OllyDbg Plugin API v1.10



License Agreement (very official) General principles - read it first! Compilation - read it second! Alphabetical list of all Plugin API elements **Information functions Data formatting functions** Data input functions **Data conversion functions** Sorted data functions Name functions Search functions **Disassembly functions** Assembly functions **Procedure functions** Watch and expression functions **Breakpoint functions Execution and stepping functions**

Trace and profiling functions

<u>CPU-specific functions</u>

Source code support functions

Window functions

Thread functions

Memory functions

Module functions

Plugin functions

Plugin callback functions

Structures

Function prototypes

Custom messages

Sample program

OllyDbg © 2000-2004 Oleh Yuschuk, All Rights Reserved.

OllyDbg Plugin API © 2001-2004 Oleh Yuschuk, All Rights Reserved. Feel free to quote any parts of this document.

All brand names and product names used in OllyDbg, accompanying files or in this help are trademarks, registered trademarks, or trade names of their respective holders.

Registration

OllyDbg 1.10 is Copyright (C) 2000-2004 Oleh Yuschuk. To use this program on a permanent basis or for commercial purposes, you should register it. The registration is **free of charge** and assumes **no financial or other obligations** from your side - just be fair and let me know that you like this software. Any personal data in the registration form is optional (use your nickname or pseudonym if you want).

If you use OllyDbg together with **Randall Hyde's HLA (High Level Assembly)**, you don't need (but still allowed) to register.

When registering, you can subscribe for information (email) on the new release versions of this program. In this case you agree not to treat this information as a spam as long as number of letters does not exceed 4 each calendar year and they contain no advertisements from the third parties. If you no longer want to receive this information - well, just let me know, and I will immediately delete your address from my database.

If you are already a registered OllyDbg user, you don't need to re-register this version. If you are new, please read <u>license argeement</u>, fill the registartion form (*register.txt*) or copy and fill the following section from the help and email it to Ollydbg@t-online.de. I will keep your information confidential and will not give it to third persons, unless forced by a law.

Registration form for OllyDbg v1.10

To use OllyDbg, you must agree with all of the terms and

conditions of the accompanying License Agreement. All other

answers are optional.

Name _			
Title			

Company _____

City,state				
Country				
Where did you find OllyDbg				
Are you going to write your own plugins				
() Yes () No () Don't know				
I agree with all the terms and condition of the accompanying				
License Agreement (Very important! Please mark!)				
() Yes () No				
Date of registration				
If you want to receive notifications when OllyDbg 2.00 and				
subsequent versions will be ready, please enter your email				
address here:				
Thank you. If you have ideas how to improve OllyDbg and make				
it easier in use, or want to have some new features, please				
let me know. Your opinion helps me a lot!				
Your first idea:				

Your second idea: _____

Your third idea:

License Agreement

Trademark information

All brand names and product names used in OllyDbg, accompanying files or in this help are trademarks, registered trademarks, or trade names of their respective holders. They are used for identification purposes only.

License Agreement

This License Agreement ("Agreement") accompanies the OllyDbg version 1.10, OllyDbg Plugin Development Kit version 1.10 and related files ("Software"). By using the Software, you agree to be bound by all of the terms and conditions of the Agreement.

The Software is distributed "as is", without warranty of any kind, expressed or implied, including, but not limited to warranty of fitness for any particular purpose. In no event will the Author be liable to you for any special, incidental, indirect, consequential or any other damages caused by the use, misuse, or the inability to use of the Software, including any lost profits or lost savings, even if Author has been advised of the possibility of such damages.

The Software is owned by Oleh Yuschuk ("Author") and is Copyright (c) 2000-2004 Oleh Yuschuk. To use this Software on a permanent basis or for commercial purposes, you must register it by filling the supplied <u>registration</u> form and sending it to the Author. You don't need to register Software if you use it exclusively with Randall Hyde's High Level Assembly. If you are already a registered OllyDbg user, you don't need to re-register the Software again. If the Software is registered to a company or organization, any person within the company or organization has the right to use it at work. You may install the registered Software on any number of storage devices, like hard disks, floppy disks etc. and are allowed to make any number of backup copies of this Software.

You are not allowed to modify, decompile, disassemble or reverse engineer the Software except and only to the extent that such activity is expressly permitted by applicable law. You are not allowed to distribute or use any parts of the Software separately. You may make and distribute copies of this Software provided that a) the copy contains all files from the original distribution and these files remain unchanged; b) if you distribute any other files (for example, plugins) together with the Software, they must be clearly marked as such and the conditions of their use cannot be more restrictive than conditions of this Agreement; and c) you collect no fee (except for transport media, like CD or diskette), even if your distribution contains additional files.

You are allowed to develop and distribute your own plugins -- Dynamic Link Libraries that connect to the Software and make use of the functions implemented in the Software -- free of charge provided that a) your plugins contain no features that persuade or force user to register them, or limit functionality of unregistered plugins; b) you allow free distribution of your plugins on the conditions similar to that of the Software; and c) you collect no fee (except for transport media, like CD or diskette). If you want to develop commercial plugin, please contact Author for a special Agreement.

The distribution includes files PSAPI.DLL and DBGHELP.DLL that are the Microsoft(R) Redistributable files. These files should be installed only in the directory where the Software resides. You should use supplied PSAPI.DLL only on Windows NT(R) 4.0. You are not allowed to distribute PSAPI.DLL and/or DBGHELP.DLL separately from the Software.

This Agreement covers only the actual version 1.10 of the OllyDbg and version 1.10 of the OllyDbg Plugin Development Kit. All other versions are covered by separate License Agreements.

Fair use

Many software manufacturers explicitly disallow you any attempts of disassembling, decompilation, reverse engineering or modification of their programs. This restriction also covers all third-party dynamic-link libraries your application may use, including system libraries. If you have any doubts, contact the owner of copyright. The so called "fair use" clause can be misleading. You may want to discuss whether it applies in your case with competent lawyer. Please don't use OllyDbg for illegal purposes!

General principles

Welcome. OllyDbg v1.10 is the final version. I decided to stop its development. This does not mean that OllyDbg is dead - currently I'm preparing v2.0 - but new version will be incompatible with v1.xx, at least what concerns plugins. Sorry, but this is the only possible solution.

This documents describes OllyDbg Plugin API v1.10. There are no significant changes in interfaces or in structures, so plugins compiled for OllyDbg 1.06 or 1.08 will usually work with OllyDbg 1.10. The only changes that may be not 100% backward-compatible are limited to:

-Structures t_reg and t_bpoint are extended;

-New option "<u>Always on top</u>" requires special support from plugin windows;

-Function **Browsefilename** supports Save File dialog;

Plugin is a DLL that resides in OllyDbg directory and adds functionality to OllyDbg. You are free to write and distribute your own plugins, provided that they are free, too. (See License Agreement for details). On your request, I am ready to place such plugins for download on my home page. Commercial plugins are also allowed, but in this case you need special license.

To co-operate, different plugins require unique names, .udd tags, nametypes and so on. If you need some of these resources, please contact me. This service is absolutely free for you!

During startup, OllyDbg loads all available DLLs one by one and looks for entry points named <u>ODBG_Plugindata</u> and <u>ODBG_Plugininit</u>. If these entries are present and plugin reports compatible interface version, OllyDbg registers plugin and adds entry or submenu to Plugins popup in the main OllyDbg menu.

Plugins can add menu items to Disassembler, Dump, Stack, Registers, Memory, Modules, Threads, Breakpoints, Watches, References, Windows and Run trace windows. They can intercept both global shortcuts and shortcuts from one of the listed windows. They also can create own MDI windows. Plugins can write plugin-specific data to .udd files with module-dependent information and ollydbg.ini and access different data structures that describe debugged application. There are several (in general, optional) callback functions that allow easy but close interaction with OllyDbg. Additionally, plugins may call more than 170 plugin API functions.

Plugin interface is not object-oriented. Perharps this will come as surprise to you, but all my experience tells me that OOP is not as good as main software vendors try to sell. It is really good if you write small application performing standard functions. For a big weird project (and OllyDbg *is* a big weird project) OOP gives no real improvements in development time, errors in components are very hard to locate and even harder to correct. And - contrary to what vendors tell us - OO programs are usually *slow*. Stop crying, this is only my opinion, albeit proved by all my experience in the last 15 years or so. Anyway, try to swallow that you will get no ready-to-use objects here and are doomed to free memory by yourself when plugin terminates.

Plugin API is not re-entrant and does not implement critical sections. If your plugin creates new thread, don't call API functions from this thread, otherwise you risk to corrupt internal data structures and crash both program and OllyDbg!

Some exported API functions are not described here. Their direct use may bring OllyDbg in unstable state. I have added them for better compatibility with future versions of plugin interface.

See also: Compilation

Always on top

OllyDbg now supports "always on top" option for its MDI winsows (called from the Appearance menu). This option means that selected MDI window remains visible on the top of other windows.

Adding this useful option to a plugin is a matter of minutes. Plugins create MDI windows by calling <u>Newtablewindow</u> or <u>Quicktablewindow</u>. In the structure t_table, passed as a first parameter, you must specify flag TABLE_ONTOP, as in the <u>sample program</u>. To support this option, plugin must pass message WM_WINDOWPOSCHANGED to default plugin function (see <u>here</u>).

That's all! Easy, isn't it?

Compilation

Compilation

To compile your own plugin, you need some C or C++ compiler (together with linker and run-time libraries). Plugin interface (file *plugin.h*) is compatible at least with following compilers:

•**Borland's C++ 5.5** - command line compiler, available for free from www.borland.com (requires registration);

•Borland's C++ Builder 5 - based on the same C++ 5.5;

•Microsoft's Visual C++ 5.0 - rather old but solid and stable.

I haven't tried any other compilers. Please let me know if you find any incompatibilities and, if possible, send me corrected version of file *plugin.h.*

Plugin Development Kit includes source code for two fully functional sample plugins: **bookmark**, that allows to set up to 10 bookmarks in debugged application, and **command line**, that implements command line interface. Plugins are well documented. You can use them as a template for your own plugins. They are freeware, i.e. your rights to modify and re-use their source code are not limited in any way.

Following compiler settings are required for correct communication between plugin and OllyDbg. For compilers listed above, *plugin.h* forces or checks some of these rules:

•Export all callback functions by name, NOT by ordinal;

•If you use C++ compiler, disable name mangling on all callback functions (declare them as extern "C");

•Force standard C-style passing of parameters to all API and callback functions (declare them as cdecl);

•Force BYTE alignment of all structures declared in *plugin.h*;

•Set default character type to UNSIGNED.

Keep in mind that all pointers you get from OllyDbg may be NULL. This is a very common error to assume opposite.

Use static run-time libraries linked directly into your plugin, otherwise differences between versions of run-time DLLs will make OllyDbg unstable. Do not split your plugin unnecessarily into several DLLs. If you need data files that are not modifiable by user, try to place this data directly into your plugin as a resource.

To link your plugin to OllyDbg, you also need import library *ollydbg.lib*. Some compilers (Borland) include utility called *implib* that scans executable file (in our case, *ollydbg.exe*) and produces a special kind of library with a list of all exported functions. Some other products, like MSVC, can generate import library from the definition file (*ollydbg.def*). Similar products from other vendors are also available. For details, please consult documentation.

And, last but not least, don't waste resources! Don't export unused callback functions and make your program fast! OllyDbg in current version supports up to 32 plugins. If each of them will take only 50 ms to reject a global shortcut, then 50 ms for window-specific shortcut... you DO understand what I mean, don't you?

Contents of *plug110.zip*

Plugin kit archive contains following files:

Root directory:

bookmark.c - source of bookmark plugin

cmdexec.c - source of command line plugin

command.c - source of command line plugin

cmdline.rtf - RTF source of help (.hlp) file for command line plugin

ollydbg.def - OllyDbg definition file, some compilers need it to produce import library *ollydbg.lib*

plugin.h - header with definitions of plugin interface

plugins.hlp - this help file

Directory BC55:

sample.bpr - project file for BCB 5, produces sample.dll (same as bookmark.dll)

sample.cpp - main file for sample.bpr

bookmark.mak - make file for BC 5.5, produces bookmark.dll

cmdline.bpr - project file for BCB 5, produces cmdline.dll

cmdline.cpp - main file for cmdline.bpr

cmdline.mak - make file for BC 5.5, produces cmdline.dll

ollydbg.lib - OllyDbg import library in OMF format

Directory VC50:

bookmark.dsp - project file for Visual Studio 97, produces bookmark.dll
bookmark.dsw - project file for Visual Studio 97, produces bookmark.dll
bookmark.mak - make file for VC 5.0, produces bookmark.dll
cmdline.dsp - project file for Visual Studio 97, produces cmdline.dll
cmdline.dsw - project file for Visual Studio 97, produces cmdline.dll
cmdline.mak - make file for VC 5.0, produces cmdline.dll
cmdline.mak - make file for VC 5.0, produces cmdline.dll
cmdline.mak - make file for VC 5.0, produces cmdline.dll
ollydbg.lib - OllyDbg import library in COFF format

Making sample plugins with BC5.5

To build sample DLLs with BC 5.5, please do the following:

1. Copy files **bookmark.c**, **cmdexec.c**, **command.c**, **plugin.h**, **bc55\bookmark.mak**, **bc55\cmdline.mak**, **bc55\ollydbg.lib** to same directory;

2. Assuming that your BC 5.5 compiler is installed to c:\bc55, issue following commands:

c:\bc55\bin\make -f bookmark.mak

c:\bc55\bin\make -f cmdline.mak

3. Suppose that you write your own plugin, myplug, consisting of source files **a.c**, **b.c** and resource **c.rc**. All you need is to rename **bookmark.mak** to **myplug.mak** and modify three lines near the top of the file in a following way:

```
PROJECT = myplug.dll
```

OBJFILES = a.obj b.obj

RESFILES = c.rc

and then command

c:\bc55\bin\make -f myplug.mak

Making sample plugins with BCB 5

BCB projects must contain main C++ program with the same name as project and extention .cpp. For this reason, bookmark plugin created with Builder is called *sample.dll*. Of course, this has no influence on its functionality.

To build **sample.dll**, please do the following:

1. Copy files **bookmark.c**, **plugin.h**, **bc55\sample.bpr**, **bc55\sample.cpp** and **bc55\ollydbg.lib** to the same directory;

2. Open **sample.bpr** in Builder and make project.

To build **cmdline.dll**, please do the following:

1. Copy files **cmdexec.c**, **command.c**, **plugin.h**, **bc55\cmdline.bpr**, **bc55\cmdline.cpp** and **bc55\ollydbg.lib** to the same directory;

2. Open **cmdline.bpr** in Builder and make project.

Making sample plugins with VC 5.0 from the command line

To build sample DLLs with VC 5.0, please do the following:

1. Copy files **bookmark.c**, **cmdexec.c**, **command.c**, **plugin.h**, **vc50\bookmark.mak**, **vc50\cmdline.mak** and **vc50\ollydbg.lib** to the same directory;

2. In .mak files, edit lines

INCLUDE=c:\vc\include

LIBPATH=c:\vc\lib

so that they point to your include and library directories;

3. Assuming that your VC compiler, *cl.exe*, and make utility, *nmake.exe*, reside in c:\vc\bin, execute following commands:

c:\vc\bin\nmake -f bookmark.mak

c:\vc\bin\nmake -f cmdline.mak

Making sample plugins from the Visual Studio

To build **bookmark.dll**:

1. Copy files **bookmark.c**, **plugin.h**, **vc50\bookmark.dsp**, **vc50\bookmark.dsw** and **vc50\ollydbg.lib** to the same directory;

2. Open project **bookmark** in Visual Studio and make it.

To build **cmdline.dll**:

1. Copy files **cmdexec.c**, **command.c**, **plugin.h**, **vc50\cmdline.dsp**, **vc50\cmdline.dsw** and **vc50\ollydbg.lib** to the same directory;

2. Open project **cmdline** in Visual Studio and make it.

Plugin API - alphabetical list

API functions

This list contains all functions exported by OllyDbg. Some of them are reserved for the future use and are not described here. Direct calls to some undescribed functions may impair OllyDbg's stability. If you need some undescribed function, please contact Oleh Yuschuk. Functions that were added or changed since version 1.08 are marked with an asterisk (*).

