Создание универсальных def и lib-файлов для «чужих» dll

by **Erfaren**
http://erfaren.narod.ru
erfaren@rambler.ru

Введение

Проблема заключается в том, что использование стандартных **lib**-файлов от **Майкрософт** иногда приводит к ошибкам компиляции ассемблерного кода (например, сгенерированного дизассемблером **IdaPro**), связанных с отсутствием адекватных определений функций в файлах библиотек. Также нередко бывает ситуация, когда нужных либов просто нет, ни в **MASM32**, ни в **MS Visual Studio**. Скажем, файлы **regapi.lib** или **utildll.lib** и другие, для системных библиотек **Windows**, вы не найдете в стандартной поставке **lib**-файлов.

Кроме того, иногда возникает необходимость вызова функций по ординалам, что фактически сводится к вопросу вызова функции по ее псевдоимени или адресу.

Проблема вызова функций по ординалу в MASM32

Заметим, что конструкция **MASM32**

```
call FuncName
```

фактически является псевдокодом. В некоторых версиях ассемблера псевдокодом является также явный вызов по ординалу функции (ее номеру в таблице экспорта). Например, отладчик **OllyDbg** может для условного выражения

```
call mfc42u.#561
```

написать такой псевдокод

```
01001A03    call <jmp.&mfc42u.#561>                 ; call 01003112
. . .
01003112    jmp dword ptr ds:[<&mfc42u.#561>]    ; jmp dword ptr ds:[1001090]    ; jmp 7F0EB6C0
```

что означает вызов функции из модуля **mfc42u.dll** по ординалу **561**.

Реальный код, эквивалентный этой конструкции, будет

```
call 7F0EB6C0
```

Аналогичный пример, конструкция **MASM32**

```
call MessageBoxA
```

или, в версии «Оли Дебаговой»,

```
0040100E    call <jmp.&user32.MessageBoxA>      ; call 00401022
. . .
00401022    jmp dword ptr ds:[402008]                ; jmp 773C425F
```

эквивалентна реальному выражению

```
call 773C425F
```

что уже непосредственно переводится в шестнадцатеричные команды процессора.

Итак, мы видим, что имена и ординалы не являются родными для процессора, однако компилятор **MASM32** может автоматически перевести имена функций в их адреса, хотя не может сделать это для ординалов функций. Впрочем, это не большая потеря. Если мы не хотим пользоваться явной адресацией вместо вызова по ординалу, то у нас еще остается в запасе назначение псевдоимени функции без имени 🙂. Только сделать это можно в **def**-файле, из которого затем компилируется **lib**-файл. Соответственно, упоминавшиеся уже неадекватные описания функций, можно заменить в нашем **def**-файле на более корректные определения. Таким образом, мы со всей очевидностью приходим к необходимости иметь собственные **def**-файлы для нужных нам **dll** и возможности генерировать такие файлы автоматически.

Проблема универсальности в вызове функций

Еще один вопрос, связанный с универсальностью вызова функций, заключается в желании написать такой **def**-файл, чтобы можно было бы вызывать любое из выражений (без использования дополнительных определений, связанных с директивой **equ**) вида:

```
call MessageBoxA
call MessageBoxA@16
call _imp__MessageBoxA
call _imp__MessageBoxA@16
```

так как на практике встречаются все эти варианты.

Оказывается, подобное решение существует.

Универсальные def-файлы для простой программы

Действительно, создадим следующие файлы определений:

user32.def (где находится функция **MessageBoxA**)

```
LIBRARY "user32.dll"
EXPORTS
MessageBoxA
_MessageBoxA@16 = MessageBoxA
```

и **kernel32.def** (где находится функция **ExitProcess**)

```
LIBRARY "kernel32.dll"
EXPORTS
ExitProcess
_ExitProcess@4 = ExitProcess
```

На их основе построим **lib**-файлы с помощью командного файла **def2lib.bat**

```
SET DEF=Def
SET LIB=Lib

SET FILENAME=kernel32

:: Создание lib файла из def файла
:: Bin\lib /DEF:%DEF%\%FILENAME%.def /OUT:%LIB%\%FILENAME%.lib /MACHINE:X86 > %FILENAME%.
Bin\polib /DEF:%DEF%\%FILENAME%.def /OUT:%LIB%\%FILENAME%.lib /MACHINE:X86 > %FILENAME%.

SET FILENAME=user32

:: Создание lib файла из def файла
:: Bin\lib /DEF:%DEF%\%FILENAME%.def /OUT:%LIB%\%FILENAME%.lib /MACHINE:X86 > %FILENAME%.
Bin\polib /DEF:%DEF%\%FILENAME%.def /OUT:%LIB%\%FILENAME%.lib /MACHINE:X86 > %FILENAME%.
```

Этот командный файл берет файлы **kernel32.def** и **user32.def** из каталога **Def** и создает файлы **kernel32.lib** и **user32.lib** в каталоге **Lib**. Попутно выводятся сообщения компиляции с теми же именами

без расширений. Если эти файлы пустые, то ошибок не обнаружено.

