unless that process is specified.
The foreground process can disable calls to SetForegroundWindow by calling the LockSetForegroundWindow function.

Другими словами, в переводе на русский, позаботиться о том, чтобы окошки вызываемого процесса открывались «поверх всего», должен вызывающий процесс. В режиме работы с «модальными» окнами, это может быть проблематично, потому что «вызывающий» процесс владеет потоком управления тогда, когда никаких «вызываемых» окошек ещё нет.

Чтобы всё-таки вызывающий процесс смог активизировать окошки программы «Печать конвертов!», в API были добавлены функции для работы со специальным «фиктивным» немодальным окошком. OpenDummyFormToSetAsForeground и CloseDummyFormAndSetAsForeground. Идея заключается в том, чтобы открыть предварительно «фиктивное» немодальное окошко и вывести его на передний план. По окончании работы нужных функций наоборот закрыть «фиктивное» окошко, причём так, чтобы при закрытии на передний план перешли окошки вызывающего процесса.

```
pepro= WScript.CreateObject("PEPro.Application");
shell= WScript.CreateObject("WScript.Shell");

current_foreground_win= pepro.OpenDummyFormToSetAsForeground();
shell.AppActivate("Печать конвертов!")
pepro.PrintEnvelope
  ("Иван",  "г Иваново, Ивановская 4", "111111",
   "Марья", "г Мариуполь, Ленина 2",   "222222");
pepro.CloseDummyFormAndSetAsForeground(current_foreground_win);
```

К сожалению, описанный способ не является «серебряной пулей» именно по той причине, что Microsoft меняет правила для всплывающих окон от версии к версии. В некоторых ситуациях это может привести к тому, что окошко всё-таки откроется под текущим. В таком случае на нужное окошко пользователю придётся переключиться вручную, например, кликнув по мигающей закладке панели задач.

Модальный режим вызова окошек

Статья из базы знаний Micrisoft № 172059 «How To Show a Modal Form from an OLE Server DLL» гласит:

> Showing a modal form from an out-of-process OLE Server is generally not recommended. One reason is that the modal form may be displayed behind the client program, making it difficult for the user to see that it needs to be addressed. Another problem is that while the method of the OLE Server is still processing, the client program may not handle Paint messages, which can cause the client screen to become cluttered by the images of other windows that have overlaid it.

В переводе на русский, помимо проблемы «всплывания», вызов модальных окошек из «out-of-process OLE Server» влечёт за собой дополнительно проблему нарушения нормального цикла обработки сообщений в вызывающем процессе. Это может привести к появлению «артефактов» на окошке «вызывающего» приложения.

Проблема связана с тем, что вызов метода с модальными окошками обычно происходит из обработчика событий окна вызывающего приложения, и этот метод (соответственно, и сам обработчик) заканчивается только тогда, когда пользователь закрывает модальное окно.

Для того, чтобы преодолеть эту проблему, в COM API программы добавлен специальный логический флажок «Запускать формы в модальном режиме». Для управления этим флажком предназначены методы GetFlagShowFormsAsModal и SetFlagShowFormsAsModal. Если флажок установлен, метод COM API завершается, не дожидаясь закрытия модального окошка. Для того чтобы воспользоваться «не модальным» режимом, достаточно просто установить флажок перед вызовом соответствующего метода.

```
pepro= WScript.CreateObject("PEPro.Application");

pepro.SetFlagShowFormsAsModal(0);
pepro.PrintEnvelope
  ("Иван",  "г Иваново, Ивановская 4", "111111",
   "Марья", "г Мариуполь, Ленина 2",   "222222");
```

Разные версии программы

Программа «Печать конвертов!» является живым, постоянно развивающимся проектом. Регулярно выпускаются новые версии программы. В новых версиях совершенствуется и COM API: добавляются новые функции, может несколько видоизменяться поведение старых. Программа, использующая COM API, должна иметь возможность учитывать эти особенности разных версий.