Addsorteddata Addtolist **Analysecode** Animate Assemble Attachtoactiveprocess * **Broadcast** Browsefilename * Checkcondition **Compress Createdumpwindow Createlistwindow** <u>Createpatchwindow</u> * **Createprofilewindow** Creatertracewindow

Createsorteddata

Createthreadwindow

Createwatchwindow

Createwinwindow

Decodeaddress

Decodeascii

Decodecharacter

Decodefullvarname

Decodeknownargument

Decodename

Decoderange

Decoderelativeoffset

<u>Decodethreadname</u>

Decodeunicode

Decompress

Defaultbar

Deletebreakpoints

Deletehardwarebreakbyaddr

Deletehardwarebreakpoint

Deletenamerange

Deletenonconfirmedsorteddata

Deleteruntrace

Deletesorteddata

Deletesorteddatarange

Deletewatch

Demanglename

Destroysorteddata

<u>Disasm</u>

Disassembleback

Disassembleforward

Discardquicknames

<u>Dumpbackup</u>

<u>Error</u>

Expression

Findallcommands

Findalldllcalls

Findallsequences

Finddecode

Findfileoffset

Findfixup

Findhittrace

Findimportbyname

Findknownfunction

<u>Findlabel</u>

Findlabelbyname

Findmemory

Findmodule

Findname

Findnextname

Findnextproc

Findnextruntraceip

Findprevproc

Findprevruntraceip

Findprocbegin

<u>Findprocend</u>

Findreferences

Findsorteddata

Findsorteddataindex

Findsorteddatarange

Findstrings

Findsymbolicname

Findthread

Findunknownfunction

<u>Flash</u>

Followcall

Get3dnow

Get3dnowxy

Getaddressfromline

Getasmfindmodel

<u>Getasmfindmodelxy</u>

Getbprelname

Getbreakpointtype

Getbreakpointtypecount *

<u>Getcputhreadid</u>

Getdisassemblerrange

<u>Getfloat</u>

<u>Getfloatxy</u>

<u>Getfloat10</u>

Getfloat10xy

Gethexstring

<u>Gethexstringxy</u>

<u>Getline</u>

<u>Getlinexy</u>

Getlinefromaddress

Getlong

<u>Getlongxy</u>

<u>Getmmx</u>

<u>Getmmxxy</u>

Getnextbreakpoint

<u>Getoriginaldatasize</u>

Getproclimits

<u>Getregxy</u>

Getresourcestring

Getruntraceregisters

Getruntraceprofile

Getsortedbyselection

Getsourcefilelimits

<u>Getstatus</u>

<u>Gettableselectionxy</u>

<u>Gettext</u>

<u>Gettextxy</u>

<u>Getwatch</u>

<u>Go</u>

Guardmemory

<u>Hardbreakpoints</u>

<u>Havecopyofmemory</u>

<u>Infoline</u>

Injectcode

<u>Insertname</u>

Insertwatch

Isfilling

<u>Isprefix</u>

<u>Isretaddr</u>

Issuspicious

<u>IstextA</u>

<u>IstextW</u>

Listmemory *

<u>Manualbreakpoint</u>

Mergequicknames

<u>Message</u>

Modifyhittrace

Newtablewindow

<u>OpenEXEfile</u>

Painttable

Plugingetvalue

Pluginreadintfromini

Pluginreadstringfromini

Pluginsaverecord

Pluginwriteinttoini

<u>Pluginwritestringtoini</u>

Print3dnow

Printfloat10

Printfloat4

Printfloat8

<u>Printsse</u>

Progress

Quickinsertname

Quicktablewindow

Readcommand

Readmemory

<u>Redrawdisassembler</u>

Registerpluginclass

Restoreallthreads

Runsinglethread

Runtracesize

Scrollruntracewindow

Selectandscroll

Sendshortcut

Setbreakpoint *

Setbreakpointext *

<u>Setcpu</u>

<u>Setdisasm</u>

<u>Setdumptype</u>

Sethardwarebreakpoint

<u>Setmembreakpoint</u>

Settracecondition

Settracecount *

Showsourcefromaddress

Sortsorteddata

<u>Startruntrace</u>

<u>Stringtotext</u>

Suspendprocess

Tablefunction

Tempbreakpoint

Unregisterpluginclass

<u>Updatelist</u>

Walkreference

Walkreferenceex

Writememory

Callback functions

ODBG_Paused *

ODBG_Pausedex *

ODBG_Pluginaction

ODBG_Pluginclose

ODBG_Plugincmd *

ODBG_Plugindata

ODBG_Plugindestroy

ODBG_Plugininit

ODBG_Pluginmainloop

ODBG_Pluginmenu

ODBG_Pluginreset

ODBG_Pluginsaveudd

ODBG_Pluginshortcut

ODBG_Pluginuddrecord

Structures

t_asmmodel

t_bpoint *

t_disasm

t_dump

t_extmodel

t_hexstr

t_memory

t_module

t_operand

<u>t_ref</u>

t_reg *

<u>t_result</u>

t_sorted

t_sortheader

<u>t_table</u>

t_thread

t_window

Function prototypes

SORTFUNC

DESTFUNC

DRAWFUNC

Custom messages

WM_USER_BAR

WM_USER_CHALL

WM_USER_CHGS

WM_USER_CHMEM

WM_USER_CHREG

WM_USER_CNTS

WM_USER_DBLCLK

WM_USER_MENU

WM_USER_SCR

WM_USER_STS

WM_USER_VABS

WM_USER_VBYTE

WM_USER_VREL

Information functions

This group of functions displays error and information messages, adds messages to log window, shows scrollbar and flash:

void Addtolist (long addr,int highlight,char *format,...);

void <u>Updatelist(void);</u>

HWND <u>Createlistwindow</u>(void);

void Error(char *format,...);

```
void <u>Message(ulong addr,char *format,...);</u>
```

```
void Infoline(char *format,...);
```

void Progress(int promille,char *format,...);

```
void <u>Flash(char *format,...);</u>
```

Addtolist

The Addtolist function adds single line of ASCII text, up to <u>TEXTLEN</u> characters long, to the log window.

void Addtolist(long addr,int highlight,char *format,...);

Parameters:

addr - memory address associated with log line. By doubleclicking the line in log window, one can instantly jump to the corresponding code or data in CPU;

highlight - color of text:

0	standard color (black in black on white color scheme);
1	highlighted (red);
-1	grayed (gray);

format - format string (as in call to printf), followed by optional arguments.

See also: <u>Updatelist</u>, <u>Createlistwindow</u>, <u>Message</u>

Updatelist

If log window is present, call to this function forces immediate update of the log window. Call it if some operation takes plenty of time and you want to make new messages immediately available for user.

void Updatelist(void);

See also: <u>Addtolist</u>, <u>Createlistwindow</u>, <u>Message</u>

Createlistwindow

Creates or restores log window (window that displays contents of log buffer) on the screen. Note that writing to buffer doesn't depend on whether log window is present; closing log window doesn't destroy the contents of buffer.

HWND Createlistwindow(void);

See also: Addtolist, Updatelist, Message

Error

Displays message box with information about error. To continue, user must click OK button, press Enter or Esc. Use this call for critical errors only; if error is not very important, <u>Flash</u>, <u>Message</u> or <u>Infoline</u> are better alternatives.

void Error(char *format,...);

Parameters:

format - format string (as in call to printf), followed by optional arguments.

See also: Flash, Message, Infoline

Message

Displays message on the bottom of main OllyDbg window and adds it to the log window. If format is NULL, message will be removed from the bottom line but not added to the log. Formatted message may contain dollar sign '\$'. This symbol is replaced by dash '-' on the bottom line and terminates line added to the log. For example, if you call Message(0,"Critical error \$ press SPACE to continue"), bottom line will display "Critical error - press SPACE to continue" and log window "Critical error". Call to this function removes flash and progress bar from the bottom line.

void Message(ulong addr,char *format,...);

Parameters:

addr - memory address associated with log line. By doubleclicking the line in log window, one can instantly jump to the corresponding code or data in CPU. addr is not displayed in the bottom line;

format - format string (as in call to printf), followed by optional arguments.

See also: <u>Addtolist</u>, <u>Updatelist</u>, <u>Createlistwindow</u>, <u>Infoline</u>, <u>Progress</u>, <u>Flash</u>

Infoline

Displays message on the bottom of main OllyDbg window. If format is NULL, currently displayed message will be removed. Call to Infoline removes flash and progress bar from the bottom line.

void Infoline(char *format,...);

Parameters:

format - format string (as in call to printf), followed by optional arguments.

See also: <u>Addtolist</u>, <u>Updatelist</u>, <u>Createlistwindow</u>, <u>Message</u>, <u>Progress</u>, <u>Flash</u>

Progress

Displays progress bar on the bottom of main OllyDbg window. Bar will contain formatted text with attached percent of execution. Formatted text may contain dollar sign '\$', in this case persent of execution, enclosed in dashes, is inserted instead of dollra sign. If promille is 0, function closes progress bar restores previously displayed message. Calls to <u>Message</u>, <u>Infoline</u> and <u>Flash</u> also will close progress bar.

void Progress(int promille,char *format,...);

Parameters:

promille - progress, in 1/1000th;

format - format string (as in call to printf), followed by optional arguments.

See also: Message, Infoline, Flash

Flash

Displays highlighted message on the bottom of main OllyDbg window. This message automatically disappears in 500 milliseconds.

void Flash(char *format,...);

Parameters:

format - format string (as in call to printf), followed by optional arguments.

See also: <u>Message</u>, <u>Infoline</u>, <u>Progress</u>

Data formatting functions

This group of functions converts binary data, like address, floating number or character to ASCII text. Functions <u>IstextA</u> and <u>IstextW</u> check whether ASCII or UNICODE character can be a part of string. <u>Isretaddr</u> checks whether address is a possible return address.

int Decodeaddress(ulong addr,ulong base,int addrmode,char *symb,int
nsymb,char *comment);

int <u>Decoderelativeoffset(ulong addr,int addrmode,char *symb,int nsymb);</u>

```
int <u>Decoderange(ulong addr,ulong size,char *s);</u>
```

int <u>Decodecharacter(char *s,uint c);</u>

int Decodeascii(ulong addr,char *s,int len,int mode);

int Decodeunicode(ulong addr,char *s,int len);

int <u>Printfloat4</u>(char *s,float f);

int <u>Printfloat8(char *s,double d);</u>

int <u>Printfloat10</u>(char *s,long double ext);

int Printsse(char *s,char *f);

int <u>Print3dnow(char *s,char *f);</u>

int <u>IstextA</u>(char c);

```
int <u>IstextW</u>(wchar_t w);
```

ulong <u>Isretaddr</u>(ulong retaddr,ulong *procaddr);

int <u>Stringtotext(char *data,int ndata,char *text,int ntext);</u>

Decodeaddress

Decodes memory address to text string and optionally comments it. Returns length of decoded string (not including terminal 0), or 0 on error. The decoding is strongly influenced by addrmode and may vary from simple 01234567 to constructs like <JMP.&USER32.GetSystemMetrics>. If address has both module- and user-defined names, user-defined name has priority and module-defined name is placed in comment.

int Decodeaddress(ulong addr,ulong base,int addrmode,char *symb,int nsymb,char *comment);

Parameters:

addr - address to decode in address space of debugged program;

base - address belonging to the module selected as current or 0 if there is no current module. Necessary if you set bits ADC_SAMEMOD or ADC_DIFFMOD;

addrmode - combination of ADC_xxx bits listed below, determines how to decode addr. Note that Decodeaddress does not support some of ADC_xxx declared in plugin.h:

ADC_VALID	decode address only if it points to allocated memory or has associated symbolic name;
ADC_INMODULE	decode address only if it points to some module or has associated symbolic name. If you want to avoid cases when some address points to gap between two memory blocks belonging to a module, specify both ADC_VALID and ADC_INMODULE flags;
ADC_SAMEMOD	decode address only if it points to module defined by parameter base or has associated symbolic name (constant or name belonging to different mnodule). Condition ADC_INMODULE is automatically true and flag need not to be explicitely specified.

ADC_SYMBOL	decode address only if it has symbolic name or if ADC_JUMP bit is set and address points to JMP to symbolic name;
ADC_JUMP	check whether addr points to JMP to address placed on some import address and decode it as <jmp.&module.importname>;</jmp.&module.importname>
ADC_DIFFMOD	display module name only if addr belongs to module which differs from the current (specified by base);
ADC_NOMODNAME	never display module name. If neither ADC_DIFFMOD nor ADC_NOMODNAME bits specified, module name is displayed when address belongs to some module;
ADC_OFFSET	if address has a symbolic name and points to data section, add word OFFSET before this name (for ex., OFFSET MODULE.DataName);
ADC_STRING	decode to comment the case when address points to ASCII or UNICODE string;
ADC_ENTRY	decode to comment the case when address is an entry point of some subroutine without symbolic name;

symb - pointer to buffer of length at least nsymb bytes where Decodeaddress places decoded string;

nsymb - length, in characters, of buffer symb;

comment - pointer to string of length at least <u>TEXTLEN</u> bytes or NULL, receives comment asociated with addr.

See also: <u>Decoderelativeoffset</u>, <u>Disasm</u>, <u>Decodeascii</u>, <u>Decodeunicode</u>

Decoderelativeoffset

If address points to a valid command within the named procedure, decodes address in form "module.procedure+offset" or "procedure+offset". Returns length of decoded string or 0 on error or when procedure is not named.

int Decoderelativeoffset(ulong addr,int addrmode,char *symb,int nsymb);

Parameters:

addr - absolute address to decode;

addrmode - combination of ADC_xxx bits listed below, determines how to decode addr. Note that <u>Decodeaddress</u> does not support some of ADC_xxx declared in plugin.h:

ADC_NOMODNAME if bit is cleared, prepend name of procedure with module name, otherwise module name is omitted ADC_NONTRIVIAL if offset is 0, do not decode relative offset

symb - pointer to buffer of length at least nsymb bytes where Decoderelativeoffset places decoded string;

nsymb - length, in characters, of buffer symb.

See also: <u>Decodeaddress</u>, <u>Decoderange</u>

Decoderange

Decodes address range, either in form "module:section" or "firstaddr..lastaddr". Returns length of resulting string.

int Decoderange(ulong addr,ulong size,char *s);

Parameters:

addr - start of address range;

size - size of address range;

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes that receives resulting string.

See also: <u>Decodeaddress</u>, <u>Decoderelativeoffset</u>

Decodecharacter

Decodes ASCII character c to string s and comments some characters with special meaning, like TAB, CR or LF. Returns length of decoded string or 0 on error.

int Decodecharacter(char *s,uint c);

Parameters:

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Decodecharacter places decoded string;

c - character to decode.

See also: <u>IstextA</u>, <u>IstextW</u>

Decodeascii

Decodes ASCII string that starts at address addr in the memory of debugged process into string s of length len. If mode is DASC_TEST or DASC_NOHEX, checks whether this really looks like a string, if DASC_ASCII - decodes as ASCII string, if DASC_PASCAL - decodes as Pascal string (not zero-terminated, preceded with byte length). If mode is DASC_NOHEX and value points to a string, precedes decoded string with "ASCII". Returns length of resulting text, not including terminal '\0'.

int Decodeascii(ulong addr,char *s,int len,int mode);

Parameters:

addr - address in the memory of debugged process where ASCII string starts;

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Decodeascii places decoded string;

len - length of string s in bytes;

mode - decoding mode, one of the following:

DASC_TEST	Test whether pointed data really looks like an ASCII string. If not, print hexadecimal address instead of string
DASC_NOHEX	Test whether pointed data really looks like an ASCII string. If not, return 0.
DASC_ASCII	Force ASCII string
DASC_PASCAL	Force Pascal string

See also: Decodeunicode, Decodeaddress, Decodecharacter

Decodeunicode

// Decodes UNICODE string that starts at address addr in the memory of debugged process into ASCII string s of length len. Returns length of resulting text, not including terminal '\0'.

int Decodeunicode(ulong addr,char *s,int len);

Parameters:

addr - address in the memory of debugged process where UNICODE string starts;

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Decodeunicode places decoded string;

len - length of string s in bytes.

See also: <u>Decodeascii</u>, <u>Decodeaddress</u>, <u>Decodecharacter</u>

Printfloat4

Decodes 32-bit (4-byte) floating point number to ASCII string. If number is INF or NAN, adds hexadecimal dump. Returns length of decoded string.

int Printfloat4(char *s,float f);

Parameters:

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Printfloat4 places decoded string;

f - 32-bit floating number to decode.

See also: Printfloat8, Printfloat10, Print3dnow, Printsse

Printfloat8

Decodes 64-bit (8-byte, double) floating point number to ASCII string. If number is INF or NAN, adds hexadecimal dump. Returns length of decoded string. Note that this procedure is safer than printf, because some printf implementations generate exception when processing INF or NAN.

int Printfloat8(char *s,double d);

Parameters:

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Printfloat8 places decoded string;

d - 64-bit (double) floating number to decode.

See also: Printfloat4, Printfloat10, Print3dnow, Printsse

Printfloat10

Decodes 80-bit (10-byte, long double) floating point number to ASCII string. If number is INF or NAN, adds hexadecimal dump. Returns length of decoded string. Note that this procedure is safer than printf, because some printf implementations generate exception when processing INF or NAN.

int Printfloat10(char *s,long double ext);

Parameters:

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Printfloat10 places decoded string;

ext - 80-bit (long double) floating number to decode.

See also: Printfloat4, Printfloat8, Print3dnow, Printsse

Printsse

Decodes 128-bit SSE consisting of 4 32-bit floating point numbers to ASCII string. If any component is INF or NAN, displays it as a hexadecimal dump. Returns length of decoded string. Note that this procedure is safer than printf, because some printf implementations generate exception when processing INF or NAN.

int Printsse(char *s,char *f);

Parameters:

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Printfsse places decoded string;

f - pointer to 16-byte array containing SSE to decode.

See also: Printfloat4, Printfloat8, Print3dnow

Print3dnow

Decodes 64-bit 3Dnow! number (consisting of two 32-bit floating numbers) to ASCII string. Returns length of decoded string.

int Print3dnow(char *s,char *f);

Parameters:

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes where Print3dnow places decoded string;

f - pointer to 8-byte buffer containing 3Dnow! number.

See also: Printfloat4, Printfloat8, Printfloat10, Printsse

IstextA

Returns PLAINASCII, DIACRITICAL or their combination if symbol can be part of valid ASCII text, and 0 otherwise. Result is influenced by option "Allow diacritical symbols in strings".

int IstextA(char c);

Parameters:

c - character to analyze.

See also: <u>IstextW</u>, <u>Decodecharacter</u>

IstextW

Returns non-zero if wide (UNICODE) character can be part of valid (from the OllyDbg's point of view) UNICODE string and 0 otherwise. Result is influenced by option "Allow diacritical symbols in strings".

int IstextW(wchar_t w);

Parameters:

w - wide character to analyze.