Файлы **lib.exe** и **polib.exe** можно взять из бесплатного пакета **MASM32**.
Теперь, на основе этих библиотек напишем небольшую ассемблерную программу **mb.asm**.

```
; ========================================================================

.686p
.mmx
.model flat

option casemap:none

; extrn MessageBoxA : proc
; extrn MessageBoxA@16 : proc
; extrn _imp__MessageBoxA : dword
extrn _imp__MessageBoxA@16 : dword

includelib Lib\user32.lib

; extrn ExitProcess : proc
; extrn ExitProcess@4 : proc
; extrn _imp__ExitProcess : dword
extrn _imp__ExitProcess@4 : dword

includelib Lib\kernel32.lib

; ========================================================================

.data

Header      db "Question", 0
MsgText     db "Do you want quit?", 0

; ========================================================================

.code

_Start:
    push 4   ; MB_YESNO
    push offset Header
    push offset MsgText
    push 0

    ; call MessageBoxA
    ; call MessageBoxA@16
    ; call _imp__MessageBoxA
    call _imp__MessageBoxA@16

    cmp eax, 6  ; IDYES : 6 - Yes, 7 - No.

    je Quit
    jmp _Start

Quit:
    push 0

    ; call ExitProcess
    ; call ExitProcess@4
    ; call _imp__ExitProcess
    call _imp__ExitProcess@4

end _Start

; ========================================================================
```

В качестве **lib**-файлов, используем только что полученные библиотеки. На основе этого кода можно получить четыре варианта исполняемого файла (раскомментируя соответствующие строки). Обратим внимание, что первые два определения **extrn** содержат ключевое слово **proc**, а последние два – **dword**. Компилируем эту программу. Все четыре ее варианта дадут один и тот же результат (рис. 1).

Рис. 1. Результат выполнения программы **mb.asm**.

Нам интересно в ней то, что вызов функции мы можем осуществлять четырьмя различными способами, которые покрывают как стиль программирования **Iczelion**'а, так и стиль **Ильфака Гильфанова** (в его программе **IdaPro**). Только, в отличие от **Iczelion**'а, нам нет необходимости создавать или использовать чужие громоздкие **inc**-файлы.

Интересно, что пары файлов генерят практически идентичный бинарный код. Первая пара дает (в **OllyDbg v. 1.10**) код

```
00401000 >/$ 6A 04          /PUSH 4                                  ; /Style = MB_YESNO|MB_APPLMODAL
00401002  |. 68 00304000     |PUSH mb1.00403000                       ; |Title = "Question"
00401007  |. 68 09304000     |PUSH mb1.00403009                       ; |Text = "Do you want quit?"
0040100C  |. 6A 00           |PUSH 0                                  ; |hOwner = NULL
0040100E  |. E8 0F000000     |CALL <JMP.&user32.MessageBoxA>          ; \MessageBoxA
00401013  |. 83F8 06         |CMP EAX,6
00401016  |. 74 02           |JE SHORT mb1.0040101A
00401018  |.^EB E6           \JMP SHORT mb1.<ModuleEntryPoint>
0040101A  |> 6A 00           PUSH 0                                   ; /ExitCode = 0
0040101C  \. E8 07000000     CALL <JMP.&kernel32.ExitProcess>         ; \ExitProcess
00401021     CC              INT3
00401022   $-FF25 08204000   JMP DWORD PTR DS:[<&user32.MessageBoxA>]   ;  user32.MessageBoxA
00401028   .-FF25 00204000   JMP DWORD PTR DS:[<&kernel32.ExitProcess>] ;  kernel32.ExitProcess
```

а вторая, код

```
00401000 >/$ 6A 04          /PUSH 4                                  ; /Style = MB_YESNO|MB_APPLMODAL
00401002  |. 68 00304000     |PUSH mb4.00403000                       ; |Title = "Question"
00401007  |. 68 09304000     |PUSH mb4.00403009                       ; |Text = "Do you want quit?"
0040100C  |. 6A 00           |PUSH 0                                  ; |hOwner = NULL
0040100E  |. FF15 08204000   |CALL DWORD PTR DS:[<&user32.MessageBoxA>] ; \MessageBoxA
00401014  |. 83F8 06         |CMP EAX,6
00401017  |. 74 02           |JE SHORT mb4.0040101B
00401019  |.^EB E5           \JMP SHORT mb4.<ModuleEntryPoint>
0040101B  |> 6A 00           PUSH 0                                   ; /ExitCode = 0
0040101D  \. FF15 00204000   CALL DWORD PTR DS:[<&kernel32.ExitProcess>]; \ExitProcess
```

Мы видим, что **extrn . . . proc** порождает переходники для системных функций, а **extrn . . . dword** – нет.

Все это, конечно, замечательно, однако для больших проектов нам нужны будут средства автоматизации или хотя бы механизации построения полных **def**-файлов для системных **dll**-ек и не только системных.

Если бы дело касалось только родных экспортируемых имен некоторой **dll**-ки, то мы могли бы взять за основу текстовый файл, генерируемой утилитой **dumpbin** из **MS VS C++** или аналогичную программу. Однако нам нужны будут еще отладочные символы, а это значит, что придется работать с **pdb**-файлами от **Майкрософт** (для системных **dll**-ек). В идеале было бы хорошо написать свою утилиту, типа **pdb2def.exe**, преобразующей отладочные символы непосредственно в определения библиотечных файлов. Но для этого нужно знать формат **pdb**-файлов или **API** для работы с ними. Но до тех пор, пока это не будет сделано, можно воспользоваться более простым вариантом, хотя и немного громоздким. Хотя, как показывает опыт, без ручной корректировки созданных **def**-файлов не обойтись в любом случае. Не обязательно, делать все изменения сразу, да это практически и невозможно ибо **Майкрософт**

не обеспечивает полного соответствия отладочных символов реально используемым. Однако имея под рукой полный **def**-файл, для данного системного **dll**-файла, всегда можно внести необходимые корректировки по ходу дела, компилируя затем, на его основе, **lib**-файл.

Получение def-файлов из листинга IdaPro для системных dll

Работая с листингами «Иды», легко обнаружить, что при условии использования отладочных символов, в них есть фактически вся информация, необходимая нам для формирования собственных **def**-файлов. Только эта информация «размазана» по всему листингу «Иды», поэтому нам придется писать внешний (потому, что демо-версия «Иды» не сохраняет результаты работы, иначе, как через буфер обмена) скрипт, который будет делать за нас всю необходимую работу.