Начиная с версии программы 2.2 в COM API добавлены два новых метода: GetProductVersion и GetAPIVersion. В описаниях функций COM API указывается, в какой версии она была добавлена. Чтобы написать код, сохраняющий свою работоспособность для разных версий программы «Печать конвертов!», этих функций достаточно.

```
pepro= WScript.CreateObject("PEPro.Application");

function pepro_api_version(pepro)
{
  var res= 0;
  try
  {
    res= pepro.GetAPIVersion();
  }
  catch (e)
  {
  }
  return res;
}

if (pepro_api_version(pepro)>0)
  pepro.SetFlagShowFormsAsModal(0);
pepro.PrintEnvelope
  ("Иван",  "г Иваново, Ивановская 4", "111111",
   "Марья", "г Мариуполь, Ленина 2",   "222222");
```

Собираем всё вместе

Резюмируя описания хитростей API, можно привести пример, который учитывает их все, т.е. печатает:

- на любой версии программы, не старее 2.1,
- используя «пользовательский» бланк,
- «выкусывая» почтовый индекс прямо из строки адреса,
- запуская окно печати в «немодальном» режиме,
- обеспечивая «всплывание» окна печати.

```
var pepro= WScript.CreateObject("PEPro.Application");

var api_version= 0;
try
{
  api_version= pepro.GetAPIVersion();
}
catch (e)
{
}

var current_foreground_win;
if (api_version>0)
{
  pepro.SafeLoadDatabase();
  pepro.SetFlagShowFormsAsModal(0);
  pepro.SetFlagParseZipIndexFromAddress(1);
  var shell= WScript.CreateObject("WScript.Shell");
```

```
  current_foreground_win= pepro.OpenDummyFormToSetAsForeground();
  shell.AppActivate("Печать конвертов!")
}
pepro.PrintEnvelope
  ("Иван",  "г Иваново, Ивановская 4", "111111",
   "Марья", "г Мариуполь, Ленина 2",   "222222");
if (api_version>0)
{
  pepro.CloseDummyFormAndSetAsForeground(current_foreground_win);
}
```

Описание функций

Печать конвертов

PrintEnvelope — самая простая функция для печати одного конверта

```
  HRESULT PrintEnvelope([in] BSTR from,        // от кого
                        [in] BSTR fromAddress, // откуда
                        [in] BSTR fromZipCode, // почтовый индекс откуда
                        [in] BSTR to,          // кому
                        [in] BSTR toAddress,   // куда
                        [in] BSTR toZipCode);  // почтовый индекс куда
```

Данный метод позволяет легко запустить печать конверта по понятным строковым аргументам для каждой графы конверта.

Смысл аргументов понятен из комментариев.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

PrintFromXmlEnvelopes — печать конвертов по xml описанию

```
  HRESULT PrintFromXmlEnvelopes([in] BSTR xml_text);
```

Данный метод позволяет запустить печать нескольких конвертов по xml описанию.

На вход подаётся непосредственно строка с xml описанием. Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

PrintFromFileEnvelopes — печать конвертов по подготовленному xml файлу

```
HRESULT PrintFromFileEnvelopes([in] BSTR filename);
```

Данный метод позволяет запустить печать нескольких конвертов по xml описанию, сохранённому в файле.

На вход подаётся путь до файла, в котором записаны исходные данные. **Внимание!** Рекомендуется указывать полный путь!

Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

Печать уведомлений

PrintFromXmlNotifications — печать уведомлений по xml описанию

```
HRESULT PrintFromXmlNotifications([in] BSTR xml_text);
```

Данный метод позволяет запустить печать нескольких уведомлений по xml описанию.

На вход подаётся непосредственно строка с xml описанием. Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

PrintFromFileNotifications — печать уведомлений по подготовленному xml файлу

```
HRESULT PrintFromFileNotifications([in] BSTR filename);
```

Данный метод позволяет запустить печать нескольких уведомлений по xml описанию, сохранённому в файле.

На вход подаётся путь до файла, в котором записаны исходные данные. **Внимание!** Рекомендуется указывать полный путь!

Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

Печать почтовых реестров

PrintFromXmlRegisters — печать почтовых реестров по xml описанию

```
HRESULT PrintFromXmlRegisters([in] BSTR xml_text);
```

Данный метод позволяет запустить печать нескольких почтовых реестров по xml описанию.