See also: <u>IstextA</u>, <u>Decodecharacter</u>

Isretaddr

Function checks whether retaddr is a possible return address, that is, points to the command that immediately follows CALL command. If procaddr is not NULL, sets procaddr to destination of CALL or to 0 if destination is not constant. Returns address of CALL command if retaddr is a possible return address and 0 otherwise.

ulong cdecl Isretaddr(ulong retaddr,ulong *procaddr);

Parameters:

retaddr - questioned address in memory space of debugged application;

procaddr - pointer to variable that receives start address of called function or NULL.

Stringtotext

Decodes ASCII data of length ndata (not necessarily NULL-terminated) into the string of length at least ntext bytes according to the mode of string decoding set in String options. Decoding stops either when ndata symbols are processed, or character '\0' is emcountered, or when output string is full. Returns length of resulting string or 0 on error.

Note: There are three decoding modes currently supported by OllyDbg:

plain	"abcdef"
Assembler	"abc",LF,"def"
С	"abc\ndef"

int Stringtotext(char *data,int ndata,char *text,int ntext);

Parameters:

data - pointer to input ASCII data of length ndata;

- ndata length of input data in bytes;
- text pointer to the buffer of length at least ntext that receives formated text;
- ntext size of output buffer in bytes.

Data input functions

These functions invoke dialog window allowing user to enter number or string and specify related options:

int Getlong(char *title,ulong *data,int datasize,char letter,int mode);

int Getline(char *title,ulong *data);

int Getfloat10(char *title,long double *fdata,char *tag,char letter,int mode);

int Getfloat(char *title,void *fdata,int size,char letter,int mode);

void Getasmfindmodel(t_asmmodel model[NMODELS], char letter, int searchall);

int Gettext(char *title,char *text,char letter,int type,int fontindex);

int Gethexstring(char *title, t_hexstr *hs, int mode, int fontindex, char letter);

int Getmmx(char *title,char *data,int mode);

int Get3dnow(char *title,char *data,int mode);

int <u>Browsefilename(char *title,char *name,char *defext,int getarguments);</u>

Most of the data input functions have ...xy counterpart allowing to specify the position of the dialog on the screen. Internally, non-xy functions just call xy-enabled functions with x=-1 and y=-1. Function <u>Getregxy</u> exists only in ...xy form:

int Getlongxy(char *title,ulong *data,int datasize,char letter,int mode,int x,int y);

int Getlinexy(char *title,ulong *data,int x,int y);

int Getfloat10xy(char *title,long double *fdata,char *tag,char letter,int mode,int
x,int y);

int Getfloatxy(char *title,void *fdata,int size,char letter,int mode,int x,int y);

void Getasmfindmodelxy(t_asmmodel model[NMODELS], char letter, int searchall, int x, int y);

int Gettextxy(char *title,char *text,char letter,int type,int fontindex,int x,int y);

int Gethexstringxy(char *title,t_hexstr *hs,int mode,int fontindex,char letter,int x,int y);

int Getregxy(char *title,ulong *data,char letter,int x,int y);

int Getmmxxy(char *title,char *data,int mode,int x,int y);

int Get3dnowxy(char *title,char *data,int mode,int x,int y);

Function <u>Gettableselectionxy</u> allows to calculate screen X-Y coordinates for standard (not user-drawn) table windows:

int Gettableselectionxy(t_table *pt,int column,int *px,int *py);

Getlong, **Getlong**xy

Functions display dialog allowing user to enter 8-, 16- or 32-bit integer number in any of 3 formats: hexadecimal, decimal unsigned or decimal signed, or (if bit DIA_HEXONLY is set) in hexadecimal format only. Optional checkboxes "Entire block" and "Aligned search" are controlled by bits DIA_ASKGLOBAL and DIA_ALIGNED and control global flags globalsearch and alignedsearch. Return 0 on success and -1 if error occured or user cancelled action. Function Getlongxy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Getlong(char *title,ulong *data,int datasize,char letter,int mode);

int Getlongxy(char *title,ulong *data,int datasize,char letter,int mode,int x,int y);

Parameters:

title - title of dialog box;

data - pointer to 32-bit buffer containing initial value of integer number. On return, buffer contains entered value. If user cancels action, value remains unchanged;

datasize - size of integer number in bytes (1, 2 or 4). Note that dependless on datasize, buffer pointed to by data nust be 32 bits (4 bytes) long;

letter - first character to be entered in default control, or 0 if there is no character. Useful if function is called as a reaction on a character entered by user;

mode - combination of DIA_xxx bits specifying additional Getlong features:

DIA_HEXONLY	hide decimal input windows
DIA_ASKGLOBAL	display checkbox "Entire block" that controls global search flag. Actual state of this flag is returned by call to <u>Plugingetvalue</u> (VAL_GLOBALSEARCH)
DIA ALIGNED	display checkbox "Aligned search" that controls aligned search flag. Actual state of this flag is
	returned by call to

Plugingetvalue(VAL_ALIGNEDSEARCH)

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Getregxy</u>, <u>Getline</u>, <u>Getfloat</u>, <u>Getfloat10</u>, <u>Getmmx</u>, <u>Get3dnow</u>, <u>Gettableselectionxy</u>

Getline, Getlinexy

Functions display dialog asking user to enter source line number in unsigned decimal format. Return 0 on success and -1 if error occured or user cancelled action. Function Getlinexy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Getline(char *title,ulong *data);

int Getlinexy(char *title,ulong *data,int x,int y);

Parameters:

title - title of dialog box;

data - pointer to 32-bit buffer containing initial value of line number. On return, buffer contains entered value. If user cancels action, value remains unchanged;

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Getlong</u>, <u>Getregxy</u>, <u>Getfloat</u>, <u>Getfloat10</u>, <u>Getmmx</u>, <u>Get3dnow</u>, <u>Gettableselectionxy</u>

Getfloat10, Getfloat10xy

Display dialog asking user to enter 80-bit floating point number, either as float or as hexadecimal code. Primarily oriented on editing of contents of FPU stack. If tag is not NULL, functions ask whether to change the associated FPU tag. If tag is NULL and bit DIA_ASKGLOBAL is set, ask whether to use global search. Bit DIA_ALIGNED enables boxes "Aligned search" and "Allow 0.1% error margin". Function Getfloat10 additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Getfloat10(char *title,long double *fdata,char *tag,char letter,int mode);

int Getfloat10xy(char *title,long double *fdata,char *tag,char letter,int mode,int x,int y);

Parameters:

title - title of dialog box;

fdata - pointer to 80-bit floating point number. On return, buffer contains entered value. If user cancels action, value remains unchanged;

tag - pointer to tag associated with FPU register. If user requested change of associated tag, Getfloat10 will set this tag to valid, zero or bad depending on the contents of *fdata;

letter - first character to be entered in edit control, or 0 if there is no character. Useful if function is called as a reaction on a numeric key pressed by user;

mode - combination of DIA_xxx bits specifying additional Getfloat10 features:

	display checkbox "Entire block" that controls global
DIA_ASKGLOBAL	search flag. Actual state of this flag is returned by
	call to Plugingetvalue (VAL_GLOBALSEARCH)
	display checkboxes "Aligned search" and "Allow
DIA_ALIGNED	0.1% error margin" that control aligned search and
	inexact search flags. Actual state of these flags is
	returned by calls to
	Plugingetvalue(VAL_ALIGNEDSEARCH) and

Plugingetvalue(VAL_SEARCHMARGIN)

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Getlong</u>, <u>Getregxy</u>, <u>Getline</u>, <u>Getfloat</u>, <u>Getmmx</u>, <u>Get3dnow</u>, <u>Gettableselectionxy</u>

Getfloat, Getfloatxy

Display dialog asking user to enter floating point number of specified precision (4, 8 or 10 bytes), either as float or as hexadecimal code. If bit DIA_ASKGLOBAL is set, ask whether to use global search. Bit DIA_ALIGNED enables boxes "Aligned search" and "Allow 0.1% error margin". Function Getfloatxy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Getfloat(char *title,void *fdata,int size,char letter,int mode);

int Getfloatxy(char *title,void *fdata,int size,char letter,int mode,int x,int y);

Parameters:

title - title of dialog box;

fdata - pointer to floating point number. On return, buffer contains entered value. If user cancels action, value remains unchanged;

size - size of floating point number in bytes (4,8 or 10);

letter - first character to be entered in edit control, or 0 if there is no character. Useful if function is called as a reaction on a key pressed by user;

mode - combination of DIA_xxx bits specifying additional Getfloat features:

	display checkbox "Entire block" that controls global
DIA_ASKGLOBAL search flag. Actual state of this flag is returned by call to <u>Plugingetvalue</u> (VAL_GLOBALSEARCH)	
0.1% error margin" that control aligned search and	
inexact search flags. Actual state of these flags is	
returned by calls to	
Plugingetvalue(VAL_ALIGNEDSEARCH) and	
Plugingetvalue(VAL_SEARCHMARGIN)	

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it

remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Getfloat10</u>, <u>Getlong</u>, <u>Getregxy</u>, <u>Getline</u>, <u>Getmmx</u>, <u>Get3dnow</u>, <u>Gettableselectionxy</u>

Getasmfindmodel, Getasmfindmodelxy

Display dialog box allowing user to enter assembler command (imprecise commands are also accepted) and create set of search models. If user cancels input, model[0].length is 0. Function Getasmfindmodelxy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

void Getasmfindmodel(t_asmmodel model[NMODELS],char letter,int searchall);

void Getasmfindmodelxy(t_asmmodel model[NMODELS],char letter,int searchall,int x,int y);

Parameters:

model - pointer of array of NMODELS <u>t_asmmodel</u> structures that receives set of models created by Getasmfindmodel on success;

letter - first character to be entered in edit control, or 0 if there is no character. Useful if function is called as a reaction on a key pressed by user;

searchall - if nonzero, hides checkbox "Entire block" that controls global search flag. Actual state of this flag is returned by call to <u>Plugingetvalue</u>(VAL_GLOBALSEARCH);

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Gettext</u>, <u>Gethexstring</u>, <u>Getlong</u>, <u>t_asmmodel</u>, <u>Gettableselectionxy</u>

MAXCMDSIZE

Constant that determines maximal possible length of the valid 80x86 command (16 bytes). You may argue that maximal allowed length is 15; that's correct, but 16 is a power of 2 and so seems more preferrable in a computer program.

#define MAXCMDSIZE 16 // Maximal length of 80x86 command

TEXTLEN

Constant that determines maximal possible length of names, text strings and messages in OllyDbg. As a general rule, if function returns string and does not contain its maximal length as an input parameter, the size of string buffer must be at least TEXTLEN characters (or 2*TEXTLEN bytes for UNICODE strings). File names are an exception, they are always MAXPATH bytes long. All other exceptions from this rule are clearly documented here.

#define TEXTLEN 256 // Maximal length of text string

t_asmmodel

Type of structure that keeps assembler search model.

typedef struct t_asmmodel { // Model to search for assembler command

char code[MAXCMDSIZE]; // Binary code

char mask[MAXCMDSIZE]; // Mask for binary code (0: bit ignored)

int length; // Length of code, bytes (0: empty)

int jmpsize; // Offset size if relative jump

int jmpoffset; // Offset relative to IP

int jmppos; // Position of jump offset in command

} t_asmmodel;

Members:

code - binary code of the command. Only bits that have 1's set in corresponding mask bits are significant;

mask - comparison mask. Search routine ignores all code bits where mask is set to 0;

length - length of code and mask, bytes. If length is 0, search model is empty or invalid;

jmpsize - if nonzero, command is a relative jump and jmpsize is a size of offset in bytes;

jmpoffset - if jmpsize is nonzero, jump offset relative to address of the following command, otherwise undefined;

jmppos - if jmpsize is nonzero, position of the first byte of the offset in code, otherwise undefined.

See also: Getasmfindmodel

Gettext, Gettextxy

Display dialog box allowing user to enter or edit ASCII text string. This dialog contains combobox with several last entered strings of specified type. For some predefined string types, these strings are saved to the .udd file. Return length of entered string or -1 on error or when user cancelled input. Function Gettextxy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Gettext(char *title,char *text,char letter,int type,int fontindex);

int Gettextxy(char *title,char *text,char letter,int type,int fontindex,int x,int y);

Parameters:

title - title of dialog box;

text - pointer to buffer at least <u>TEXTLEN</u> bytes long that receives entered string;

letter - first character to be entered in edit control, or 0 if there is no character. Useful if function is called as a reaction on a key pressed by user;

type - type of saved strings (0..255). Some string types (<u>NM_xxx</u> or <u>NM_xxx</u>|NMHISTORY) are predefined. In general, it is safe to use types in range 192..254, of course, if they are not used by other plugins. Contact me if you need unique type that is automatically saved to .udd file;

fontindex - index of OllyDbg font used in edit control and combobox. Use either FIXEDFONT or, if <u>Plugingetvalue</u>(VAL_WINDOWFONT) returns non-zero, index of font used in parent window;

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Plugingetvalue</u>, <u>Gethexstring</u>, <u>Browsefilename</u>, <u>Gettableselectionxy</u>

Gethexstring, Gethexstringxy

Display dialog box allowing user to enter or edit masked ASCII, UNICODE or hexadecimal string. Return 0 on success and -1 on error or when user cancelled input. Function Gethextsringxy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Gethexstring(char *title,t_hexstr *hs,int mode,int fontindex,char letter);

int Gethexstringxy(char *title,<u>t_hexstr</u> *hs,int mode,int fontindex,char letter,int x,int y);

Parameters:

title - title of dialog box;

hs - pointer to string descriptor that contains initial data to be displayed in the dialog and on exit contains masked string entered by user;

mode - combination of DIA_xxx bits specifying additional options. Options DIA_DEFHEX, DIA_DEFASCII and DIA_DEFUNICODE are mutually exclusive:

DIA_ASKGLOBAL	if this bit is cleared, dialog contains "Keep size" checkbox; if bit is set, dialog contains checkboxes "Entire block" that controls global search flag and "Case sensitive" that controls case ignoring flag. Actual state of these three flags is returned by calls to <u>Plugingetvalue</u> (VAL_KEEPSELSIZE), <u>Plugingetvalue</u> (VAL_GLOBALSEARCH) and <u>Plugingetvalue</u> (VAL_IGNORECASE)
DIA_DEFHEX	default data type is hexadecimal
DIA_DEFASCII	default data type is ASCII
DIA_DEFUNICODE default data type is UNICODE	

fontindex - index of OllyDbg font used in edit controls and comboboxes. Use either FIXEDFONT or, if <u>Plugingetvalue</u>(VAL_WINDOWFONT) returns non-zero, index of font used in parent window;

letter - first character to be entered in active edit control, or 0 if there is no character. Useful if function is called as a reaction on a key pressed by user;

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Plugingetvalue</u>, <u>Gettext</u>, <u>Browsefilename</u>, <u>t_hexstr</u>, <u>Gettableselectionxy</u>

t_hexstr

Type of structure that keeps masked binary string.

typedef struct t_hexstr { // String used for hex/text search

int n; // String length

char data[TEXTLEN]; // Data

char mask[TEXTLEN]; // Mask, 0 bits are masked

} t_hexstr;

Members:

n - length of the string in bytes;

data - array with string data. Only those data bits are significant which has 1 in corresponding bits of mask;

mask - array with mask data.

See also: <u>Gethexstring</u>

Getregxy

Similar to Getlongxy, display dialog allowing user to enter 32-bit integer number in any of 4 formats: hexadecimal, decimal unsigned, decimal signed or as a set of 4 characters. Intended primarily to edit contents of general-purpose registers EAX, EBX, CX and EDX. Returns 0 on success and -1 if error occured or user cancelled action.

int Getregxy(char *title,ulong *data,char letter,int x,int y);

Parameters:

title - title of dialog box;

data - pointer to 32-bit buffer containing initial value of integer number. On return, buffer contains entered value. If user cancels action, value remains unchanged;

letter - first hexadecimal character to be entered in hex control, or 0 if there is no character. Useful if function is called as a reaction on a character entered by user;

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Getlongxy</u>, <u>Getline</u>, <u>Getfloat</u>, <u>Getfloat10</u>, <u>Getmmx</u>, <u>Get3dnow</u>, <u>Gettableselectionxy</u>

Getmmx, Getmmxxy

Display dialog box allowing user to enter or edit 64-bit MMX number as a combination of 8-, 16- or 32-bit integers in signed decimal, unsigned decimal or hexadecimal formats. Return 0 on success and -1 on error or when user cancelled input. Function Getmmxxy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Getmmx(char *title,char *data,int mode);

int Getmmxxy(char *title,char *data,int mode,int x,int y);

Parameters:

title - title of dialog box;

data - pointer to 64-bit (8-byte) memory area containing initial value of MMX number. On exit, contains number modified by user;

mode - reserved, must be 0;

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: <u>Getlong</u>, <u>Getregxy</u>, <u>Getfloat</u>, <u>Getfloat10</u>, <u>Get3dnow</u>, <u>Gettableselectionxy</u>

Get3dnow, Get3dnowxy

Display dialog box allowing user to enter or edit 64-bit 3DNow! number as a combination of two floating-point or hexadecimal 32-bit numbers. Return 0 on success and -1 on error or when user cancelled input. Function Get3dnowxy additionally contains the preferred screen coordinates of the bottom left point of the dialog window.

int Get3dnow(char *title,char *data,int mode);

int Get3dnowxy(char *title,char *data,int mode,int x,int y);

Parameters:

title - title of dialog box;

data - pointer to 64-bit (8-byte) memory area containing initial value of 3DNow! number. On exit, contains number modified by user;

mode - reserved, must be 0;

x - absolute X screen coordinate, in pixels, of the bottom left corner of the dialog window. If necessary, dialog will automatically adjust its position so that it remains visible;

y - absolute Y screen coordinate, in pixels, of the bottom left corner of the dialog window.