Мы будем использовать **IdaPro v. 5.7 demo**. Для других версий «Иды» строки сигнатур (**Line1** и **Line2**) могут отличаться (например, за счет использования пробелов вместо табуляторов или наоборот). Скрипт мы будем писать на **Visual FoxPro**. Вы можете использовать другой, более удобный вам язык, применяя похожий алгоритм.

Приведем сразу окончательный вариант нашего скрипта **lst2def.prg**, изучать который удобней всего в процессе работы. Если у вас будут какие-то замечания или предложения по его функционированию, то можете высказаться, используя контактную информацию. Конструктивная критика приветствуется  .

```
*****************************************************************************
* lst2def.prg
*****************************************************************************
CLEAR
SET TALK OFF
SET SAFETY OFF
SET TEXTMERGE ON

filename = "mfc42u"

infile = filename + ".lst"
outfile = filename + ".txt"
dbffile = "temp.dbf"
dbffile2 = filename + ".dbf"
deffile = filename + ".def"

Line1 = " ; Exported entry"
Len1 = LEN(Line1)
Line2 = "                  public"
Len2 = LEN(Line2)

MaxFuncNameLen = 125  && Выделенная длина для имени функции
OrdinalOffset = 75  && Отступ для записи ординала

ToExportOrdinals = .T.

crlf = CHR(13) + CHR(10)  && Символы завершения строки
tabulator = CHR(9)
* tabulator2 = tabulator + tabulator
tabulator2 = "  "

file1 = FOPEN(infile)

IF file1 < 0
  WAIT "Не могу открыть файл: " + infile
  QUIT
ENDIF

file2 = FCREATE(outfile)

IF file2 < 0
  WAIT "Не могу создать файл: " + outfile
  QUIT
ENDIF

End1 = FSEEK(file1, 0, 2)  && Определяем размер файла
Top1 = FSEEK(file1, 0)  && Идем в начало. Сама переменная не нужна
```

```
IF End1 <= 0  && Если файл пуст
  MESSAGEBOX("Пустой файл: " + infile)
  QUIT
ENDIF

pos = 0
IsLine1 = .F.

&& Максимально возможное количество строк для общих функций и до строки public (д.б. >= 6)
KolStr = 50

DIMENSION aSnum[KolStr]  && Строковые ординалы функций
DIMENSION aFunc[KolStr]  && Их базовые имена

&& Обнуляем массивы
FOR i = 1 TO KolStr
  aSnum[i] = ""
  aFunc[i] = ""
ENDFOR

DO WHILE NOT EOF(file1)  && На самом деле мы не можем доверять этому условию
  strline = FGETS(file1)

  Curpos1 = FSEEK(file1, 0, 1)

  IF Curpos1 >=  End1  && Достигнут конец файла
    EXIT
  ENDIF

  pos = AT(Line1, strline)  && Ищем первую строку

  IF pos > 0  && Нашли первую строку Line1
    IsLine1 = .T.
  ELSE
    LOOP
  ENDIF

  && Первая строка Line1 найдена! Однако этих строк может быть несколько.
  && Т.е. разные функции могут иметь общий код.

  strline = SUBSTR(strline, pos + Len1 + 1)  && Выделяем число (номер / ординал функции)

  pos = AT(".", strline)  && Ищем признак завершения числа

  IF pos > 0  && Нашли конец числа
    aSnum[1] = RIGHT("     " + LTRIM(LEFT(strline, pos - 1)), 5) + " "
    aFunc[1] = LEFT(SUBSTR(strline, pos + 2) + REPLICATE(" ", MaxFuncNameLen), MaxFuncNameLen)
  ELSE
    MESSAGEBOX("Нет признака для номера функции!")
    QUIT
  ENDIF

  KolLine1 = 1  && Количество найденных подряд строк Line1 по умолчанию

  && В течении следующих KolStr - 1 пытаемся искать другие строки Line1
  FOR i = 2 TO KolStr
    strline = FGETS(file1)

    pos = AT(Line1, strline)  && Снова ищем первую строку

    IF pos > 0  && Нашли новую копию первой строки Line1
      KolLine1 = i

      strline = SUBSTR(strline, pos + Len1 + 1)  && Выделяем число (номер / ординал функции)

      pos = AT(".", strline)  && Ищем признак завершения числа

      IF pos > 0  && Нашли конец числа
        aSnum[i] = RIGHT("     " + LTRIM(LEFT(strline, pos - 1)), 5) + " "
        aFunc[i] = LEFT(SUBSTR(strline, pos + 2) + REPLICATE(" ", MaxFuncNameLen), MaxFuncNameLen)
      ELSE
        MESSAGEBOX("Нет признака для номера функции!")
        QUIT
      ENDIF
    ELSE  && Нет других копий строк Line1
      EXIT
```

```
    ENDIF
  ENDFOR

  && Ищем вторую строку Line2, которой могут соответствовать несколько первых строк Line1

  IsLine2 = .F.

  && В течении следующих KolStr должна быть найдена вторая строка Line2
  FOR i = 1 TO KolStr
    pos = AT(Line2, strline)  && Ищем вторую строку Line2

    IF pos > 0  && Нашли вторую строку Line2
      IsLine2 = .T.
      EXIT
    ENDIF

    strline = FGETS(file1)
  ENDFOR

  IF NOT IsLine2
    MESSAGEBOX("Нет пары для первой строки!")
    QUIT
  ENDIF

  && Вторая строка Line2 найдена!

  FuncName = TRIM(SUBSTR(strline, pos + Len2 + 1))  && Выделяем полное имя экспортируемой функции
  FuncName = LEFT(FuncName + REPLICATE(" ", MaxFuncNameLen), MaxFuncNameLen)

  i1 = 0
  OrdList = ""

  && Следующим KolLine1 строкам Line1 соответствует одна строка Line2
  FOR i = 1 TO KolLine1
    IF NOT EMPTY(aFunc[i])
      FWRITE(file2, FuncName + aSnum[i] + aFunc[i] + crlf)
    ELSE
      i1 = IIF(EMPTY(i1), i, i1)
      OrdList = OrdList + ALLTRIM(aSnum[i]) + ", "
    ENDIF
  ENDFOR

  && Включаем список ординалов с одинаковым телом функции
  IF NOT EMPTY(OrdList)
    FWRITE(file2, FuncName + aSnum[i1] + aFunc[i1] + LEFT(OrdList, LEN(OrdList) - 2) + crlf)
  ENDIF
ENDDO

IF NOT IsLine1
  MESSAGEBOX("Искомая строка не найдена!")
  QUIT
ENDIF

CLOSE ALL

&& Dbf файл для экспортируемых функций
CREATE TABLE (dbffile) (FuncName c((MaxFuncNameLen)), Ord n(6,0), Comments c((MaxFuncNameLen)),
    OrdList c(254))
APPEND FROM (outfile) TYPE SDF

CLOSE ALL

&& Сортруем dbf файл по ординалу
SELECT * FROM (dbffile) ORDER BY Ord INTO TABLE (dbffile2)

SELECT (dbffile2)

SET TEXTMERGE TO (deffile)

\\LIBRARY "<<filename + '.dll'>>"
\EXPORTS

ExpLine = ""
ExpLine2 = ""
FuncLine = ""
```

```
&& Генерим def файл
FOR i = 1 TO RECCOUNT()
  GO i
  SCATTER MEMVAR

  IF NOT EMPTY(m.Comments) AND TRIM(m.FuncName) <> TRIM(m.Comments)
    \<<TRIM(m.Comments)>>
    FuncLine = TRIM(m.FuncName) + " = " + TRIM(m.Comments)
  ELSE
    IF ToExportOrdinals
      OrdList2 = ALLTRIM(OrdList)
      pos = AT(", ", OrdList2)  && Ищем первую запятую в списке ординалов

      IF pos > 0  && Нашли первую запятую в списке ординалов
        OrdList1 = "@" + LEFT(OrdList2, pos - 1)
        OrdList3 = SUBSTR(OrdList2, pos + 2)
        FuncLine = TRIM(m.FuncName)  + tabulator2 + OrdList1 + IIF(EMPTY(OrdList3), "", "  ; @") +
OrdList3
      ELSE
        FuncLine = TRIM(m.FuncName)  + tabulator2 + IIF(EMPTY(OrdList), "", "@") + OrdList2
      ENDIF
    ELSE  && NOT ToExportOrdinals
      FuncLine = TRIM(m.FuncName)  + tabulator2 + IIF(EMPTY(OrdList), "", "; @") + ALLTRIM(OrdList)
    ENDIF
  ENDIF

  FuncLineLen = LEN(TRIM(FuncLine))

  IF FuncLineLen < OrdinalOffset
    ExpLine = LEFT(TRIM(FuncLine) + REPLICATE(" ", OrdinalOffset), OrdinalOffset)
  ELSE
    ExpLine = LEFT(TRIM(FuncLine) + REPLICATE(" ", FuncLineLen + 1), FuncLineLen + 1)
  ENDIF

  \<<TRIM(FuncLine)>>
ENDFOR

CLOSE ALL
QUIT
*****************************************************************************
*****************************************************************************
```