На вход подаётся непосредственно строка с xml описанием. Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

PrintFromFileRegisters — печать почтовых реестров по подготовленному xml файлу

```
HRESULT PrintFromFileRegisters([in] BSTR filename);
```

Данный метод позволяет запустить печать нескольких почтовых реестров по xml описанию, сохранённому в файле.

На вход подаётся путь до файла, в котором записаны исходные данные. **Внимание!** Рекомендуется указывать полный путь!

Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

GetRegisterTemplateNames — получить список имён пользовательских шаблонов реестров

```
HRESULT GetRegisterTemplateNames([out, retval] SAFEARRAY(BSTR)* pRetVal);
```

Данный метод позволяет получить список названий пользовательских шаблонов реестров, доступных в программе.

Метод не имеет аргументов

Метод возвращает массив строк с именами пользовательских шаблонов реестров.

После вызова метода, о его результатах можно судить, воспользовавшись GetWarningType.

Метод был добавлен специально для интеграции с программой "Помощник Арбитражного Управляющего", в которой используются собственные формы свойств реестров.

Присутствует, начиная с версии 2.2 (API v 1)

PrintFromXmlRegistersWithoutForm — печатать почтовый реестр по xml описанию без открытия формы

```
HRESULT PrintFromXmlRegistersWithoutForm(
                      [in] BSTR xml_text,
                      [in] BSTR nametemplate,
                      [in] BSTR resultFullPath);
```

Данный метод позволяет получить rtf-файл реестра по имени пользовательского шаблона и xml описанию.

На вход подаются аргументы:

Таблица 5.1.

аргумент	**описание**
xml_text	Строка с данными реестра. Формат xml описания приведён в соответствующем разделе.
nametemplate	Строка - имя пользовательского шаблона реестра. Полный список возможных имён можно получить воспользовавшись методом GetRegisterTemplateNames.
resultFullPath	Строка - путь к файлу, в котором должен быть сохранён результат. **Внимание!** Рекомендуется указывать полный путь!

Метод не возвращает никаких значащих данных.

После вызова метода, о его результатах можно судить, воспользовавшись GetWarningType.

Метод был добавлен специально для интеграции с программой "Помощник Арбитражного Управляющего", в которой используются собственные формы свойств реестров.

Присутствует, начиная с версии 2.2 (API v 1)

Печать описей вложений

PrintFromXmlInventories — печать описей вложений по xml описанию

```
HRESULT PrintFromXmlInventories([in] BSTR xml_text);
```

Данный метод позволяет запустить печать нескольких описей вложений по xml описанию.

На вход подаётся непосредственно строка с xml описанием. Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

PrintFromFileInventories — печать описей вложений по подготовленному xml файлу

```
HRESULT PrintFromFileInventories([in] BSTR filename);
```

Данный метод позволяет запустить печать нескольких описей вложений по xml описанию, сохранённому в файле.

На вход подаётся путь до файла, в котором записаны исходные данные. **Внимание!** Рекомендуется указывать полный путь!

Формат xml описания приведён в соответствующем разделе.

Метод не возвращает никаких значащих данных.

Присутствует, начиная с версии 2.1 (API v 0)

Вспомогательные методы

DatabaseIsLoaded — узнать, загружена ли база данных

```
HRESULT DatabaseIsLoaded([out, retval] VARIANT_BOOL* pRetVal);
```

Данный метод позволяет проверить, подгрузил ли COM объект базу данных (*.mdb файл).

В частности от этого зависит, будут ли доступны "пользовательские" бланки конвертов, уведомлений, реестров и описей вложений.

Если возвращает true, значит, база данных загружена.

Описание концепции приведено в разделе "Хитрости / Загрузка базы данных".

Присутствует, начиная с версии 2.2 (API v 1)

SafeLoadDatabase — загрузить базу данных, если она ещё не загружена.

```
HRESULT SafeLoadDatabase([out, retval] VARIANT_BOOL* pRetVal);
```

Данный метод позволяет безопасно "подгрузить" базу данных (*.mdb файл).