See also: Getlong, Getregxy, Getfloat, Getfloat10, Getmmx, Gettableselectionxy

Gettableselectionxy

Calculates screen coordinates of the left top corner of the first visible selected line in the specified column of table window. Returns 0 on success and -1 if coordinates cannot be computed or table is user-defined.

Note: this function fails if table is user-defined!

int Gettableselectionxy(t_table *pt,int column,int *px,int *py);

Parameters:

pt - pointer to descriptor of table window;

column - column in table;

px - pointer to variable that receives X coordinate (in pixels of the screen). Either px or py (but not both) can be NULL;

py - pointer to variable that receives Y coordinate (in pixels of the screen).

See also: Data input functions

Browsefilename

Opens dialog box allowing user to select file name and additional file-related options, according to specified mode. In modes 0, 1 and 2 returns TRUE if valid file was selected and FALSE in any other case.

int Browsefilename(char *title,char *name,char *defext,int mode);

Parameters:

title - title of dialog box;

name - pointer to buffer containing initial file name, at least MAXPATH bytes long. On exit, contains name of file selected by user;

defext - pointer to string containing set of one or several default extentions. First extention must start with point ('.'). To specify several extentions, separate them with vertical line ('|'). To specify several extentions as a single selection, separate them with ";*" (like ".exe;*.dll"). Browsefilename knows several types of extentions and their combinations and automatically comments them;

mode - mode of operation. Modes 3 to 8 are not intended for use in plugins and are not described here:

- 0 standard dialog without additional elements
- 1 dialog with combobox "Arguments"
- 2 dialog with checkbox "Append to existing file"

New in version 1.10: if mode is ORed with 0x80, Browsefilename opens Save File dialog instead of Open File.

Sorted data functions

Many kinds of internal OllyDbg data consist of homogenous elements that has start and final address and do not overlap with each other. Good example is the table of memory blocks. Breakpoints may be treated as elements occupying 1 byte in memory space of debugged program. Threads exist in the address space of thread identifiers and also occupy 1 address of this space. Elements usually can be displayed in some window and sorted using some criterium. Set of such elements is called sorted data.

OllyDbg implements a powerful set of functions that allow easy operations with sorted data, like initilaization, adding or replacing of elements, removing of elements or address ranges, sorting, search and so on. OllyDbg automatically allocates new memory for sorted data if necessary.

Elements of sorted data are always kept sorted by address in a contiguous buffer. This allows for simple and extremely fast binary search. Adding new data is, of course, not so easy and can take significant time. Weighted binary trees may look as a better solution, but in our case data is read much more frequently than added to the table. If you sort data by method other than increasing addresses, OllyDbg simply creates additional array of indexes pointing to data elements.

All elements of sorted data begin with a standard 12-byte header:

typedef struct t_sortheader { // Header of sorted data field

ulong addr; // Base address of the element

ulong size; // Size occupied by element in address space

ulong type; // Type of data element, TY_xxx

} t_sortheader;

Please don't mix the size specified in this header and physical size of the element. They belong to different address spaces! Size in header is the size of piece of virtual address space described by sorted data and usually belongs to debugged program. Physical size of element is the size of memory ocuppied by element in the OllyDbg's memory. All elements have same physical size

necessary to fit all the characteristics and descriptions of the described object; size in header is simply one (albeit most important) of the object's characteristics and may be different for each object.

In most cases sorted data functions ignore type and you may use it as you want. Only <u>Deletenonconfirmedsorteddata</u> checks for bit TY_CONFIRMED and removes at once all elements where this bit is not set (a very fast way to get rid of unnecessary elements). Standard header can be followed by any additional fields. OllyDbg does not aligns data elements; to assure effective memory access, make physical size of element a multiple of 4 bytes.

There is a special kind of sorted data called autoarrangeable. Autoarrangeable data assumes that address of the element is simply its 0-based ordinal number in the data array and size occupied by element in address space is always 1. Even in this case, elements must begin with valid header. <u>Addsorteddata</u> always inserts new items to autoarrangeable data and never replaces existing.

To create your own table of sorted data, first of all you must allocate table descriptor (structure of type <u>t_sorted</u>) and initialize all its fields to 0. Then you call <u>Createsorteddata</u> to initialize table and allocate data buffers. After initialization, you can use all sorted data functions to change or retrieve data. Do not modify items of table descriptor directly, this may lead to severe data integrity problems!

Index array is allocated only if valid sortfunc is specified. To assure that sorted data is valid and correctly initialized, check that data pointer is not NULL. If n is 0, table is empty (but is not necessarily initialized).

Table version increments by 1 each time table of sorted data changes. This allows for easy implementation of small cache: if version is not changed, previously fetched data is still valid. In any imaginable application, wraparound of 32-bit variable is impossible. <u>Createsorteddata</u> initializes version to 1, so set cache version to 0 to indicate that cache is invalid.

If sorted is 0, index table was not updated after last modification of the data. To force sorting, call <u>Sortsorteddata</u>. If data is already sorted, <u>Sortsorteddata</u> returns immediately.

int <u>Createsorteddata(t_sorted</u> *sd,char *name,int itemsize,int nmax,<u>SORTFUNC</u> *sortfunc,<u>DESTFUNC</u> *destfunc); void Destroysorteddata(t_sorted *sd);

- void *<u>Addsorteddata</u> (t_sorted *sd,void *item);
- void Deletesorteddata(t_sorted *sd,ulong addr);
- void Deletesorteddatarange(t_sorted *sd,ulong addr0,ulong addr1);
- int <u>Deletenonconfirmedsorteddata(t_sorted</u> *sd);
- void* Findsorteddata(t_sorted *sd,ulong addr);
- void* <u>Findsorteddatarange(t_sorted</u> *sd,ulong addr0,ulong addr1);
- int <u>Findsorteddataindex(t_sorted</u> *sd,ulong addr0,ulong addr1);
- int <u>Sortsorteddata(t_sorted</u> *sd,int sort);
- void* Getsortedbyselection(t_sorted *sd,int index);

t_sorted

Type of descriptor of sorted data.

typedef struct t_sorted { // Descriptor of sorted table

char name[MAXPATH]; // Name of table, as appears in error messages

int n; // Actual number of entries

int nmax; // Maximal number of entries

int selected; // Index of selected entry or -1

ulong seladdr; // Base address of selected entry

int itemsize; // Size of single entry

ulong version; // Unique version of table

void *data; // Elements, sorted by address

SORTFUNC *sortfunc; // Function which sorts data or NULL

DESTFUNC *destfunc; // Destructor function or NULL

int sort; // Sorting criterium (column)

int sorted; // Whether indexes are sorted

int *index; // Indexes, sorted by criterium

int suppresserr; // Suppress multiple overflow errors

} t_sorted;

Members:

name - name of the sorted data, of no real importance. You can set it to empty string or use for your own purposes;

n - actual number of elements in sorted data;

nmax - maximal number of elements that fit in allocated memory. If necessary, <u>sorted data functions</u> allocate additional memory to fit new elements;

selected - index of selected entry in data sorted by specified criterium. Only when t_sorted.sorted is NULL or data is sorted by address, this index coincides with index in t_sorted.data;

seladdr - base address of selected element;

itemsize - size of element of sorted data in bytes ;

version - variable that increments by 1 each time the contents of sorted data is changed. One can use version to avoid unnecessary searches in sorted data: as long as version remains unchanged, pointers to elements of sorted data are valid. <u>Createsorteddata</u> initializes version to 1;

data - pointer to contiguous buffer that contains elements of sorted data sorted by address. If data is NULL, sorted data is not initialized;

sortfunc - pointer to function that sorts data by given criterium, or NULL if data is not sortable. See <u>SORTFUNC</u>;

destfunc - pointer to destructor function that frees resources allocated by element of sorted data, can be NULL if element doesn't allocate resources. See <u>DESTFUNC</u>;

sort - actual sorting criterium. OllyDbg passes this parameter to sortfunc;

sorted - flag indicating whether index array is actual;

index - array containing indexes of elements sorted by specified criterium. NULL if data is not initialized or sortfunc is NULL;

suppresserr - flag preventing from multiple error reports.

See also: <u>Sorted data functions</u>

Createsorteddata

Initializes descriptor of sorted data (structure <u>t_sorted</u>). If descriptor alseady contains data, this data is destroyed. Returns 0 on success and -1 on error.

int Createsorteddata(<u>t_sorted</u> *sd,char *name,int itemsize,int nmax,<u>SORTFUNC</u> *sortfunc,<u>DESTFUNC</u> *destfunc);

Parameters:

sd - pointer to descriptor of sorted data;

name - optional name of sorted data, can be NULL. OllyDbg uses this name only in some rare cases;

itemsize - size, in bytes, of the element of sorted data (including standard header);

nmax - initial number of data elements that allocated buffer can keep. If necessary, OllyDbg will automatically allocate additional memory;

sortfunc - pointer to function that compares two data elements according to sorting criterium, or NULL if data cannot be sorted. This criterium is usually the index of column in table window. If you specify AUTOARRANGE, data is autoarrangeable, that is, assumes that address of the element is simply its (0-based) ordinal number in the data and size of element is always 1. Even in this case, element must begin with valid header. Addsorteddata always inserts new items to autoarrangeable data and never replaces existing;

destfunc - pointer to function that is called for each element being removed from the table, or NULL if destructor is not necessary. You need destfunc, for example, if elements of sorted data allocate additional memory that must be freed before element is deleted.

See also: Destroysorteddata, SORTFUNC, DESTFUNC

SORTFUNC

Type of optional callback function used by OllyDbg to sort elements of sorted data according to some criterium. This function receives two pointers to elements of sorted data and sort criterium (which is usually the index of column in the window displaying sorted data). Function must return 0 if elements are equal, 1 if first element is greater (comes later) and -1 if first element is less than the second (comes earlier).

A special predefined sort pseudofunction AUTOARRANGE makes sorted data autoarrangeable. See <u>Createsorteddata</u> for details.

typedef int SORTFUNC(const t_sortheader *p1,const t_sortheader *p2,const int sort);

Parameters:

- p1 pointer to the first element;
- p2 pointer to the second element;
- sort sort criterium. I recommend that you use 0 to sort data by address.

See also: Createsorteddata, Sortsorteddata

DESTFUNC

Type of optional callback function used by OllyDbg to free resources allocated by element of sorted data when element is removed. Corresponds to destructor in C++ objects.

typedef void DESTFUNC(<u>t_sortheader</u> *pe);

Parameters:

pe - pointer to the element of sorted data to be removed.

See also: Createsorteddata

Destroysorteddata

Removes all elements from the sorted data and deallocates data memory. If sorted data has destructor function, this destructor will be called for each deleted element.

void Destroysorteddata(t_sorted *sd);

Parameters:

sd - pointer to descriptor of sorted data.

See also: Createsorteddata

Addsorteddata

Adds or replaces element in initialized sorted data. Returns pointer to item in the data if item is correctly added or replaced and NULL if either input parameters are invalid, data buffer is full and OllyDbg is unable to allocate more memory, new element cannot replace old because it is neither subset nor superset of the old item, or it overlaps with two or more existing elements. This pointer is valid till the next operation that adds or removes data. Do not change address or size of element after it is added to sorted data, this may lead to severe data integrity problems.

void *Addsorteddata(t_sorted *sd,void *item);

Parameters:

sd - pointer to initialized descriptor of sorted data;

item - pointer to new element.

See also: <u>Deletesorteddata</u>, <u>Deletesorteddatarange</u>, <u>Findsorteddata</u>, <u>Findsorteddatarange</u>, <u>Findsorteddataindex</u>

Deletesorteddata

Deletes element which begins exactly at specified address from sorted data.

void Deletesorteddata(t_sorted *sd,ulong addr);

Parameters:

sd - pointer to initialized descriptor of sorted data;

addr - address of element.

See also: <u>Deletesorteddatarange</u>, <u>Addsorteddata</u>, <u>Findsorteddata</u>, <u>Findsorteddatarange</u>, <u>Findsorteddataindex</u>

Deletesorteddatarange

Deletes all elements which contain at least 1 address within the specified range from the table of sorted data.

void Deletesorteddatarange(t_sorted *sd,ulong addr0,ulong addr1);

Parameters:

sd - pointer to initialized descriptor of sorted data;

addr0 - start of address range (included);

addr1 - end of address range (not included).

See also: <u>Deletesorteddata</u>, <u>Addsorteddata</u>, <u>Findsorteddata</u>, <u>Findsorteddatarange</u>, <u>Findsorteddataindex</u>

Deletenonconfirmedsorteddata

Deletes all elements with type bit TY_CONFIRMED reset to 0 from sorted data and resets this bit in all remaining elements. Returns number of deleted items. This is usually the fastest way to delete multiple non-adjacent elements from the sorted data. Autoarrangeable data cannot be deleted in this way.

int Deletenonconfirmedsorteddata(t_sorted *sd);

Parameters:

sd - pointer to initialized descriptor of sorted data.

See also: <u>Deletesorteddata</u>, <u>Deletesorteddatarange</u>

Findsorteddata

Searches for element containing specified address in sorted data. Returns pointer to found item on success and NULL on error or when there is no such item. Returned pointer is valid till the next operation that adds or removes data. Do not change address or size of element, this may lead to severe data integrity problems.

void *Findsorteddata(t_sorted *sd,ulong addr);

Parameters:

- sd pointer to initialized descriptor of sorted data;
- addr address in the address space of specified sorted data.

See also: Findsorteddatarange, Findsorteddataindex, Getsortedbyselection

Findsorteddatarange

Searches for the first element of sorted data containing address within the specified range. Returns pointer to found item on success and NULL on error or when there is no such item. Returned pointer is valid till the next operation that adds or removes data. Do not change address or size of element, this may lead to severe data integrity problems.

void* Findsorteddatarange(t_sorted *sd,ulong addr0,ulong addr1);

Parameters:

sd - pointer to initialized descriptor of sorted data;

addr0 - start of address range in the address space of specified sorted data (included);

addr1 - end of address range in the address space of specified sorted data (not included).

See also: Findsorteddata, Findsorteddataindex, Getsortedbyselection

Findsorteddataindex

Searches for the first element of sorted data containing address within the specified range. Returns index of found item on success and -1 on error or when there is no such item. Index is valid till the next operation that adds or removes data.

int Findsorteddataindex(t_sorted *sd,ulong addr0,ulong addr1);

Parameters:

sd - pointer to descriptor of sorted data;

addr0 - start of address range in the address space of specified sorted data (included);

addr1 - end of address range in the address space of specified sorted data (not included).

See also: Findsorteddata, Findsorteddatarange, Getsortedbyselection

Sortsorteddata

Sorts sorted data according to the specified sort criterium and saves results to the index array associated with sorted data. Returns 1 if data was updated and 0 otherwise.

int Sortsorteddata(<u>t_sorted</u> *sd,int sort);

Parameters:

sd - pointer to descriptor of sorted data;

sort - sort criterium.

See also: Createsorteddata, Getsortedbyselection, SORTFUNC

Getsortedbyselection

Returns pointer to element with specified index in sorted data sorted by actual criterium, or NULL on error. If necessary, function actualizes associated index table, so preliminary call to <u>Sortsorteddata</u> is not necessary. Function is very useful for extraction of selected element in table windows.

void* Getsortedbyselection(t_sorted *sd,int selection);

Parameters:

sd - pointer to descriptor of sorted data;

selection - zero-based index in data sorted by selected sort criterium.

See also: Sortsorteddata, Findsorteddata, Findsorteddatarange

Window functions

All MDI windows in OllyDbg are the so called table windows. They have up to 17 resizable columns, unlimited number of rows and hideable bar which can act as a string of buttons. OllyDbg supports resizing of columns and scrolling of table windows. For simple table windows, it automatically adds possibility to copy whole table, row or single element to clipboard without extra code. Table windows support UNICODE, highlighting and selection and several pseudographical symbols. User can select font and colour scheme, and so on.

Ordinary table windows display contents of sorted data. OllyDbg makes it especially easy for the programmer, one only needs to supply several relatively simple functions. For example, function that implements WM_PAINT functionality simply returns text to be drawn in specified cell, and function that allows to sort contents of window just compares two elements of sorted data.

Custom (user-defined) table windows may display any data. Disassembler and Dump are good examples of custom windows. They also obtain plenty of support from OllyDbg, but require significantly more programming.