Исходный листинг (**lst**-файл) для данного **dll**-файла с отладочными символами (**pdb**-файл) просто создаем, копируя в него содержимое буфера обмена полученного в **IdaPro v. 5.7 demo**, на который затем направляем наш скрипт. В итоге получим **def**-файл (другие временные файлы можно удалить) из которого уже можно делать соответствующий **lib**-файл. Если возникнут шероховатости, то можно подправить либо скрипт, либо сгенерированный **def**-файл.

Код скрипта **lst2def.prg** можно найти также в папке **Prg** прилагаемого файла с результатами исследований. Там же в качестве примера приложен листинг **version.lst** соответствующей **dll**-ки. Запуская на выполнение скрипт, получим в результате файл **version.def** (ненужные файлы удаляем). В папке **Def** находятся уже отредактированные вручную созданные, таким образом, файлы определений. Получая свои дефы, вы можете сравнить их с нашими.

Работа с программой Visual FoxPro

Наверное, пару слов надо сказать относительно **Visual FoxPro**. Кому-то нравится **Perl** для обработки текстовых файлов, а мне **VFP**. Для его работы (на уровне ядра) достаточно двух файлов, например, для **9**-й версии **VFP** (годится и меньшая версия) это будут **vfp9.exe** (**5.6** Мб) и **vfp9enu.dll** (**1.5** Мб), которые можно взять непосредственно из любой инсталляции **Visual FoxPro**. Для исполнения скрипта достаточно записать в командной строке

```
vfp9.exe lst2def.prg
```

в каталоге, где находится исходный файл листинга. Однако более удобно запускать программу **Visual FoxPro** в его собственной оболочке, назначив ассоциацию на файл ***.prg** (клавиша **F10** в **Total Commander**) вида

```
Microsoft Visual FoxPro Program ("C:\Program Files\Microsoft Visual FoxPro 9\vfp9.exe" -SHELLOPEN "%1")
```

Нажав **Enter** на файл **lst2def.prg**, мы загружаемся в **IDE VFP**, где мы можем отредактировать скрипт перед его исполнением (**Ctrl+E**). Есть еще варианты работы с **runtime** версиями, но, думаю, при желании, вы разберетесь с ними сами.