В частности от этого зависит, будут ли доступны "пользовательские" бланки конвертов, уведомлений, реестров и описей вложений.

В случае если база данных была загружена ранее, метод не выполняет никаких действий.

Если возвращает true, значит, база данных загружена.

Описание концепции приведено в разделе "Хитрости / Загрузка базы данных".

Присутствует, начиная с версии 2.2 (API v 1)

GetFlagParseZipIndexFromAddress — выкусывать ли почтовый индекс из адреса

```
HRESULT GetFlagParseZipIndexFromAddress([out, retval] VARIANT_BOOL* pRetVal);
```

Данный метод позволяет получить значение логического флага "Выкусывать почтовый индекс прямо из строки адреса".

Если возвращает true, значит, программа будет пытаться "выкусить" почтовый индекс из адресной строки.

Описание концепции приведено в разделе "Хитрости / Почтовый индекс в адресной строке".

Присутствует, начиная с версии 2.2 (API v 1)

SetFlagParseZipIndexFromAddress — выкусывать почтовый индекс из адреса

```
HRESULT SetFlagParseZipIndexFromAddress(
                      [in] VARIANT_BOOL f,
                      [out, retval] VARIANT_BOOL* pRetVal);
```

Данный метод позволяет установить значение логического флага "Выкусывать почтовый индекс прямо из строки адреса".

Единственным аргументом метода является желаемое значение флага.

Если возвращает true, значит, программа будет пытаться "выкусить" почтовый индекс из адресной строки.

Описание концепции приведено в разделе "Хитрости / Почтовый индекс в адресной строке".

Присутствует, начиная с версии 2.2 (API v 1)

GetFlagShowFormsAsModal — запускать ли формы в модальном режиме

```
HRESULT GetFlagShowFormsAsModal([out, retval] VARIANT_BOOL* pRetVal);
```

Данный метод позволяет получить значение логического флага "Запускать формы в модальном режиме".

Если возвращает true, значит, программа будет запускать формы в модальном режиме.

Описание концепции приведено в разделе "Хитрости / Модальный режим вызова окошек".

Присутствует, начиная с версии 2.2 (API v 1)

SetFlagShowFormsAsModal — запускать формы в модальном режиме

```
HRESULT SetFlagShowFormsAsModal(
                      [in] VARIANT_BOOL f,
                      [out, retval] VARIANT_BOOL* pRetVal);
```

Данный метод позволяет установить значение логического флага "Запускать формы в модальном режиме".

Единственным аргументом метода является желаемое значение флага.

Если возвращает true, значит, программа будет запускать формы в модальном режиме.

Описание концепции приведено в разделе "Хитрости / Модальный режим вызова окошек".

Присутствует, начиная с версии 2.2 (API v 1)

OpenDummyFormToSetAsForeground — открыть подготовительную форму

```
HRESULT OpenDummyFormToSetAsForeground([out, retval] BSTR* pRetVal);
```

Данный метод позволяет открыть специальное фиктивное маленькое "подготовительное" немодальное окошко, чтобы иметь возможность перенести его на передний план.

Метод не принимает никаких параметров.

Возвращает строку - заголовок окна, которое было на переднем плане на момент "переключения" (как правило - заголовок главного окна вызывающего процесса).

Описание концепции приведено в разделе "Хитрости / Активация выскакивающих окошек".

Присутствует, начиная с версии 2.2 (API v 1)

CloseDummyFormAndSetAsForeground — закрыть подготовительную форму

```
HRESULT CloseDummyFormAndSetAsForeground([in] BSTR process);
```

Данный метод позволяет закрыть специальное фиктивное маленькое "подготовительное" немодальное окошко, и заставить его перенести на передний план перед своим закрытием другое.

Единственный аргумент строка - заголовок окна, которое нужно перевести на передний план.

Метод не возвращает никаких значащих данных.

Описание концепции приведено в разделе "Хитрости / Активация выскакивающих окошек".

Присутствует, начиная с версии 2.2 (API v 1)

GetProductVersion — получить версию установленной программы

```
HRESULT GetProductVersion([out, retval] BSTR* pRetVal);
```

Данный метод позволяет получить версию установленной программы в виде строки.