Table windows are described by structure <u>t_table</u>. It is on the responsibility of the programmer to maintain data in custom windows. <u>Registerpluginclass</u> allocates 8 additional longwords accessible by SetWindowLong and GetWindowLong. First two longwords (with offsets 0 and 4) are reserved for internal use. You can freely use remaining offsets 8, 12, ..., 28.

typedef int <u>DRAWFUNC(char *s,char *mask,int *select,t_sortheader</u> *ps,int column);

void <u>Defaultbar(t_bar *pb);</u>

int Tablefunction(t_table *pt,HWND hw,UINT msg,WPARAM wp,LPARAM
lp);

void <u>Painttable(HWND hw,t_table</u> *pt,<u>DRAWFUNC</u> getline);

void <u>Selectandscroll(t_table</u> *pt,int index,int mode);

void <u>Sendshortcut(int where,ulong addr,int msg,int ctrl,int shift,int vkcode);</u>

HWND <u>Newtablewindow(t_table</u> *pt,int nlines,int maxcolumns,char *winclass,char *wintitle);

HWND <u>Quicktablewindow(t_table</u> *pt,int nlines,int maxcolumns,char *winclass,char *wintitle);

int **Broadcast** (UINT msg,WPARAM wp,LPARAM lp);

HWND <u>Createdumpwindow</u>(char *name,ulong base,ulong size,ulong addr,int type,SPECFUNC *specdump);

void <u>Setdumptype(t_dump</u> *pd,int type);

void Dumpbackup(t_dump *pd,int action);

HWND <u>Createwatchwindow</u>(void);

HWND <u>Createwinwindow</u>(void);

HWND <u>Creatertracewindow</u>(void);

HWND <u>Createthreadwindow(void);</u>

HWND <u>Createpatchwindow</u>(void);

Createwatchwindow

Creates new or brings to top existing window that contains watches. Only one such window may exist at a time. Returns handle of the window or NULL on error.

HWND Createwatchwindow(void);

Createwinwindow

Creates new or brings to top existing window that lists all windows (including childs) created by debugged application. Only one such window may exist at a time. Returns handle of the window or NULL on error.

HWND Createwinwindow(void);

Createthreadwindow

Creates new or brings to top existing window that lists all threads of debugged application. Only one such window may exist at a time. Returns handle of the window or NULL on error.

HWND Createthreadwindow(void);

Createpatchwindow

Creates new or brings to top existing window that lists patches applied to debugged application in current and previous sessions. Only one such window may exist at a time. Returns handle of the window or NULL on error.

HWND Createpatchwindow(void);

t_table

Type of descriptor of table of <u>sorted data</u>. Starting from the version 1.08, this structure contains two new elements: colsel and hilite. To keep it backward compatible with previous versions, I have splitted hscroll and scheme into two short 16-bit variables each.

typedef struct t_table { // Window with sorted data and bar

HWND hw; // Handle of window or NULL

t_sorted data; // Sorted data

t_bar bar; // Bar

int showbar; // Bar: 1-displayed, 0-hidden, -1-absent

short hscroll; // Horiz. scroll: 1-displayed, 0-hidden

short colsel; // Active column in TABLE_COLSEL window

int mode; // Combination of bits TABLE_xxx

int font; // Font used by window

short scheme; // Colour scheme used by window

short hilite; // Code highlighting scheme used by window

int offset; // First displayed row

int xshift; // Shift in X direction, pixels

DRAWFUNC *drawfunc; // Function which decodes table fields

} t_table;

Members:

hw - handle of window that displays contents of the table, or NULL if there is no

associated window;

data - descriptor of sorted data;

bar - descriptor of columns and bar buttons in the window;

showbar - status of the bar in window: 1 - bar visible, 0 - hidden, -1 - bar is permanently hidden;

hscroll - flag indicating presence of the horizontal scroll in the window;

colsel - column with selection in TABLE_COLSEL window. Ordinary sorted data windows select complete row; TABLE_COLSEL windows select single cell in the table;

mode - combination of bits TABLE_xxx describing additional table properties. Plugins can use following bits:

TABLE_DIR	Bottom-to-top table with reversed order of lines. Log window is an example of the bottom-to-top table
TABLE_COPYMENU	Attach copy menu item
TABLE_SORTMENU	Attach sort menu
TABLE_APPMENU	Attach appearance menu
TABLE_WIDECOL	Attach wide columns menu item
TABLE_USERDEF	User-drawn table
TABLE_NOHSCR	Table contains no horizontal scroll
TABLE_SAVEPOS	Save position of window to the .ini file
TABLE_FASTSEL	Update whel selection changes
TABLE_HILMENU	Attach highlighting menu
TABLE_ONTOP	Attach Always on top menu

font - index of font used to paint window;

scheme - colour scheme used to paint window;

hilite - code highlighting scheme used to display disassembled code, or 0 if

highlighting is disabled or not applicable;

offset - index of first row visible in the window;

xshift - horizontal shift in pixels;

drawfunc - function that prepares data used to paint window, see **DRAWFUNC**.

DRAWFUNC

Type of pointer to callback function that prepares data for painting in table windows. Given line and column, function must prepare ASCII or UNICODE string that will be displayed on their intersection. If string contains graphical symbols, or when it uses different colors, function must fill mask with individual graphical attributes for each character. Function returns number of characters (UNICODE: wide characters) in prepared string. String is not necessarily null-terminated.

For standard table windows (bit TABLE_USERDEF in <u>t_table</u>.mode is cleared), parameter ps points directly to the element of sorted data.

For user-defined table window (TABLE_USERDEF is set), ps is a pointer to the structure t_table that describes this window. Before OllyDbg calls DRAWFUNC, it sets t_table.offset to the index of currently processed line in table window (topmost displayed line has index 0) and sets table.data.n to the total number of completely or partially visible lines. Drawing function is called once for every crossing of visible row with visible column. Individual decoding of each item may impose severe overhead and make drawing slow. So OllyDbg sets table.data.net only once at the beginning of the sequence. Drawing function may use it as a command to prepare the entire block of requested data in some static buffer and then reset n to 0. It is guaranteed that sequence of calls to DRAWFUNC will not be interrupted by call with different t_table.

To implement scrolling in custom window, its window procedure must process several <u>custom messages</u>.

typedef int DRAWFUNC(char *s,char *mask,int *select,<u>t_sortheader</u> *ps,int column);

Parameters:

s - pointer to buffer for output string of size at least 2*<u>TEXTLEN</u> characters. Length of returned string must not exceed <u>TEXTLEN</u> ASCII or UNICODE characters. If function returns UNICODE string, it must set bit DRAW_UNICODE in *select. String is not necessarily null-terminated;

mask - array of individual graphical attributes for every character in output

string. OllyDbg uses mask only if DRAWFUNC sets bit DRAW_MASK in *select. Each byte of the mask is a combination of bits DRAW_xxx, see detailed description below;

select - pointer to graphical attributes common to all characters in output string. *select is a combination of bits DRAW_xxx, see detailed description below;

ps - for standard table windows (without attribute TABLE_USERDEF), pointer to the element of sorted data to be decoded. For custom (user-defined) windows, cast ps to pointer to structure <u>t_table</u> that describes custom window, see detailed description above;

column - zero-based index of the processed column. Note that if column is not visible at all, OllyDbg does not call DRAWFUNC.

Meaning of bits DRAW_xxx

Mask and select consist of combination of bits DRAW_xxx. They are summarized in the table below. Note that bits which are not allowed in the mask may have values that don't fit into byte:

Bit allowed in:	select	mask	
DRAW_NORMAL	*	*	normal plain text
DRAW_GRAY	*	*	grayed text
DRAW_HILITE	*	*	highlighted text
DRAW_UL		*	underlined text
DRAW_SELECT	*	*	selected background
DRAW_EIP	*	*	inverted normal text/background
DRAW_BREAK	*	*	breakpoint background
DRAW_GRAPH		*	graphical symbol, see below
DRAW_DIRECT		*	direct text and background colour indices
DRAW_MASK	*		use individual mask attributes for each symbol
DRAW_EXTSEL	*		extend selection from last mask till end of column
DRAW_UNICODE	*		text is in UNICODE

DRAW_TOP	*	draw top half of the text shifted 1/2 row down
DRAW_BOTTOM	*	draw bottom half of the text shifted $1/2$ row up

If entire string has same highlight and selection attributes, don't set DRAW_MASK. OllyDbg ignores mask and uses only attributes from *select. Attributes DRAW_NORMAL, DRAW_GRAY and DRAW_HILITE are mutually exclusive. You cannot set DRAW_EIP together with either DRAW_SELECT or DRAW_BREAK. If bits DRAW_BREAK and DRAW_SELECT are set simultaneously, background corresponds to that of conditional breakpoint.

To highlight and select each character individually, set DRAW_MASK in *select and fill in the mask with combination of bits describing corresponding character in output string. Bit DRAW_HILITE in the mask has priority over *select. Bits DRAW_GRAY, DRAW_SELECT, DRAW_EIP and DRAW_BREAK in *select have priority over remaining bits in mask. Mask also allows to draw pseudographical characters. If DRAW_GRAPH bit is set, character is decoded in a special way:

Symbol	Char	Meaning
D_SPACE	'N'	space
D_SEP	• •	thin vertical separating line
D_POINT	'.'	point
D_BEGIN	'B'	begin of procedure, loop or stack scope
D_BODY	'I'	body of procedure, loop or stack scope
D_ENTRY	'J'	loop entry point
D_LEAF	'K'	Intermediate leaf on a tree
D_END	'E'	end of procedure, loop or stack scope
D_SINGLE	'S'	scope consisting of single line
D_ENDBEG	'T'	begin and end of stack scope
D_JMPUP	'U'	small thin arrow upstairs (jump upstairs)
D_JMPOUT	'<'	short dash (jump to different module)
D_JMPDN	'D'	small thin arrow downstairs (jump downstairs)

D_PATHUP	'u'	start of highlighted jump path upstairs
D_GRAYUP	'V'	start of grayed jump path upstairs
D_PATHDN	'd'	start of highlighted jump path downstairs
D_GRAYDN	'e'	start of grayed jump path downstairs
D_PATH	'i'	body of highlighted jump path
D_GRAYPATH	'j'	body of grayed jump path
D_PATHUPEND	'r'	end of highlighted jump path upstairs
D_GRAYUPEND	's'	end of grayed jump path upstairs
D_PATHDNEND	'f'	end of highlighted jump path downstairs
D_GRAYDNEND) 'g'	end of grayed jump path downstairs
D_PATHPTUP	'a'	jump entry upstairs (highlighted)
D_PATHPTDN	'h'	jump entry downstairs (highlighted)
D_PATHEND	'z'	two-sided end of jump (highlighted)
D_SWTOP	't'	start of switch
D_SWBODY	'b'	switch body
D_CASE	'C'	intermediate switch case
D_LASTCASE	'l'	last switch case

Any other character is displayed as space.

OllyDbg allows direct setting of foreground and background colour for each character in the string. To use this feature, allow mask in *select and fill corresponding mask bytes with the following data:

DRAW_DIRECT ORed with **background colour** ORed with **foreground colour**,

where backgrond colour is one of BKxxx constants defined in plugin.h (BKTRANSP for default background), and foreground colour is any colour in range 0..15. Colours 16 to 19 are not supported. You can't combine DRAW_DIRECT with any other DRAW_xxx flags in the mask.

If bit BAR_SHIFTSEL is set for the actual column, background will be shifted 1/2 character to the left. This is a nice trick allowing better highlighting. In this case assure that last highlighted character is a space.

OllyDbg's Register window is also a custom table window. Please have a close look on EIP and EFL: they are shifted down by 1/2 line! How is it possible? Well, here I use another trick: I draw these lines twice, first time with bit DRAW_TOP and second time with bit DRAW_BOTTOM. However, this trick is relatively time-consuming, and mouse will select within each complete line. I do not recommended it for the future.

Defaultbar

Sets default widths of the columns in table window in accordance with currently selected font. You must redraw window to make effect of this function visible.

void Defaultbar(t_bar *pb);

Parameters:

pb - pointer to bar descriptor.

Tablefunction

Default window function for all table windows, implements most of their functionality. Call it only as a reaction on received WM_xxx message. Return value depends on the message, it is safe to pass this value to the operating system. For standard table windows, always pass following messages to Tablefunction:

WM_DESTROY

WM_MOUSEMOVE

WM_LBUTTONDOWN

WM_LBUTTONDBLCLK

WM_LBUTTONUP

WM_RBUTTONDOWN

WM_RBUTTONDBLCLK

WM_HSCROLL

WM_VSCROLL

WM_TIMER (unprocessed messages only)

WM_KEYDOWN (unprocessed messages only)

WM_SYSKEYDOWN (unprocessed messages only)

WM_WINDOWPOSCHANGED (to support <u>Always on top</u> option)

Tablefunction also processes most of custom OllyDbg messages from standard table windows. Custom windows usually must process these messages by itself.

int Tablefunction(t_table *pt,HWND hw,UINT msg,WPARAM
wParam,LPARAM lParam);

Parameters:

pt - pointer to descriptor of table window;

hw, msg, wParam, lParam - message parameters as received from Windows.

See also: <u>Custom messages</u>

Custom messages

OllyDbg defines following custom messages that must be processed by table windows:

WM_USER_MENU		activate context-sensitive menu
WM_USER_SCR	(*)	redraw scroll(s)
<u>WM_USER_VABS</u>	(*)	scroll contents of window by lines
WM_USER_VREL	(*)	scroll contents of window by percent
WM_USER_VBYTE	(*)	scroll contents of window by bytes
WM_USER_STS	(*)	start selection in window
WM_USER_CNTS	(*)	continue selection in window
WM_USER_CHGS	(*)	move single-line selection
WM_USER_BAR		message from bar segment acting as button
WM_USER_DBLCLK	- -	doubleclick in column
WM_USER_CHALL		redraw (almost) everything
WM_USER_CHMEM		range of debuggee's memory changed
WM_USER_CHREG		debuggee's register(s) changed

Standard table windows usually redirect messages marked with asterisk (*) to <u>Tablefunction</u>.

See also: <u>Tablefunction</u>

WM_USER_MENU

Custom message sent to table window when user presses right mouse button or shortcut Alt+F10. Window should create and fill pop-up menu and pass this message to <u>Tablefunction</u> with menu handle in parameter lp. Window can use identifiers from 1 to MENU_SORT-1 (0x27F) and from MENU_APPMAX+1 (0x300) to MENU_PLUGIN-1. It can pass NULL if only standard menus are required.

<u>Tablefunction</u> checks for attributes listed in <u>t_table</u>.mode and performs following actions:

Attribute	Action
TABLE_COPYMENU	If some line is selected, adds menu item "Copy". This attribute also adds processing of keyboard shortcuts Ctrl+C and Ctrl+Ins
TABLE_SORTMENU	Adds submenu "Sort by" with a list of all bar segments without BAR_NOSORT. To hide part of the segment title in menu, separate it with '\$'
TABLE_APPMENU	Adds submenu "Appearance" that includes bar, column, font and colour options
TABLE_WIDECOL	When set simultaneously with TABLE_APPMENU, adds menu item "Wide columns", allowing to double default widths
TABLE_HILMENU	When set simultaneously with TABLE_APPMENU, adds menu item "Highlighting", allowing to select one of code highlighting schemes
TABLE_ONTOP	Adds menu item " <u>Always on top</u> " that allows to keep one MDI window always visible

On return from <u>Tablefunction</u>, window gets id of selected item. If selection is processed internally by <u>Tablefunction</u>, or when there is no selection, it gets 0. Window then must destroy all newly created menus, process selection and return to caller.

See also: <u>Tablefunction</u>

WM_USER_SCR

Asks window to update horizontal and vertical scrollbars. Simply pass this message to <u>Tablefunction</u>.

WM_USER_VABS

This message requests table window to scroll vertically by (signed) number of lines specified in lParam. Positive lParam means scrolling forward in data (contents of window moves up), negative - backward. wParam contains number of data lines completely visible in the window (1 if data area is smaller than 1 line). If lParam is 0, message requests to calculate new position of vertical scrollbar.

Standard table window should simply pass this message to <u>Tablefunction</u>.

Owner-drawn window must modify table data but neither redraw nor invalidate the window. If window's appearance remains unchanged and lParam is not 0, window function must return -1. If window supports byte scrolling, it must return (index of topmost line)*MAXTRACK/(total number of lines). If total number of lines is less than or equal to wParam, it returns 0. Otherwise, it must return (index of topmost line)*MAXTRACK/(total number of lines-wParam). As constant MAXTRACK is relatively big, use MulDiv to calculate return value.

WM_USER_VREL

This message requests vertical scrolling to the position relative to the total size of the table. wParam contains number of completely visible lines in the window (1 if data area is smaller than 1 line). lParam contains new scrolling position in 1.0/MAXTRACK parts of the total height of the table.

Standard table window should simply pass this message to **Tablefunction**.

If custom table window supports byte scrolling, it must make line with index (total number of lines)*lParam/MAXTRACK topmost visible in the window. If byte scrolling is not supported, it must be line (total number of lineswParam)*lParam/MAXTRACK. Window is not allowed to either redraw or invalidate the window. If window's appearance remains unchanged, window function must return -1. If window supports byte scrolling, it must return (topmost line)*MAXTRACK/(total number of lines). If total number of lines is less than or equal to wParam, it returns 0. Otherwise, it must return (topmost line)*MAXTRACK/(total number of lines-wParam). As constant MAXTRACK is relatively big, use MulDiv to calculate return value.

WM_USER_VBYTE

This message requests table window to scroll up or down lParam bytes. wParam contains number of completely visible lines in the window (1 if data area is smaller than 1 line).

Standard table window should simply pass this message to <u>Tablefunction</u> where it is interpreted as <u>WM_USER_VABS</u>.

Custom table window must modify data but neither redraw nor invalidate the window. If position of data remains unchanged, window's function must return -1. If window supports byte scrolling, it must return (topmost line)*MAXTRACK/(total number of lines). If total number of lines is less than or equal to wParam, it returns 0. Otherwise, it must return (topmost line)*MAXTRACK/(total number of lines-wParam). As constant MAXTRACK is relatively big, use MulDiv to calculate return value.