Ручное редактирование полученных def-файлов

Вот пример сгенерированного нашим скриптом файла определений **mfc42u.def** для **mfc42u.dll**, взятого из папки **C:\WINDOWS\system32** .

```
LIBRARY "mfc42u.dll"
EXPORTS
?classCCachedDataPathProperty@CCachedDataPathProperty@@2UCRuntimeClass@@B
?classCDataPathProperty@CDataPathProperty@@2UCRuntimeClass@@B
DllCanUnloadNow
_DllCanUnloadNow@0 = DllCanUnloadNow
DllGetClassObject
_DllGetClassObject@12 = DllGetClassObject
DllRegisterServer
_DllRegisterServer@0 = DllRegisterServer
DllUnregisterServer
_DllUnregisterServer@0 = DllUnregisterServer
??0_AFX_CHECKLIST_STATE@@QAE@XZ  @256
??0_AFX_COLOR_STATE@@QAE@XZ  @257
?_AfxBinaryCompatibleStubFunction@@YAXXZ  @258  ; @590, 598, 944, 946, 947, 948, 949, 951, 952, 953, 954,
1178, 1187, 1188, 2186, 2761
??0_AFX_DAO_STATE@@QAE@XZ  @259
??0_AFX_EDIT_STATE@@QAE@XZ  @260
. . .
?GetMenuStringW@CMenu@@QBEHIAAVCString@@I@Z  @6923
??0CInvalidArgException@@QAE@HI@Z  @6924
??1CInvalidArgException@@UAE@XZ  @6925
?GetRuntimeClass@CInvalidArgException@@UBEPAUCRuntimeClass@@XZ  @6926
?classCInvalidArgException@CInvalidArgException@@2UCRuntimeClass@@B  @6927
?AfxThrowInvalidArgException@@YGXXZ  @6928
?AtlA2WHelper@@YGPAGPAGPBDH@Z  @6929
?AtlW2AHelper@@YGPADPADPBGH@Z  @6930
?GetMonth@CTime@@QBEHXZ  @6931
?GetThreadValue@CThreadSlotData@@QAEPAXH@Z  @6932
?GetYear@CTime@@QBEHXZ  @6933
?HashKey@CMapStringToPtr@@QBEIPBG@Z  @6934  ; @6935, 6936
```

Скажем пару слов по структуре нашего **def**-файла. Как показывает запуск **dumpbin.bat** (**dumpbin.exe** взят из **MS Visual Studio C++**)

```
SET WINSYS32=C:\WINDOWS\SYSTEM32
SET FILENAME=mfc42u

:: Экспортируемые функции данной dll
Bin\dumpbin /EXPORTS %WINSYS32%\%FILENAME%.dll /OUT:%FILENAME%.txt > %FILENAME%.
```

в файле **mfc42u.dll** практически все функции определены по ординалам

```
  Ord Hint RVA      Name

    5    0 000117E0 ?classCCachedDataPathProperty@CCachedDataPathProperty@@2UCRuntimeClass@@B
    6    1 00011810 ?classCDataPathProperty@CDataPathProperty@@2UCRuntimeClass@@B
    7    2 000DE1D0 DllCanUnloadNow
    8    3 000DE190 DllGetClassObject
    9    4 000DE210 DllRegisterServer
   10    5 000DE260 DllUnregisterServer
  256      00042FD0 [NONAME]
  257      0004A4A0 [NONAME]
```

```
 258      000D3820 [NONAME]
 259      000718B0 [NONAME]
 260      0005A4E0 [NONAME]
. . .
6936      00024190 [NONAME]
6937      00080A30 [NONAME]
6938      00080A30 [NONAME]
6939      00080A30 [NONAME]
6940      00080A30 [NONAME]
```

Поскольку экспортируемых имен функций в **mfc42u.dll** практически нет (за исключением первых шести), то наши имена определенные в **mfc42u.def** по сути являются псевдоименами, для определенности связанные явно с экспортируемыми ординалами. Когда же в **dll**-ке экспортируются имена, то нам уже нет необходимости и смысла указывать ординалы (которые могут быть скорее подвержены изменениям в будущих версиях библиотек, чем сами имена). Поэтому первые шесть функций (с их алиасами) указаны без ординалов.

В данном случае, для экспорта по ординалам, мы объявляем переменную

```
ToExportOrdinals = .T.
```

в результате чего ординалы функций указаны явно. Когда же нет необходимости экспортировать ординалы, то в этом случай мы пишем

```
ToExportOrdinals = .F.
```

в скрипте **lst2def.prg**. В смешанном же случае ситуация будет сложнее. Пока мы можем меньшую часть отредактировать вручную (до сих пор это не превышало несколько случаев на один файл). В перспективе, можно отслеживать экспорт по ординалу в исследуемом **dll**-файле (как это делает **dumpbin.exe**) для более корректной генерации **def**-файла. Тем не менее, если имя функции имеет алиас в отладочных символах, то этот случай подразумевает экспорт имен и наша переменная **ToExportOrdinals** игнорируется. В любом случае, вы всегда может подрегулировать данный скрипт под свои нужды.

Тестирование полученных lib-файлов

В качестве теста для созданных **lib**-файлов используем имеющиеся уже у нас **asm**-файлы из первых двух статей. Они также продублированы в каталоге **Results**. Содержимое соответствующей подпапки копируем на два уровня выше (туда, куда скопировано содержимое каталога **write**). Все подкаталоги с ассемблерным кодом содержат следующие командные файлы:

def2lib.cmd (для генерации **11**-ти, используемых в нашем проекте системных **lib**-файлов)
def2lib.bat (для создания только одного, явно указанного, **lib**-файла)
asm.bat (один или несколько, для генерации соответствующего **exe**-шника, на базе созданных **lib**-файлов).