Метод не принимает никаких параметров.

Метод возвращает версию программы в виде строки (например "2.1.0.6").

Описание концепции приведено в разделе "Хитрости / Разные версии программы".

Присутствует, начиная с версии 2.2 (API v 1)

GetAPIVersion — получить версию COM API

```
HRESULT GetAPIVersion([out, retval] long* pRetVal);
```

Данный метод позволяет получить версию COM API виде числа.

Метод не принимает никаких параметров.

Метод возвращает версию программы в виде числа (например, для версии "2.2" вернётся 1).

Описание концепции приведено в разделе "Хитрости / Разные версии программы".

В отличие от метода GetProductVersion данный метод удобнее в использовании, потому что:

1. целое число меньше по размеру (функция работает быстрее)
2. операции сравнения с целым числом быстрее (не нужно сложное сравнение версий с учётом всяких минорных изменений и патчей)
3. предполагается, что COM API будет меняться реже, чем сама программа и нумероваться версии будут простой сквозной нумерацией.

Присутствует, начиная с версии 2.2 (API v 1)

GetWarningType — получить код успешности предыдущего вызова

```
   HRESULT GetWarningType([out, retval] long* pRetVal);
```

Данный метод позволяет получить код успешности предыдущего вызова метода виде числа.

В настоящий момент работает только для методов PrintFromXmlRegistersWithoutForm и GetRegisterTemplateNames.

Метод не принимает никаких параметров.

Метод возвращает число, по которому можно определить, нужно ли сообщать пользователю о каких то проблемах.

Значения, которые возвращает метод:

- 0 - печать реестра или получение списка шаблонов реестров выполнено без ошибок
- 1 - база данных отсутствует или некорректна
- 2 - произошла ошибка при чтении xml-данных
- 3 - не удалось создать объект word
- 4 - отсутствует файл шаблона

Присутствует, начиная с версии 2.2 (API v 1)

Используемые форматы данных

Данные для печати конвертов и уведомлений

Для печати конвертов и уведомлений используется один и тот же формат данных:

```
<EnvelopesData>
    <EnvelopeData>
        <From>
           <Name>От кого..</Name>
           <Address>Откуда..</Address>
           <ZipCode>Почтовый индекс откуда..</ZipCode>
        </From>
        <To>
           <Name>Кому..</Name>
           <Address>Куда..</Address>
           <ZipCode>Почтовый индекс куда..</ZipCode>
        </To>
    </EnvelopeData>
    ..
</EnvelopesData>
```

Смысл элементов ясен по их наименованию:

EnvelopesData
: Элемент, содержащий в себе исходные данные для печати всех конвертов или уведомлений. Может содержать внутри себя много элементов EnvelopeData.

EnvelopeData
: Элемент, содержащий в себе исходные данные (об отправителе и получателе) для печати одного конверта или уведомления.

From
: Элемент, содержащий данные об отправителе ("От кого", "Откуда" и почтовый индекс отправителя).

To
: Элемент, содержащий данные о получателе ("Кому", "Куда" и почтовый индекс получателя).

Name
: Имя контрагента (в зависимости от места - поле "Кому" или "От кого").

Address
: Адрес контрагента единым текстом (в зависимости от места - поле "Куда" или "Откуда").

ZipCode
: Почтовый индекс - в зависимости от места для адреса "Куда" или "Откуда".

Данные для печати почтовых реестров

Для печати почтовых реестров используется следующий формат данных:

```
<RegistersData>
    <RegisterData>
        <From>
           <Name>От кого..</Name>
           <Address>Откуда..</Address>
           <ZipCode>Почтовый индекс откуда..</ZipCode>
        </From>
        <Packages>
            <Package>
                <To>
                    <Name>Кому..</Name>
                    <Address>Куда..</Address>
                    <ZipCode>Почтовый индекс куда..</ZipCode>
                </To>
            </Package>
            ..
        </Packages>
    </RegisterData>
    ..
</RegistersData>
```

Смысл элементов ясен по их наименованию:

RegistersData

Элемент, содержащий в себе исходные данные для печати всех почтовых реестров. Может содержать внутри себя много элементов RegisterData.