WM_USER_STS

Message requests table window to start selection. HIWORD(wParam) contains column where selection begins, LOWORD(wParam) - X offset within the column in character widths, lParam - Y offset within the window in character heigths.

Standard table window should simply pass this message to <u>Tablefunction</u>.

Custom table window must modify data to reflect start of selection but neither redraw nor invalidate the window. It must return 1 if screen appearance is changed, 0 if not and -1 if start of selection at this point is not possible.

WM_USER_CNTS

Message is sent to table window to continue selection started by <u>WM_USER_STS</u>. HIWORD(wParam) contains column with current end of selection, LOWORD(wParam) - X offset within the column in character widths, lParam - Y offset within the window in character heigths.

Standard table window should simply pass this message to <u>Tablefunction</u>.

Custom table window must modify data to reflect change of selection but must neither redraw nor invalidate the window. It returns 1 if screen appearance is changed and 0 if not.

WM_USER_CHGS

Message requests table window to change selection to single-line, move selection up or down by lParam lines and scroll window so that selection is still visible. Special lParam values of MOVETOP and MOVEBOTTOM move selection directly to first or last line in the table. wParam contains number of completely visible lines in the window (1 if data area is smaller than 1 line).

If window does not support single-line selection, it must scroll by specified number of lines.

Standard table window (which anyway does not allow multiline selection) should simply pass this message to <u>Tablefunction</u>.

Custom table window must modify data but neither redraw nor invalidate the window. If position of data remains unchanged, window's function must return -1. If window supports byte scrolling, it must return (topmost line)*MAXTRACK/(total number of lines). If total number of lines is less than or equal to wParam, it returns 0. Otherwise, it must return (topmost line)*MAXTRACK/(total number of lines-wParam). As constant MAXTRACK is relatively big, use MulDiv to calculate return value.

WM_USER_BAR

Bar segment with mode bit BAR_BUTTON works as a button and, when pressed, sends this message to the window which owns bar. wParam contains column, lParam is 0. OllyDbg ignores value returned by this message.

WM_USER_DBLCLK

When user doubleclicks left mouse button within the data area (but neither in bar nor over the dividing line), table window receives this message. HIWORD(wParam) contains column, LOWORD(wParam) - X offset within the column in character widths, lParam - Y offset within the window in rows. If window processes this message, it must return 1, otherwise doubleclick is treated as simple click.

WM_USER_CHALL

Due to changes in debugged application or display options, window must be updated. Window's procedure is expected to postpone redrawing using actual data and return CONT_BROADCAST.

WM_USER_CHMEM

Memory of debugged process in range from wParam (included) to lParam (not included) is possibly changed. Update window if necessary and return CONT_BROADCAST.

WM_USER_CHREG

Some registers of debugged process (general-purpose, FPU, MMX etc.) are changed. Update window if necessary and return CONT_BROADCAST.

Painttable

Implements processing of WM_PAINT message for all table windows. Call this function only when processing WM_PAINT.

void Painttable(HWND hw,<u>t_table</u> *pt,<u>DRAWFUNC</u> getline);

Parameters:

hw - handle of window to be redrawn;

pt - pointer to descriptor of table window;

getline - pointer to custom function that prepares data to be drawn in specified cell of table window.

See also: **DRAWFUNC**

Selectandscroll

Selects element of sorted data with specified index according to current sort mode and scrolls window so that selection is visible. This function neither redraws nor invalidates nor creates window and has no effect on owner-drawn table windows.

void Selectandscroll(t_table *pt,int index,int mode);

Parameters:

pt - pointer to descriptor of table window;

index - index of element of sorted data according to current sort mode;

mode - request for position of selected line in window. If mode is 0, this is always the topmost line, if 1 - line in the middle of the data area, 2 - selected automatically (recommended when calling function walks through all table entries).

Sendshortcut

Emulates either global keyboard shortcut or shortcut in some CPU subwindow. Designed primarily for use in command line plugin.

void Sendshortcut(int where,ulong addr,int msg,int ctrl,int shift,int vkcode);

Parameters:

where - addressee of the emulated keyboard shortcut:

PM_MAIN	Main window (global shortcut)
	,
PM_DISASM	CPU Disassembler
PM_CPUDUMP	CPU Dump
PM_CPUSTACK	CPU Stack
PM_CPUREGS	CPU Registers

addr - for all CPU subwindows except PM_CPUREGS, address to which shortcut is applied. Ignored if where is PM_CPUREGS or PM_MAIN;

msg - keyboard message to emulate: WM_KEYDOWN, WM_SYSKEYDOWN or WM_CHAR;

ctrl - emulated state of Control key on the keyboard (0 - released, 1 - pressed);

shift - emulated state of Shift key on the keyboard (0 - released, 1 - pressed);

vkcode - key to emulate, character or one of VK_xxx (for example, VK_F1 to emulate F1 key).

Quicktablewindow

If window already exists, restores it and brings to the top. Otherwise, sets default appearance parameters and creates new window. If record with window's title already exists in ollydbg.ini, table has TABLE_SAVEPOS attribute and option "Restore windows position and appearance" is selected, restores old position, size and appearance. Returns pointer to window or NULL on error. Note that alternative function, <u>Newtablewindow</u>, neither restores window nor changes its appearance.

HWND Quicktablewindow(<u>t_table</u> *pt,int nlines,int maxcolumns,char *winclass,char *wintitle);

Parameters:

pt - pointer to descriptor of table window;

nlines - preferred number of visible lines;

maxcolumns - preferred number of visible columns;

winclass - name of registered window class (for example, obtained from call to <u>Registerpluginclass</u>);

wintitle - window's title. If table has TABLE_SAVEPOS attribute, OllyDbg uses title to save and restore window's position and appearance.

See also: <u>Registerpluginclass</u>, <u>Newtablewindow</u>

Newtablewindow

Creates new table window. If record with window's title already exists in ollydbg.ini, table has TABLE_SAVEPOS attribute and option "Restore windows position and appearance" is selected, restores old position, size and appearance of the table window. Returns pointer to window or NULL on error. Note that alternative function, <u>Quicktablewindow</u>, restores window if it already exists and sets default appearance parameters.

HWND Newtablewindow(<u>t_table</u> *pt,int nlines,int maxcolumns,char *winclass,char *wintitle);

Parameters:

pt - pointer to descriptor of table window;

nlines - preferred number of visible lines;

maxcolumns - preferred number of visible columns;

winclass - name of registered window class (for example, obtained from call to <u>Registerpluginclass</u>);

wintitle - window's title. If table has TABLE_SAVEPOS attribute, OllyDbg uses title to save and restore window's position and appearance.

See also: <u>Registerpluginclass</u>, <u>Quicktablewindow</u>

Createdumpwindow

Creates new dump window that can show either context of file or memory range of debugged program in one of predefined dump formats. Returns handle of created window or NULL on error. Number of simultaneously displayed dump windows is (theoretically) unlimited.

HWND Createdumpwindow(char *name,ulong base,ulong size,ulong addr,int type,SPECFUNC *specdump);

Parameters:

name - if parameter size is 0, name of file to display, otherwise window's title or NULL, in this last case OllyDbg generates title automatically;

base - if size is 0, base is ignored, otherwise this is the base address of displayed memory range;

size - 0 if window should dump contents of file, or size of displayed memory range otherwise;

addr - address or offset of the first element displayed after window is created;

type - combination of dump type (one of DU_xxx), number of items per line ((n<<8) & DU_COUNT) and size of single item (l & DU_SIZE). For variable-length types size is 1. See table below for a list of commonly used dump types;

specdump - function that performs special data decoding, set to NULL.

Commonly used dump types:

0x01101 Hex/ASCII (16 bytes) 0x01081 Hex/ASCII (8 bytes) 0x0A101 Hex/UNICODE (16 bytes) 0x0A081 Hex/UNICODE (8 bytes) 0x02401 ASCII (64 chars) 0x02201 ASCII (32 chars) 0x03402 UNICODE (64 chars)

- 0x03202 UNICODE (32 chars)
- 0x04082 Signed short decimal
- 0x05082 Unsigned short decimal
- 0x06082 Short hex
- 0x04044 Signed long decimal
- 0x05044 Unsigned long decimal
- 0x06044 Long hex
- 0x08014 Address
- 0x0B041 Address with ASCII dump
- 0x0C041 Address with UNICODE dump
- 0x07044 32-bit float
- 0x07028 64-bit double
- 0x0701A 80-bit long double
- 0x09011 Disassemble
- 0x0D001 PE header

See also: <u>Setdumptype</u>, <u>Dumpbackup</u>

Setdumptype

Sets or changes type of information displayed in dump window. Window associated with pd is not updated, you must invalidate it to visualize this change.

void Setdumptype(t_dump *pd,int type);

Parameters:

pd - pointer to <u>dump descriptor</u>;

type - combination of dump type (one of DU_xxx), number of items per line ((n<<8) & DU_COUNT) and size of single item (l & DU_SIZE). For variable-length types size is 1. See table <u>here</u> for a list of commonly used dump types.

See also: <u>Createdumpwindow</u>, <u>Dumpbackup</u>

Dumpbackup

Function performs specified backup action (like creating or updating backup, reading backup from file, destroying backup etc.) on the dump. If action involves file operations (read data from file, save data or backup to file), user is prompted to select file name. Function neither redraws nor invalidates backup window.

void Dumpbackup(t_dump *pd,int action);

Parameters:

pd - pointer to <u>dump descriptor</u>;

action - constant that specifies requested backup action:

BKUP_CREATECreate or update backup copyBKUP_VIEWDATAView original dataBKUP_VIEWCOPYView backup copyBKUP_LOADCOPYRead backup copy from fileBKUP_SAVEDATASave original data to fileBKUP_DELETEDelete backup copy

See also: Createdumpwindow, Setdumptype

Broadcast

Function sends message to all open MDI windows. Stops either after message is sent to all windows or when some window returns STOP_BROADCAST. Usually used to broadcast custom messages <u>WM_USER_CHALL</u>, <u>WM_USER_CHMEM</u> and <u>WM_USER_CHREG</u>. Note that you don't need to broadcast <u>WM_USER_CHMEM</u> after call to <u>Writememory</u> with mode flag MM_RESTORE.

int Broadcast(UINT msg,WPARAM wParam,LPARAM lParam);

Parameters:

msg - message to be broadcasted;

wParam - first message parameter;

lParam - second message parameter.

See also: <u>Writememory</u>, <u>WM_USER_CHALL</u>, <u>WM_USER_CHMEM</u>, <u>WM_USER_CHREG</u>

Name functions

Any zero-terminated ASCII string that is shorter than <u>TEXTLEN</u> characters can be a name from the OllyDbg's point of view. Every name has associated 32-bit address and 8-bit type. OllyDbg stores all names in a huge centralized dynamical buffer that can keep up to 10,000,000 names, provided of course that you have enough memory. When used correctly, name functions are very fast.

Several name types are predefined:

NM_NONAME Undefined name NM_ANYNAME Name of any type

Names that are stored in the .udd file of module where they appear:

NM_LABEL	User-defined label
NM_EXPORT	Exported (global) name
NM_IMPORT	Imported name
NM_LIBRARY	Name extracted from library, object file or debug data
NM_CONST	User-defined constant (currently not implemented)
NM_COMMENT	User-defined comment
NM_LIBCOMM	Automatically generated comment from library or object file
NM_BREAK	Condition related with breakpoint
NM_ARG	Arguments decoded by analyser
NM_ANALYSE	Comment added by analyser
NM_BREAKEXPR	Expression related with breakpoint
NM_BREAKEXPL	Explanation related with breakpoint
NM_ASSUME	Assume function with known arguments
NM_STRUCT	Code structure decoded by analyzer
NM_CASE	Case description decoded by analyzer
NM_PLUGCMD	Plugin commands to execute at breakpoint

Names that are stored in the .udd file of main module:

NM_INSPECT	Several last entered inspect expressions
NM_WATCH	Watch expressions
NM_ASM	Several last entered assembled strings
NM_FINDASM	Several last entered assembler search strings
NM_LASTWATCH	Several last entered watch expressions
NM_SOURCE	Several last entered source search strings
NM_REFTXT	Several last entered reference text search strings
NM_GOTO	Several last expressions to follow in Disassembler
NM_GOTODUMP	Several last expressions to follow in Dump
NM_TRPAUSE	Several last expresions to pause run trace
NM_LABEL NMHISTORY	Several last entered user- defined labels
NM_COMMENT NMHISTORY	Several last entered user- defined comments
NM_BREAK NMHISTORY	Several last entered breakpoint conditions
NM_BREAKEXPR NMHISTORY	Several last entered breakpoint expressions
NM_BREAKEXPL NMHISTORY	Several last entered breakpoint explanations

If you need unique name type for your plugin, please contact the author of

OllyDbg.

To find name by its address, OllyDbg uses binary search on contiguous sorted index array. For this reason, search is extermely fast, but adding new names to the table may take significant time. If you need to add multiple names at once, use <u>Quickinsertname</u>. Names added in this way are unaccessible until you call <u>Mergequicknames</u>. As a rule of thumb, this method is preferrable if number of names exceeds 10-15.

int Insertname(ulong addr,int type,char *name);

int <u>Quickinsertname(ulong addr,int type,char *name);</u>

void <u>Mergequicknames(void);</u>

void <u>Discardquicknames(void);</u>

int Findname(ulong addr,int type,char *name);

int Decodename(ulong addr,int type,char *name);

ulong Findnextname(char *name);

int Findlabel(ulong addr,char *name);

void Deletenamerange(ulong addr0,ulong addr1,int type);

int Findlabelbyname(char *name,ulong *addr,ulong addr0,ulong addr1);

ulong Findimportbyname(char *name,ulong addr0,ulong addr1);

int <u>Demanglename(char *name,int type,char *undecorated);</u>

int <u>Findsymbolicname(ulong addr,char *fname);</u>

Insertname

Inserts new or replaces existing name of given type in the name table. If name is NULL or empty, entry is deleted. Returns 0 on success and -1 on error. Note: do not call this function between calls to <u>Quickinsertname</u> and <u>Mergequicknames</u>!

int Insertname(ulong addr,int type,char *name);

Parameters:

addr - name address;

type - name type (<u>NM_xxx</u> for predefined types);

name - name to insert. If name is NULL or empty, entry is removed from the name table.

See also: <u>Quickinsertname</u>, <u>Mergequicknames</u>, <u>Discardquicknames</u>, <u>Findname</u>, <u>Deletenamerange</u>

Quickinsertname

Inserts new or replaces existing name of given type in the name table. NULL or empty names are not allowed. Returns 0 on success and -1 on error. Names added by this function are unavailable until you call <u>Mergequicknames</u>. If you add multiple names, Quickinsertname is much faster than <u>Insertname</u>. Note: do not call <u>Insertname</u> between calls to Quickinsertname and <u>Mergequicknames</u>!

int Quickinsertname(ulong addr,int type,char *name);

Parameters:

addr - name address;

type - name type (<u>NM_xxx</u> for predefined types);

name - name to insert. If name is NULL or empty, entry is removed from the name table.

See also: <u>Insertname</u>, <u>Mergequicknames</u>, <u>Discardquicknames</u>, <u>Findname</u>, <u>Deletenamerange</u>

Mergequicknames

Function adds names posted by <u>Quickinsertname</u> to the name table. Note that posted names are not available until you call Mergequicknames.

void Mergequicknames(void);

See also: <u>Quickinsertname</u>, <u>Insertname</u>, <u>Discardquicknames</u>

Discardquicknames

Discards all names posted by <u>Quickinsertname</u> after last call to <u>Mergequicknames</u>.

void Discardquicknames(void);

See also: <u>Quickinsertname</u>, <u>Mergequicknames</u>

Findname

Searches for name with given address and type. Returns length of the name or 0 if name is absent. As a side effect, sets global arguments for <u>Findnextname</u>.

int Findname(ulong addr,int type,char *name);

Parameters:

addr - name address;

type - name type (<u>NM_xxx</u> for predefined types);

name - pointer to buffer of length at least <u>TEXTLEN</u> characters or NULL. If name is found, function copies it to this buffer.

Se also: <u>Findnextname</u>, <u>Decodename</u>, <u>Findlabel</u>, <u>Findlabelbyname</u>, <u>Findimportbyname</u>

Decodename

Searches for name with given address and type. If name is found, scans it for combinations <+XXXXXXX>, where XXXXXXXX is a hexadecimal number, and substitutes them by sum of base and XXXXXXXX in hexadecimal format. Returns length of resulting string or 0 if name is absent. OllyDbg uses this function to correct automatically generated comments in relocatable modules.

int Decodename(ulong addr,int type,char *name);

Parameters:

addr - name address;

- type name type (<u>NM_xxx</u> for predefined types);
- name pointer to output buffer of length at least <u>TEXTLEN</u> characters.

See also: Findname, Findlabel, Findlabelbyname, Findimportbyname

Findnextname

Searches for name with type specified in last call to <u>Findname</u> and address exceeding that in <u>Findname</u> or returned by last call to Findnextname. Returns address or 0 if there are no more compatible entries. If name is NULL, name itself is not fetched.

ulong Findnextname(char *name);

Parameters:

name - pointer to output buffer of length at least <u>TEXTLEN</u> characters.

See also: Findname, Findlabel, Findlabelbyname, Findimportbyname

Findlabel

Searches for name of types NM_LABEL, NM_EXPORT, NM_IMPORT, NM_LIBRARY, NM_CONST (in the listed order). If some name is found, gets name and returns its type, otherwise returns NM_NONAME.

int Findlabel(ulong addr,char *name);

Parameters:

addr - name address;

name - pointer to output buffer of length at least <u>TEXTLEN</u> characters or NULL.