Пакетный файл **def2lib.cmd** достаточно запустить один раз, который из **11**-ти **def**-файлов из папки **Def** сгенерирует такое же количество **lib**-файлов в папку **Lib** (изначально пустой). Если надо пересоздать какой-нибудь один **lib**-файл, то запускаем **def2lib.bat** (указав в нем нужное имя). **asm.bat** генерит непосредственно приложение, а если их несколько, то различные варианты этого приложения.

В результате работы пакетных файлов, создаются разнообразные файлы сообщений. Пустые файлы означают нормальную работу соответствующей команды. Все их можно спокойно удалять.

Для тестирования нам понадобятся наши собственные файлы:

advapi32.lib
imm32.lib

kernel32.lib
mfc42u.lib
msvcrt.lib
ntdll.lib
shell32.lib
shlwapi.lib
user32.lib
version.lib

В качестве исходных **dll**-файлов можно взять, в принципе, любые версии **Windows NT** помня только о различиях в составе функций разных версий **NT** и их сервис паков. В нашем случае, дефы, построенные по длл-кам **Windows 2003 Server, sp. 2**, работали под **Windows XP, sp. 3** и наоборот.

Проблемы, возникающие при использование дефов / либов, полученных данным способом, возникают из-за неполноты информации в отладочных символах. Например, в листинге **mfc42u.lst** для **mfc42u.dll** можно встретить такой код

```
.text:7F05FE80 ; Exported entry 1016.
.text:7F05FE80 ; Exported entry 1055.
.text:7F05FE80 ; Exported entry 1747.
.text:7F05FE80 ; Exported entry 2015.
.text:7F05FE80 ; Exported entry 2016.
.text:7F05FE80 ; Exported entry 2117.
.text:7F05FE80 ; Exported entry 2474.
.text:7F05FE80 ; Exported entry 2977.
.text:7F05FE80 ; Exported entry 3074.
.text:7F05FE80 ; Exported entry 3101.
.text:7F05FE80 ; Exported entry 3142.
.text:7F05FE80 ; Exported entry 3254.
.text:7F05FE80 ; Exported entry 3744.
.text:7F05FE80 ; Exported entry 4043.
.text:7F05FE80 ; Exported entry 4103.
.text:7F05FE80 ; Exported entry 4112.
.text:7F05FE80 ; Exported entry 4383.
.text:7F05FE80 ; Exported entry 4564.
.text:7F05FE80 ; Exported entry 4650.
.text:7F05FE80 ; Exported entry 4666.
.text:7F05FE80 ; Exported entry 4721.
.text:7F05FE80 ; Exported entry 4876.
.text:7F05FE80 ; Exported entry 4974.
.text:7F05FE80 ; Exported entry 5010.
.text:7F05FE80 ; Exported entry 5501.
.text:7F05FE80 ; Exported entry 5508.
.text:7F05FE80 ; Exported entry 5523.
.text:7F05FE80 ; Exported entry 5524.
.text:7F05FE80 ; Exported entry 5526.
.text:7F05FE80 ; Exported entry 5549.
.text:7F05FE80 ; Exported entry 5559.
.text:7F05FE80 ; Exported entry 5561.
.text:7F05FE80 ; Exported entry 6051.
.text:7F05FE80 ; Exported entry 6320.
.text:7F05FE80
.text:7F05FE80 ; =============== S U B R O U T I N E =======================================
.text:7F05FE80
.text:7F05FE80 ; public: virtual unsigned long __stdcall COlePropertiesDialog::XOleUIObjInfo::Release(void)
.text:7F05FE80                 public ?Release@XOleUIObjInfo@COlePropertiesDialog@@UAGKXZ
.text:7F05FE80 ?Release@XOleUIObjInfo@COlePropertiesDialog@@UAGKXZ proc near
.text:7F05FE80                                         ; CODE XREF: COleServerDoc::GetInterfaceHook(void const *)+55□p
.text:7F05FE80                                         ; CMonikerFile::Open(ushort const *,CFileException *)+40□p ...
.text:7F05FE80                 xor     eax, eax        ; MFC42u_1016 and others
.text:7F05FE82                 retn    4
.text:7F05FE82 ?Release@XOleUIObjInfo@COlePropertiesDialog@@UAGKXZ endp
.text:7F05FE82
.text:7F05FE82 ; ---------------------------------------------------------------------------
```

Это означает, что определена только одна функция

virtual unsigned long __stdcall **COlePropertiesDialog::XOleUIObjInfo::Release**(void)
или, в записи понятной компилятору,

public **?Release@XOleUIObjInfo@COlePropertiesDialog@@UAGKXZ** proc near

которая является телом следующих функций, определенных только по ординалам:

1016, 1055, 1747, 2015, 2016, 2017, 2474, **2977**, **3074**, 3101, **3142**, **3254**, 3744, 4043, 4103, 4112, 4383, 4564, 4650, 4666, 4721, 4876, 4974, 5010, 5501, 5508, 5523, 5524, 5526, 5549, 5559, 5561, 6051, 6320.

Всего **34** функции. Однако для этих ординалов, в **exe**-шниках (от **Майкрософт**), где они используются, могут быть определены их реальные имена. Скажем, в **winmsd.exe**, рассмотренном в прошлой статье, встречается импорт из **mfc42u.dll** этих функций не по ординалам, а по их реальным именам (известных только **Майкрософту**)

```
Address  Ord  FuncName

01001054 2977 ?GetConnectionHook@CCmdTarget@@MAEPAUIConnectionPoint@@ABU_GUID@@@Z
01001058 3142 ?GetExtraConnectionPoints@CCmdTarget@@MAEHPAVCPtrArray@@@Z
0100105C 3254 ?GetInterfaceHook@CCmdTarget@@UAEPAUIUnknown@@PBX@Z
01001088 3074 ?GetDispatchIID@CCmdTarget@@UAEHPAU_GUID@@@Z
```

(другие функции просто не применяются в **winmsd.exe**). Получается, что в одних источниках **Майкрософт** секретит информацию, а в других рассекречивает («правая рука не знает, что делает левая» 🙂). Подобные случаи трудно обработать автоматически, поэтому, чтобы использовать данный файл **mfc42u.def** для **winmsd.asm** его нужно немного подправить напильником 🙂. В этом и состоит смысл построения **def**-файлов, чтобы всегда иметь возможность внести в них нужные исправления.