RegisterData
: Элемент, содержащий в себе исходные данные (об отправителе и получателях) для печати одного почтового реестра.

From
: Элемент, содержащий данные об отправителе ("От кого", "Откуда" и почтовый индекс отправителя).

Packages
: Элемент, содержащий данные для строк почтового реестра. Может содержать внутри себя много элементов Package - по одному на каждую строку.

Package
: Элемент, содержащий данные для одной строки почтового реестра.

To
: Элемент, содержащий данные о получателе ("Кому", "Куда" и почтовый индекс получателя).

Name
: Имя контрагента (в зависимости от места - поле "Кому" или "От кого").

Address
: Адрес контрагента единым текстом (в зависимости от места - поле "Куда" или "Откуда").

ZipCode
: Почтовый индекс - в зависимости от места для адреса "Куда" или "Откуда"

Данные для печати описей вложений

Для печати описей вложений используется следующий формат данных:

```
<InventoriesData>
    <InventoryData>
        <From>
            <Name>От кого..</Name>
        </From>
        <To>
            <Name>Кому..</Name>
            <Address>Куда..</Address>
        </To>
        <InventoryItems>
            <InventoryItem>
                <ItemName>Вложение 1</ItemName>
            </InventoryItem>
            ..
        </InventoryItems>
    </InventoryData>
    ..
</InventoriesData>
```

Смысл элементов ясен по их наименованию:

InventoriesData
: Элемент, содержащий в себе исходные данные для печати всех описей вложений. Может содержать внутри себя много элементов InventoryData.

InventoryData
: Элемент, содержащий в себе исходные данные (об отправителе, получателе и вложениях) для печати одной описи вложения.

From
: Элемент, содержащий данные об отправителе ("От кого", "Откуда" и почтовый индекс отправителя).

InventoryItems
: Элемент, содержащий данные для строк описи вложений. Может содержать внутри себя много элементов InventoryItem - по одному на каждую строку.

InventoryItem
: Элемент, содержащий данные для одной строки описи вложений.

ItemName
: Элемент, содержащий текст-описание вложения.

To
: Элемент, содержащий данные о получателе ("Кому", "Куда").

Name
: Имя контрагента (в зависимости от места - поле "Кому" или "От кого").

Address
: Адрес контрагента единым текстом (поле "Куда").

Примеры использования

Примеры вызова из jscript

Примеры вызовов из jscript можно найти во введении к главе.

Примеры вызова из excel

Вызов из Excel структурно не отличается от вызова из jscript, хотя в этом случае используется другой язык программирования — VBA.

```
Sub PrintEnvelope()
    Dim pepro As Object
    Set pepro = CreateObject("PEPro.Application")
    pepro.PrintEnvelope "Иван", "г Иваново, Ивановская 4", "111111", _
                        "Марья", "г Мариуполь, Ленина 2", "222222"
End Sub
```

Примеры вызова из 1С

Минимальный пример вызова из 1С также практически полностью повторяет jscript:

```
Перем pepro;
pepro= CreateObject("PEPro.Application");
pepro.PrintEnvelope
  ("Иван",  "г Иваново, Ивановская 4", "111111",
   "Марья", "г Мариуполь, Ленина 2",   "222222");
```