See also: <u>Findname</u>, <u>Findlabelbyname</u>, <u>Findimportbyname</u>

Deletenamerange

Deletes all names of specified type (or all names if type is NM_ANYNAME) in the specified range.

void Deletenamerange(ulong addr0,ulong addr1,int type);

Parameters:

addr0 - start of address range (included);

addr1 - end of address range (not included);

type - type of names to delete (NM_ANYNAME to delete all names in the range).

See also: Insertname, Quickinsertname

Findlabelbyname

Searches for name of types NM_LABEL, NM_EXPORT, NM_IMPORT, NM_LIBRARY or NM_CONST in the specified range. If name is found, copies its address to *addr and returns type of label, otherwise returns NM_NONAME. Attention, this function is *very* slow, it searches name table sequentially!

int Findlabelbyname(char *name,ulong *addr,ulong addr0,ulong addr1);

Parameters:

name - pointer to output buffer of length at least <u>TEXTLEN</u> characters;

addr - pointer to variable that receives address of found name;

addr0 - start of address range (included);

addr1 - end of address range (not included).

See also: <u>Findname</u>, <u>Findlabel</u>, <u>Findimportbyname</u>

Findimportbyname

Searches for name of type NM_IMPORT in the specified range. If name is found, returns its address, otherwise returns 0. If name contains no module prefix, routine searches for import name with any module prefix. Attention, this function is very slow, it searches name table sequentially!

ulong Findimportbyname(char *name,ulong addr0,ulong addr1);

Parameters:

name - pointer to output buffer of length at least <u>TEXTLEN</u> characters;

- addr0 start of address range (included);
- addr1 end of address range (not included).

See also: Findname, Findlabel, Findlabelbyname

Findsymbolicname

Checks that there is a symbolic name associated with address. Returns 0 if there is no symbolic name. Returns 1 if name exists buf fname is NULL. Extracts name to fname and returns its size otherwise.

int Findsymbolicname(ulong addr,char *fname);

Parameters:

addr - address;

fname - pointer to output buffer of length at least <u>TEXTLEN</u> characters that receives found name.

See also: <u>Findname</u>, <u>Findlabel</u>, <u>Findlabelbyname</u>

Disassembly functions

Disasm is the most important OllyDbg function, and one of the most complicated. In version 1.06, its C code together with declarations, service subroutines and tables is 4291 lines (210 K bytes) long! Almost every part of OllyDbg calls <u>Disasm</u>, directly or indirectly.

Disasm requires that you supply binary code of the command to disassemble. <u>Readcommand</u> allows you to easily read command from the memory of debugged process.

Two other disassembly functions, <u>Disassembleforward</u> and <u>Disassembleback</u>, allow walking through the binary code, command by command. Note that 80x86 commands have variable length. <u>Disassembleback</u> use heuristical methods to separate commands and in some (astoundingly rare!) cases may return invalid answer. To avoid risks of invaling backward walking, use analysis data.

Functions <u>Issuspicious</u> and <u>Isfilling</u> can determine whether command is potentially invalid or equivalent to NOP.

ulong <u>Disasm</u>(char *src,ulong srcsize,ulong srcip,char *srcdec,<u>t_disasm</u> *disasm,int disasmmode,ulong threadid);

ulong <u>Readcommand(ulong ip,char *cmd);</u>

ulong Disassembleback(char *block,ulong base,ulong size,ulong ip,int n,int usedec);

ulong <u>Disassembleforward</u>(char *block,ulong base,ulong size,ulong ip,int n,int usedec);

ulong Followcall(ulong addr);

```
int <u>Issuspicious</u>(char *cmd,ulong size,ulong ip,ulong threadid,t_reg *preg,char
*s);
```

int <u>Isfilling</u>(ulong offset,char *data,ulong size,ulong align);

int Isprefix(int c);

t_disasm

<u>Disasm</u> uses this structure to report disassembly results. Which fields of the structure are filled depends on the disassembling mode:

DISASM_SIZE	Only error is valid
DISASM_DATA	Only members of t_disasm marked with asterisk (*) are valid
DISASM_TRACE	Only members marked with asterisk (*) and minus (-) are valid
DISASM_FILE	Complete disassembly, but <u>Disasm</u> assumes that registers are undefined and does not decode symbolic names. Members marked with minus (-) are invalid
DISASM_CODE	Complete disassembly, but <u>Disasm</u> assumes that registers are undefined. Members marked with minus (-) are invalid
DISASM_ALL	Complete disassembly. Members marked with minus (-) are invalid

typedef struct t_disasm { // Results of disassembling

ulong ip; // (*) Instrucion pointer

char dump[TEXTLEN]; // Hexadecimal dump of the command

char result[TEXTLEN]; // Disassembled command

char comment[<u>TEXTLEN</u>]; // Brief comment

char opinfo[3][<u>TEXTLEN</u>]; // Comments to command's operands

int cmdtype; // (*) One of C_xxx

int memtype; // (*) Type of addressed variable in memory

int nprefix; // (*) Number of prefixes

int indexed; // Address contains register(s) ulong jmpconst; // (*) Constant jump address ulong jmptable; // (*) Possible address of switch table ulong adrconst; // (*) Constant part of address ulong immconst; // (*) Immediate constant int zeroconst; // (*) Whether contains zero constant int fixupoffset; // (*) Possible offset of 32-bit fixups int fixupsize; // (*) Possible total size of fixups or 0 ulong jmpaddr; // Destination of jump/call/return int condition; // 0xFF:unconditional, 0:false, 1:true int error; // (*) Error while disassembling command int warnings; // (*) Combination of DAW_xxx int optype[3]; // Type of operand (extended set DEC_xxx) int opsize[3]; // Size of operand, bytes int opgood[3]; // Whether address and data valid ulong opaddr[3]; // Address if memory, index if register ulong opdata[3]; // Actual value (only integer operands) t_operand op[3]; // Full description of operand ulong regdata[8]; // Registers after command is executed int regstatus[8]; // Status of registers, one of RST_xxx ulong addrdata; // Traced memory address

int addr
status; // Status of addr
data, one of RST_xxx

ulong regstack[NREGSTACK]; // Stack tracing buffer

int rststatus[NREGSTACK]; // Status of stack items

int nregstack; // Number of items in stack trace buffer

ulong reserved[29]; // Reserved for plugin compatibility

} t_disasm;

Members:

ip - address of the disassembled command;

dump - ASCII string, formatted hexadecimal dump of the command;

result - ASCII string, disassembled command itself;

comment - ASCII string, brief comment that applies to the whole command;

opinfo - array of ASCII strings, comments to individual operands (explicit or implicit, like ESP, EBP and ECX in MOVSB);

cmdtype - type of the disassembled command, one of C_xxx possibly ORed with C_RARE to indicate that command is seldom in ordinary Win32 applications. Commands of type C_MMX additionally contain size of MMX data in the 3 least significant bits (0 means 8-byte operands). Non-MMX commands may have C_EXPL bit set which means that some memory operand has size which is not conform with standard 80x86 rules;

memtype - type of memory operand, one of DEC_xxx, or DEC_UNKNOWN if operand is non-standard or command does not access memory;

nprefix - number of prefixes that this command contains;

indexed - if memory address contains index register, set to scale, otherwise 0;

jmpconst - address of jump destination if this address is a constant, and 0 otherwise;

jmptable - if indirect jump can be interpreted as switch, base address of switch table and 0 otherwise;

adrconst - constant part of memory address;

immconst - immediate constant or 0 if command contains no immediate constant. The only command that contains two immediate constants is ENTER. <u>Disasm</u> ignores second constant which is anyway 0 in most cases;

zeroconst - nonzero if command contains immediate zero constant;

fixupoffset - possible start of 32-bit fixup within the command, or 0 if command can't contain fixups;

fixupsize - possible total size of fixups (0, 4 or 8). If command contains both immediate constant and immediate address, they are always adjacent on 80x86 processors;

jmpaddr - destination of jump, call or return. If jump address contains undefined register, jmpaddr is 0;

condition - whether condition in command is met: 0 - condition is false, 1 - true, -1 - command is unconditional or EFL is undefined;

error - <u>Disasm</u> was unable to disassemble command (for example, command does not exist or crosses end of memory block), one of DAE_xxx;

warnings - command is suspicious or meaningless (for example, far jump or MOV EAX,EAX preceded with segment prefix), combination of DAW_xxx bits;

optype - array of operand types, DEC_xxx or DECR_xxx;

opsize - array of operand sizes in bytes;

opgood - array of flags indicating opaddr and opdata are valid;

opaddr - array containing memory addresses of memory operands and register indexes for register operands. Valid only if corresponding opgood is set;

opdata - array of actual operand's values (integer operands only), valid only if

corresponding opgood is set;

op - full descriptions of operands.

Register tracing is still relatively raw and is not described.

Disasm

Disassembles command, determines its size and decodes operands. Returns size of the command. Disasm functionality depends on the selected mode and global disassembling/analysis options. See description of <u>t_disasm</u> for more details:

Mode	Actions
DISASM_SIZE	Fastest mode, only calculates command size
DISASM_DATA	Extracts most important data, no textual information
DISASM_TRACE	Extracts most important data and traces contents of integer registers, no textual information
DISASM_FILE	Disassembles command in assumption that registers are undefined and symbolic names are invalid. Usually used to disassemble contents of file
DISASM_CODE	Disassembles command assuming that registers are undefined
DISASM_ALL	Complete and relatively slow disassembly

ulong Disasm(char *src,ulong srcsize,ulong srcip,char *srcdec,<u>t_disasm</u> *disasm,int disasmmode,ulong threadid);

Parameters:

src - pointer to binary command that must be disassembled;

srcsize - size of src. Length of 80x86 commands is limited to <u>MAXCMDSIZE</u> bytes;

srcip - address of the command;

srcdec - pointer to decoding data produced by Analyzer or NULL if decoding data is absent. You must supply srcdec if you want to decode switch tables, constants and strings;

disasm - pointer to <u>t_disasm</u> structure that receives results of disassembling;

disasmmode - disassembly mode, one of DISASM_xxx. See description of <u>t_disasm</u> and table above;

threadid - identifier of thread containing registers, or NULL if registers are undefined.

See also: <u>Readmemory</u>, <u>Finddecode</u>, <u>t_disasm</u>, <u>MAXCMDSIZE</u>

Disassembleback

Calculates address of assembler instruction which is n instructions (maximally 127) back from instruction at specified address. Returns address of found instruction. In case of error, it may be less than n instructions apart.

80x86 commands have variable length. Disassembleback use heuristical methods to separate commands and in some (astoundingly rare!) cases may return invalid answer. To avoid risks of invaling backward walking, or correctly walk through constants and strings, use results of code analysis.

ulong Disassembleback(char *block,ulong base,ulong size,ulong ip,int n,int usedec);

Parameters:

block - pointer to copy of code. If block is NULL, Disassembleback assumes memory of debugged process and if necessary reads it;

base - address of first byte of code block;

size - size of code block;

ip - address of current instruction;

n - number of instructions to walk back;

usedec - flag indicating whether Disassembleback should try to use decoding data.

See also: Disassembleforward, Followcall, Findmemory, Readmemory

Disassembleforward

Calculates address of assembler instruction which is n instructions forward from instruction at specified address. If copy of code is not supplied, Disassembleforward guarantees correct results up to n=127 (typically 300). Returns address of found instruction. In case of error, it may be less than n instructions apart.

If you want to correctly walk through constants and strings, use results of code analysis.

ulong Disassembleforward(char *block,ulong base,ulong size,ulong ip,int n,int usedec);

Parameters:

block - pointer to copy of code. If block is NULL, Disassembleforward assumes memory of debugged process and if necessary reads it;

base - address of first byte of code block;

size - size of code block;

ip - address of current instruction;

n - number of instructions to walk forward;

usedec - flag indicating whether Disassembleforward should try to use decoding data.

See also: Disassembleback, Followcall, Findmemory, Readmemory

Followcall

Follows sequence of jumps (direct or indirect) and Win95 thunks that starts at specified address. Stops if:

-next command is neither jump nor thunk, or

-next command is exported entry in difefrent module, or

-length of sequence exceeds 10 jumps.

Returns address of final destination, or 0 on error. Parameter addr is usually the destination of CALL command, hence the name. As any access to the debuggee's memory takes significant time, this function may be slow.

ulong Followcall(ulong addr);

Parameters:

addr - address of first command in jump chain.

See also: Disassembleforward, Disassembleback, Disasm

Issuspicious

Checks whether command is somehow suspicious. Returns -1 on error, 0 if command is not suspicious and 1 if command is suspicious. Use only with program in memory, do not apply to file! Command is considered suspicious when:

·this command is erroneous or unknown, or

·it is potentially invalid according to active analysis options, or

·it sets single-step trap, or

·it accesses memory operand in unused part of stack (i.e. addr>ESP), or

·it is command CLI, or

•memory operand contains INT3 breakpoint set by OllyDbg.

int Issuspicious(char *cmd,ulong size,ulong ip,ulong threadid,<u>t_reg</u> *preg,char *comment);

Parameters:

cmd - pointer to the binary command code;

size - size of cmd in bytes;

ip - address of the command in the memory of debugged process;

threadid - identifier of the thread in which context this command will be executed;

preg - pointer to <u>registers</u> at the moment of execution;

comment - buffer, at least <u>TEXTLEN</u> bytes long, that receives explanation why this command is suspicious, or NULL.

See also: Disasm, Isfilling, Isprefix, Readcommand

Isfilling

Function checks whether command which binary code starts at data[offset] is a valid filling command (usually some kind of NOP) used to align code to a specified border. Returns length of command if this is recognized as filling and 0 otherwise. Checks include:

·NOP

·INT3

·XCHG RA,RA

•MOV RA,RA

·LEA RA,[RA] (with or without SIB byte)

·LEA RA,[RA+0000000]

This list is far from completeness but includes commands most frequently used as filling by actual compilers.

int Isfilling(ulong offset,char *data,ulong size,ulong align);

Parameters:

offset - offset of binary command in data;

data - buffer containing copy of executable code;

size - size of valid code in data (if size<offset+*size of tested command*, function returns 0);

align - expected code alignment, must be either power of 2 (1,2,4,8...) or 0 that means no alignment.

See also: Disasm, Issuspicious, Isprefix, Readcommand

Isprefix

Very quick and straightforward function, returns 1 if byte c is a 80x86 command prefix (**ES:**, **CS:**, **SS:**, **DS:**, **FS:**, **GS:**, **DATASIZE**, **ADDRSIZE**, **LOCK**, **REPNE**, **REP**) and 0 otherwise. Attention, it doesn't distinguish the cases when byte is part of the SSE/SSE2 command!

int Isprefix(int c);

Parameters:

c - byte to verify.

See also: <u>Issuspicious</u>, <u>Isfilling</u>

Readcommand

Reads command from the memory of debugged process and restored breakpoints. Returns length of the read code (at most <u>MAXCMDSIZE</u> bytes) or 0 if memory can't be read.

Note: Any access to the memory in different process is extremely timeexpensive. As in many cases different parts of OllyDbg access same command several times, Readcommand maintains small 1-command cache significantly improves the wholesave productivity of OllyDbg. If you need to access several compactly placed commands, <u>Readmemory</u> is usually much faster.

ulong Readcommand(ulong ip,char *cmd);

Parameters:

ip - address of the command in the memory space of debugged process. If ip is 0, function invalidates cache and returns 0;

cmd - buffer of length at least <u>MAXCMDSIZE</u> bytes that receives command.

See also: Disasm, Readmemory

Assembly functions

int <u>Assemble(char *cmd,ulong ip,t_asmmodel</u> *model,int attempt,int constsize,char *errtext);

int <u>Checkcondition(int code,ulong flags);</u>

Assemble

Function Assemble, as expected, converts command in ASCII form to binary 32bit code. It shares command table with <u>Disasm</u>, so if some command can be disassembled, it can be assembled back too, with one exception: Assemble doesn't support 16-bit addresses. Some commands have more than one encoding. By calling Assemble with parameter attempt=0,1... and constsize=0,1,2,3 one can get alternative variants and then select the shortest possible form (this is how OllyDbg implements assembling). However, only one address form is generated in each case ([EAX*2] but not [EAX+EAX]; [EBX+EAX] but not [EAX+EBX]; [EAX] will not use SIB byte; no DS: prefix and so on).

Assemble compiles imprecise commands (where, for example, R32 replaces any general-purpose 32-bit register). This allows to generate imprecise search patterns, where mask contains zeros at the position occupied in code by register). Returns number of bytes in assembled code or non-positive number in case of detected error or when variant selected by combination of attempt and constsize doesn't exist. This number is the negative position of error in the input command.

int Assemble(char *cmd,ulong ip,<u>t_asmmodel</u> *model,int attempt,int constsize,char *errtext);

Parameters:

cmd - pointer to zero-terminated ASCII command;

ip - address of the generated binary code in memory;

model - pointer to structure that receives machine code and mask;

attempt - index of alternative verison of the command. Call Assemble with attempt=0,1,2... to obtain all possible versions of the command. Stop this sequence when Assemble reports error;

constsize - requested size of address constant and immediate data. Call Assemble with constsize=0,1,2,3 to obtain all possible variants of the version selected by attempt;

errtext - pointer to text buffer of length at least <u>TEXTLEN</u> that receives description of detected error.

See also: Disasm

Checkcondition

Checks whether 80x86 flags meet condition set in the command. Returns 1 if condition is met and 0 if not.

int Checkcondition(int code,ulong flags);

Parameters:

code - first byte of conditional command;

flags - contents of register EFL.