В нашем случае компилятор ругается на **12** функций, используемых в **winmsd.asm**, но определения которых отсутствуют в нашем **mfc42u.def**. Модифицируем этот **def**-файл. Вместо строки

?Release@XOleUIObjInfo@COlePropertiesDialog@@UAGKXZ @1016 ; @1055, 1747, 2015, 2016, 2117, 2474, **...**

созданной нашим скриптом, напишем

```
?Release@XOleUIObjInfo@COlePropertiesDialog@@UAGKXZ @1016 ; @1055, 1747, 2015, 2016, 2117, 2474, ...
?GetConnectionHook@CCmdTarget@@MAEPAUIConnectionPoint@@ABU_GUID@@@Z @2977
?GetDispatchIID@CCmdTarget@@UAEHPAU_GUID@@@Z @3074
?GetExtraConnectionPoints@CCmdTarget@@MAEHPAVCPtrArray@@@Z @3142
?GetInterfaceHook@CCmdTarget@@UAEPAUIUnknown@@PBX@Z @3254
```

По-хорошему, надо бы описать и оставшиеся **30** функций. Но в данном тесте в **winmsd.asm** их нет, а мы народ ленивый 🙂, привыкли «решать проблемы по мере их поступления» 🙂. Короче, перекомпилируем модифицированный **mfc42u.def** в **mfc42u.lib** и затем снова компилируем **winmsd.asm**. Как и ожидалось, число ошибок уменьшилось на четыре. Остальные ошибки устраняются аналогично. Для этого берем имя (возможно неполное) из файла сообщения линковщика (у нас это файл **c.a**). На вкладке импорта **IdaPro** (или в другом походящем месте «Иды») для загруженного файла **winmsd.exe** смотрим какой ординал соответствует этому имени. В нашем, уже созданном, **def**-файле ищем этот ординал и рядышком (для порядка 🙂) делаем соответствующую запись вида

ImportFuncName @**Ord**

как в примере выше.

Проделав эту работу для остальных неопознанных линковщиком функций, мы получаем конечный, для данного теста, вариант файла **mfc42u.def**, который уже можно преобразовать в **lib**-файл.

Для других библиотек ошибки линковщика устраняются аналогично. Отметим только некоторые дополнительные нюансы.

В файле **kernel32.def** имя функции

```
HeapSize
```

заменили, с учетом информации «Иды», на

```
NTDLL.RtlSizeHeap
_HeapSize@12 = NTDLL.RtlSizeHeap
```

Просто оставить **HeapSize** не удается, так как функция **«forwarded to NTDLL.RtlSizeHeap»**.

Некоторые функции имеют несколько алиасов. Так, например, в **user32.def** пришлось к записям

```
DestroyWindow
_NtUserDestroyWindow@4 = DestroyWindow
```

добавить еще одну

```
_DestroyWindow@4 = DestroyWindow
```

В **shlwapi.def** функцию **_IsOS@4** пришлось явно определить по ординалу

```
_IsOS@4   @437
```

(Смотрите по этому поводу предыдущую статью.)

Скомпилированный для обновленных **lib**-файлов **winmsd.exe** выполнился успешно. Все используемые **def**-файлы (кроме **mfc42u.def**) создавались с отключенным экспортом по ординалам (**ToExportOrdinals = .F.**). Исправлений в **def**-файлах было относительно немного для компилируемых нами **asm**-файлов, Все **exe**-шники работали без явных проблем. Впрочем, и внешних функций они использовали не очень много.