Описание на языке idl

```
// Generated .IDL file (by the OLE/COM Object Viewer)
//
// typelib filename: PrintEnvelopePro.tlb

[
  uuid(22BC298D-DF0E-3C29-B589-7A50D3A8B871),
  version(2.2),
  helpstring("\xFFFFFFCF\xFFFFFFE5ча\xFFFFFFF2ь \xFFFFFFEA\xFFFFFFEE\xFFFFFFED\xFFFFFFE2\xFFFFFF
  custom(90883F05-3D28-11D2-8F17-00A0C9A6186D, PrintEnvelopePro, Version=2.2.0.0, Culture=neutra

]
library PrintEnvelopePro
{
    // TLib :     // TLib : Common Language Runtime Library : {BED7F4EA-1A96-11D2-8F08-00A0C9A61
    importlib("mscorlib.tlb");
    // TLib : OLE Automation : {00020430-0000-0000-C000-000000000046}
    importlib("stdole2.tlb");

    // Forward declare all types defined in this typelib
    interface IPEProApplication;

    [
      odl,
      uuid(3A98FA89-27EE-4DDF-8F5D-ABC47EA942E6),
      version(1.0),
      dual,
      oleautomation,
      custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9, PEP.COM.IPEProApplication)

    ]
    interface IPEProApplication : IDispatch {
        [id(0x60020000)]
        HRESULT PrintFromFileEnvelopes([in] BSTR filename);
        [id(0x60020001)]
        HRESULT PrintFromFileNotifications([in] BSTR filename);
        [id(0x60020002)]
        HRESULT PrintFromFileInventories([in] BSTR filename);
        [id(0x60020003)]
        HRESULT PrintFromFileRegisters([in] BSTR filename);
        [id(0x60020004)]
        HRESULT PrintFromXmlEnvelopes([in] BSTR xml_text);
        [id(0x60020005)]
        HRESULT PrintFromXmlNotifications([in] BSTR xml_text);
        [id(0x60020006)]
```

```
        HRESULT PrintFromXmlInventories([in] BSTR xml_text);
        [id(0x60020007)]
        HRESULT PrintFromXmlRegisters([in] BSTR xml_text);
        [id(0x60020008)]
        HRESULT PrintEnvelope(
                        [in] BSTR from,
                        [in] BSTR fromAddress,
                        [in] BSTR fromZipCode,
                        [in] BSTR to,
                        [in] BSTR toAddress,
                        [in] BSTR toZipCode);
        [id(0x60020009)]
        HRESULT OpenDummyFormToSetAsForeground([out, retval] BSTR* pRetVal);
        [id(0x6002000a)]
        HRESULT CloseDummyFormAndSetAsForeground([in] BSTR process);
        [id(0x6002000b)]
        HRESULT DatabaseIsLoaded([out, retval] VARIANT_BOOL* pRetVal);
        [id(0x6002000c)]
        HRESULT SafeLoadDatabase([out, retval] VARIANT_BOOL* pRetVal);
        [id(0x6002000d)]
        HRESULT GetFlagParseZipIndexFromAddress([out, retval] VARIANT_BOOL* pRetVal);
        [id(0x6002000e)]
        HRESULT SetFlagParseZipIndexFromAddress(
                        [in] VARIANT_BOOL f,
                        [out, retval] VARIANT_BOOL* pRetVal);
        [id(0x6002000f)]
        HRESULT GetFlagShowFormsAsModal([out, retval] VARIANT_BOOL* pRetVal);
        [id(0x60020010)]
        HRESULT SetFlagShowFormsAsModal(
                        [in] VARIANT_BOOL f,
                        [out, retval] VARIANT_BOOL* pRetVal);
        [id(0x60020011)]
        HRESULT GetProductVersion([out, retval] BSTR* pRetVal);
        [id(0x60020012)]
        HRESULT GetAPIVersion([out, retval] long* pRetVal);
        [id(0x60020013)]
        HRESULT GetRegisterTemplateNames([out, retval] SAFEARRAY(BSTR)* pRetVal);
        [id(0x60020014)]
        HRESULT PrintFromXmlRegistersWithoutForm(
                        [in] BSTR xml_text,
                        [in] BSTR nametemplate,
                        [in] BSTR resultFullPath);
        [id(0x60020015)]
        HRESULT GetWarningType([out, retval] long* pRetVal);
    };

    [
      uuid(5715286E-F98F-4C1C-AC86-AB943AFAE004),
      version(1.0),
      custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9, PEP.COM.PEProApplication)
    ]
    coclass PEProApplication {
        interface _Object;
```

```
        [default] interface IPEProApplication;
    };
};
```

Версия #0
Виктория Дудина создал 26 October 2018 09:28:22
Виктория Дудина обновил 26 September 2022 14:53:49