Просмотр структуры lib-файлов

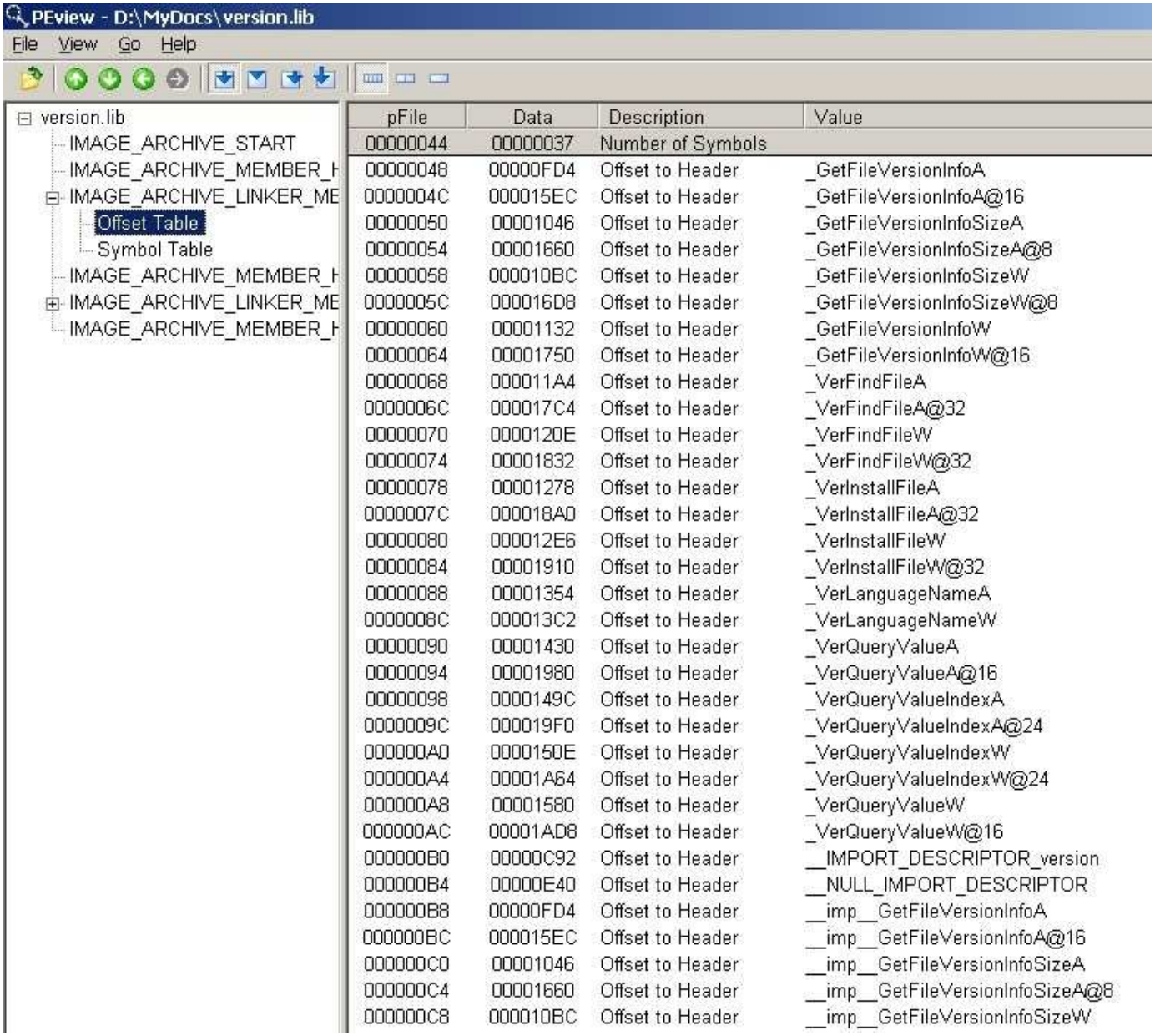
Есть такая утилита **PEview** (http://www.magma.ca/~wjr/PEview.zip). Очень хороша для просмотра структуры **lib**-файлов (рис.2). При желании, вы можете пользоваться ею также.

Выводы

Понятно, что представленное решение не является идеальным, И для новых проектов может потребоваться дополнительная правка **def**-файлов. Однако думается, что это будет вполне обозримым по трудоемкости. В любом случае, встречные предложения по совершенствованию техники создания собственных **def** и **lib**-файлов для различных **dll** только приветствуются.

Примечание

К данному тексту приложен файл **CreatingLibFiles.003** (http://erfaren.narod.ru/Asm/CreatingLibFiles.003 - измените расширение в **zip**), с результатами исследования и сгенерированными и отредактированными **def**-файлами. Можно также посмотреть **html** версию этой статьи (http://erfaren.narod.ru/Asm/Erfaren003.htm) либо ее **pdf**-файл (http://erfaren.narod.ru/Asm/Erfaren-003-Creating-lib-files.pdf).



pFile	Data	Description	Value
00000044	00000037	Number of Symbols	
00000048	00000FD4	Offset to Header	_GetFileVersionInfoA
0000004C	000015EC	Offset to Header	_GetFileVersionInfoA@16
00000050	00001046	Offset to Header	_GetFileVersionInfoSizeA
00000054	00001660	Offset to Header	_GetFileVersionInfoSizeA@8
00000058	000010BC	Offset to Header	_GetFileVersionInfoSizeW
0000005C	000016D8	Offset to Header	_GetFileVersionInfoSizeW@8
00000060	00001132	Offset to Header	_GetFileVersionInfoW
00000064	00001750	Offset to Header	_GetFileVersionInfoW@16
00000068	000011A4	Offset to Header	_VerFindFileA
0000006C	000017C4	Offset to Header	_VerFindFileA@32
00000070	0000120E	Offset to Header	_VerFindFileW
00000074	00001832	Offset to Header	_VerFindFileW@32
00000078	00001278	Offset to Header	_VerInstallFileA
0000007C	000018A0	Offset to Header	_VerInstallFileA@32
00000080	000012E6	Offset to Header	_VerInstallFileW
00000084	00001910	Offset to Header	_VerInstallFileW@32
00000088	00001354	Offset to Header	_VerLanguageNameA
0000008C	000013C2	Offset to Header	_VerLanguageNameW
00000090	00001430	Offset to Header	_VerQueryValueA
00000094	00001980	Offset to Header	_VerQueryValueA@16
00000098	0000149C	Offset to Header	_VerQueryValueIndexA
0000009C	000019F0	Offset to Header	_VerQueryValueIndexA@24
000000A0	0000150E	Offset to Header	_VerQueryValueIndexW
000000A4	00001A64	Offset to Header	_VerQueryValueIndexW@24
000000A8	00001580	Offset to Header	_VerQueryValueW
000000AC	00001AD8	Offset to Header	_VerQueryValueW@16
000000B0	00000C92	Offset to Header	__IMPORT_DESCRIPTOR_version
000000B4	00000E40	Offset to Header	__NULL_IMPORT_DESCRIPTOR
000000B8	00000FD4	Offset to Header	__imp__GetFileVersionInfoA
000000BC	000015EC	Offset to Header	__imp__GetFileVersionInfoA@16
000000C0	00001046	Offset to Header	__imp__GetFileVersionInfoSizeA
000000C4	00001660	Offset to Header	__imp__GetFileVersionInfoSizeA@8
000000C8	000010BC	Offset to Header	__imp__GetFileVersionInfoSizeW

Рис. 2. Просмотр **lib**-файла с помощью утилиты **PEview.exe**.