Watch and expression functions

For some obscure reasons, watches in OllyDbg are 1-based. That means that to access the first available watch, you must set index in watch functions to 1. Internally, OllyDbg keeps watch expressions as names of type NM_WATCH, where first watch has address 1, next - address 2 and so on. Access to watch expressions using <u>name functions</u> is not recommended, direct deletion or insertion of new watches will bring watch window out of synchronization. Instead, use functions listed below.

int Insertwatch(int indexone, char *text);

```
int Deletewatch(int indexone);
```

```
int Getwatch(int indexone, char *text);
```

int Expression(t_result *result,char *expression,int a,int b, char *data,ulong
database,ulong datasize,ulong threadid);

Insertwatch

Inserts new watch before the watch with specified 1-based index and updates watch window. Returns number of watches after new watch is inserted, or -1 on error.

int Insertwatch(int indexone,char *text);

Parameters:

indexone - 1-based index of existing watch. If this index exceeds total number of existing watches, new watch will be added to the end of the watch table;

text - new watch expression to insert.

See also: Deletewatch, Getwatch

Deletewatch

Deletes watch with specified 1-based index and updates watch window. Returns number of remaining watches, or -1 on error.

int Deletewatch(int indexone);

Parameters:

indexone - 1-based index of existing watch.

See also: Insertwatch, Getwatch

Getwatch

Gets current expression of watch with given 1-based index. Returns length of expression or 0 in case of error.

int Getwatch(int indexone,char *text);

Parameters:

indexone - 1-based index of existing watch to retrieve;

text - buffer of length at least <u>TEXTLEN</u> bytes that receives watch expression.

See also: Insertwatch, Deletewatch

Expression

Expression calculates value and, if available, address of arithmetical expression. Expression can include constants, registers, memory addresses and to some limited extent symbolic names, all standard arithmetical operations, parentheses and two parameters %A and %B. You can find both intuitive and formal descriptions of allowed expressions in file ollydbg.hlp. On success, Expression fills in structure <u>t_result</u> and returns length of valid expression. On error (result->type==DEC_UNKNOWN) it returns position of error in expression string and error message in result->value.

Notice that starting from version 1.08, Expression() doesn't report error "Extra characters on line". Unrecognized symbols remain unprocessed.

int Expression(<u>t_result</u> *result,char *expression,int a,int b, char *data,ulong database,ulong datasize,ulong threadid);

Parameters:

result - pointer to structure <u>t_result</u> that receives results of evaluation;

expression - input string containing expression to evaluate;

a - value of parameter %A;

b - value of parameter %B;

data - optional pointer to the copy of memory of debugged process. If data is not NULL and expression accesses variable in memory in range from database to database+datasize, Expression takes contents of memory from data, otherwise it reads memory of debugged process. This spares time, especially if you estimates multiple expressions.

database - address of data in memory space of debugged process;

datasize - size of data;

threadid - identifier of thread whose registers will be used in evaluation of expression. If threadid is 0 and expression includes register, Expression reports

erorr.

See also: <u>Checkcondition</u>, <u>t_result</u>

t_result

Type of structure that contains result of expression evaluation.

typedef struct t_result { // Result of expression's evaluation

int type; // Type of expression, DEC(R)_xxx

int dtype; // Type of data, DEC_xxx

union {

char data[10]; // Binary form of expression's value

ulong u; // Value as unsigned integer

long l; // Value as signed integer

long double f; }; // Value as 80-bit float

union {

char value[TEXTLEN]; // ASCII form of expression's value

wchar_t wvalue[<u>TEXTLEN</u>/2]; };// UNICODE form of expression's value

ulong lvaddr; // Address or index of lvalue or NULL

} t_result;

Members:

type - exact type of expression, one of DEC_xxx or DECR_xxx possibly ORed with DEC_SIGNED if result should be interpreted as signed number. type is DEC_UNKNOWN if expression is invalid. Expression is lvalue (can be assigned to) if either type is DEC_xxx and lvaddr is not 0, or if type is one of DECR_xxx. All possible types are listed in the table below:

type & Meaning DECR_TYPEMASK

DEC_UNKNOWN	Error in expression
DEC_BYTE	Byte
DEC_WORD	Short integer
DEC_DWORD	Long integer
DEC_FLOAT4	32-bit float
DEC_FWORD	48-bit descriptor or long pointer
DEC_FLOAT8	64-bit double
DEC_QWORD	Quadword
DEC_FLOAT10	80-bit long double
DEC_STRING	Zero-terminated ASCII string
DEC_UNICODE	Zero-terminated UNICODE string
DECR_BYTE	Byte register
DECR_WORD	Short integer register
DECR_DWORD	Long integer register
DECR_QWORD	MMX register
DECR_FLOAT10	Floating-point register
DECR_SEG	Segment register

dtype - simplified type of data, possibly ORed with DEC_SIGNED, describes value stored in t_result.data. If bit DEC_SIGNED is set, result must be interpreted as signed, otherwise as unsigned:

dtype	Interpretation of t_result.data
DEC_UNKNOWN	Error in expression or result doesn't fit into data
DEC_DWORD	32-bit unsigned integer in t_result.u
DEC_DWORD DEC_SIGNED	32-bit signed integer stored in t_result.l
DEC_QWORD	64-bit integer in data[07]
DEC_FLOAT10	80-bit long double stored in t_result.f

data, u, l, f - result of expression if this can be represented as integer or float.

Which field to select depends on dtype;

value - result of expression of type DEC_STRING (truncated to <u>TEXTLEN</u> characters) or error message if type is DEC_UNKNOWN;

wvalue - result of expression of type DEC_UNICODE (truncated to <u>TEXTLEN</u>/2 characters);

lvaddr - address of expression if type is one of DEC_xxx, or index of register if type is DECR_xxx.

See also: <u>Expression</u>

Thread functions

OllyDbg keeps list of active thread in a sorted data consisting of elements of type t_thread. You can receive pointer to table of threads by calling Plugingetvalue(VAL_THREADS) and casting result to (t_table *). If you know thread's identifier, Findthread will return pointer to thread descriptor. Plugingetvalue(VAL_MAINTHREADID) gives identifier of main thread of debugged process.

OllyDbg functions use thread identifiers, but some Windows functions require handles. Following code converts identifier to handle:

t_thread *pthread;

```
HANDLE hthread;
```

```
pthread=Findthread(threadid);
```

```
if (pthread!=NULL)
```

```
hthread=pthread->handle;
```

else

```
hthread=NULL;
```

```
Note that after application started and before OllyDbg received CREATE_PROCESS_DEBUG_EVENT event, thread's handle is unknown.
```

```
t_thread* Findthread(ulong threadid);
```

```
int Decodethreadname(char *s,ulong threadid,int mode);
```

```
ulong Getcputhreadid(void);
```

```
HWND <u>Createthreadwindow(void);</u>
```

t_thread

Type of thread descriptor. typedef struct t_thread { // Information about active threads ulong threadid; // Thread identifier ulong dummy; // Always 1 ulong type; // Service information, TY_xxx HANDLE thread; // Thread handle ulong datablock; // Per-thread data block ulong entry; // Thread entry point ulong stacktop; // Working variable of Listmemory() ulong stackbottom; // Working variable of Listmemory() CONTEXT context; // Actual context of the thread t reg reg; // Actual contents of registers int regvalid; // Whether reg is valid t reg oldreg; // Previous contents of registers int oldregvalid; // Whether oldreg is valid int suspendcount; // Suspension count (may be negative) long usertime; // Time in user mode, 1/10th ms, or -1 long systime; // Time in system mode, 1/10th ms, or -1 ulong reserved[16]; // Reserved for future compatibility

} t_thread;

Members:

threadid - thread identifier;

dummy - size of thread in space of thread identifiers, must be 1. See <u>Sorted data</u> <u>functions</u> for explanation;

type - type of thread, combination of bits TY_xxx. If bit TY_MAIN is set, this is the main thread;

thread - thread handle. After application started and before OllyDbg received CREATE_PROCESS_DEBUG_EVENT event, thread's handle is unavailable;

datablock - base address of per-thread data block;

entry - address of thread entry point;

context - actual context of the thread. Do not modify context directly, or you risk to crash debugged application!

reg - excerpt from context that contains CPU registers sorted in a natural way. Valid only when regvalid is non-zero. If you need to modify register, stop application if necessary, check that regvalid is non-zero, apply your changes and set reg.modified to 1. Do not change single step flag or debugging register DR6;

regvalid - flag indicating that reg contains actual contents of thread's registers;

oldreg - previous contents of registers, don't modify. If reg.modifiedbyuser is 0, this is a copy of registers on a previous step, otherwise copy of original registers;

oldregvalid - flag indicating that contents of oldreg is valid;

suspendcount - number of times this thread was suspended by OllyDbg. May be negative in case when thread was suspended by user or program and resumed by OllyDbg. Do not modify directly!

usertime - time the thread spent in user mode, in 100-microsecond units, or -1 if unavailable;

systime - time the thread spent in system mode, in 100-microsecond units, or -1 if unavailable;

reserved - reserved for future use exclusively by OllyDbg.

See also: <u>Findthread</u>, <u>Plugingetvalue</u>

Findthread

Given thread's identifier, returns pointer to descriptor of specified thread, or NULL if thread does not exist.

t_thread* Findthread(ulong threadid);

Parameters:

threadid - identifier (not handle!) of the requested thread.

See also: <u>Getcputhreadid</u>, <u>t_thread</u>

Decodethreadname

Decodes name of thread with specified thread identifier to ASCII string, like "Main thread" or "thread 12345678". Returns length of name or 0 on error.

int Decodethreadname(char *s,ulong threadid,int mode);

Parameters:

s - pointer to buffer of length at least <u>TEXTLEN</u> bytes that receives decoded name;

threadid - thread identifier;

mode - combination of bits ADC_xxx that tell how to decode name of thread:

ADC_VALID	decode name of thread only if threadid is a valid thread identifier
ADC_SYMBOL	decode name of thread only if it has symbolic name
ADC_UPPERCASE	force first character of name to be in uppercase
ADC_WIDEFORM	include word "thread" into decoded name

Getcputhreadid

Returns identifier of thread that is currently selected in CPU window.

ulong Getcputhreadid(void);

Memory functions

OllyDbg keeps list of memory blocks allocated by debugged application in a table of sorted data consisting of elements of type <u>t_memory</u>. You can receive pointer to memory table by calling <u>Plugingetvalue</u>(VAL_MEMORY) and casting result to (<u>t_table</u> *).

t_memory* Findmemory(ulong addr);

void <u>Havecopyofmemory(char *copy,ulong base,ulong size);</u>

ulong <u>Readmemory</u>(void *buf,ulong addr,ulong size,int mode);

ulong <u>Writememory</u>(void *buf,ulong addr,ulong size,int mode);

int Listmemory(void);

t_memory

Type of memory descriptor, do not modify directly! typedef struct t_memory { // Memory block descriptor ulong base; // Base address of memory block ulong size; // Size of block ulong type; // Service information, TY_xxx ulong owner; // Address of owner of the memory ulong initaccess; // Initial read/write access ulong access; // Actual status and read/write access ulong threadid; // Block belongs to this thread or 0 char sect[SHORTLEN]; // Name of module section char *copy; // Copy used in CPU window or NULL ulong reserved[8]; // Reserved for plugin compatibility } t_memory;

Members:

base - base address of memory block in the memory space of debugged process;

size - size of memory block;

type - memory characteristics, combination of bits TY_xxx:

TY_CODE	Memory block contains image of code section
TY_DATA	Contains image of data section
TY_IMPDATA	Includes import data

TY_EXPDATA	Includes export data	
TY_RSRC	Contains resources	
TY_RELOC	Includes relocation data	
TY_STACK	Contains stack of thread with identifier threadid	
TY_THREAD	Contains data block of thread with identifier threadid	
TY_HEADER	Contains COFF header	
TY_DEFHEAP	Contains default heap	
TY_HEAP	Contains non-default heap	
TY_SFX	Contains self-extractor	
TY_GUARDED NT only: guarded memory block		

owner - address of memory block that owns this block;

initaccess - type of allowed memory access when block was allocated, one of PAGE_xxx (see description of Windows function VirtualQueryEx for details);

access - actual type of allowed memory access, one of PAGE_xxx

threadid - if memory contains stack of thread data block, identifier of owning thread, otherwise undefined;

sect - name of section (not necessarily null-terinated!) if block is an image of section in executable file, otherwise empty string;

copy - if memory block was backuped in CPU window, pointer to backup copy, or NULL otherwise;

reserved - reserved for future use exclusively by OllyDbg.

See also: <u>Findmemory</u>

Findmemory

Given address of memory, returns pointer to descriptor of memory block that this address belongs to, or NULL if there is no allocated memory.

t_memory* Findmemory(ulong addr);

Parameters:

addr - address of memory in the memory space of debugged application.

See also: <u>t_memory</u>

Havecopyofmemory

Optimizes access to memory of debugged process. Function <u>Readmemory</u> is slow. If you expect multiple reads from the same block, read requested piece of memory to some internal buffer and report it to OllyDbg. All subsequent calls to <u>Readmemory</u> will, whenever possible, use this copy. Don't forget to call Havecopyofmemory(NULL,0,0) when you no longer need this copy, or OllyDbg will crash! Note that <u>Writememory</u> will not update this copy.

void Havecopyofmemory(char *copy,ulong base,ulong size);

Parameters:

copy - pointer to copy of memory of debugged process;

base - base address of memory;

size - size of memory.

See also: <u>Readmemory</u>

Readmemory

Reads memory of debugged process optionally removing INT3 breakpoints. You can read memory "on the fly": if necessary, Readmemory temporally pauses debugged application and enables read access. Returns size of memory actually read. Currently, this is either size or 0 if memory cannot be read at once.

Important note: Any access to the memory of debugged application is timeconsuming. To optimize access, consider use of <u>Havecopyofmemory</u>.

ulong Readmemory(void *buf,ulong addr,ulong size,int mode);

Parameters:

buf - pointer to buffer of size at least size that receives copy of memory;

addr - address of memory in the memory space of debugged application;

size - size of requested memory block;

mode - mode of operation, combination of following bits:

MM_RESTORE Restore INT3 breakpoints MM_SILENT On error, don't display error message box

Note that header declares MM_RESILENT as a combination of (MM_RESTORE|MM_SILENT).

See also: <u>Writememory</u>, <u>Havecopyofmemory</u>

Writememory

Modifies memory of debugged process, optionally removing INT3 breakpoints, broadcasting memory changes and removing analysis data. Returns size of actually modified memory. Currently, this is either size or 0 if memory cannot be written at once.

ulong Writememory(void *buf,ulong addr,ulong size,int mode);

Parameters:

buf - pointer to buffer with new contents of memory;

addr - address of memory in the memory space of debugged application;

size - size of new contents;

mode - mode of operation, combination of following bits:

MM_RESTORERemove INT3 breakpoints in the modified
area and broadcast memory changesMM_DELANALWipe off analysis in the modified areaMM_SILENTOn error, don't display error message box

See also: <u>Readmemory</u>

Listmemory

Function actualizes list of memory blocks and (in case if Windows 95) list of heaps allocated by Debuggee. If memory and/or heap windows are open, also updates windows. Returns 0 if tables are actualized and -1 if some or all of entries may be invalid.

As this operation is time-consuming, OllyDbg usually updates memory tables only if application is paused. If plugin accesses memory tables "on the fly", it may need to call this function. Note that reading or writing to the memory does not require actualization of memory tables.

int Listmemory(void);

Module functions

Module is an executable file (ususlly EXE or DLL) loaded into memory. OllyDbg keeps list of loaded modules in a table of sorted data consisting of elements of type <u>t_module</u>. You can receive pointer to table of modules by calling <u>Plugingetvalue</u>(VAL_MODULES) and casting result to (<u>t_table</u> *).

t_module *Findmodule(ulong addr);

t_fixup *<u>Findfixup(t_module</u> *pmod,ulong addr);

char *<u>Finddecode(ulong addr,ulong *psize);</u>

ulong <u>Findfileoffset(t_module</u> *pmod,ulong addr);

int <u>Analysecode(t_module</u> *pmod);

t_module

Type of module descriptor. This is a very sensitive structure, do not modify directly!

typedef struct t_module { // Executable module descriptor

ulong base; // Base address of module

ulong size; // Size occupied by module

ulong type; // Service information, TY_xxx

ulong codebase; // Base address of module code block

ulong codesize; // Size of module code block

ulong resbase; // Base address of resources

ulong ressize; // Size of resources

t_stringtable *stringtable; // Pointers to string resources or NULL

int nstringtable; // Actual number of used stringtable

int maxstringtable; // Actual number of allocated stringtable

ulong entry; // Address of <ModuleEntryPoint> or NULL

ulong database; // Base address of module data block

ulong idatatable; // Base address of import data table

ulong idatabase; // Base address of import data block

ulong edatatable; // Base address of export data table

ulong edatasize; // Size of export data table

ulong reloctable; // Base address of relocation table

ulong relocsize; // Size of relocation table char name[SHORTLEN]; // Short name of the module char path[MAXPATH]; // Full name of the module int nsect; // Number of sections in the module IMAGE SECTION HEADER *sect; // Copy of section headers from file ulong headersize; // Total size of headers in executable ulong fixupbase; // Base of image in executable file int nfixup; // Number of fixups in executable t_fixup *fixup; // Extracted fixups or NULL char *codedec; // Decoded code features or NULL ulong codecrc; // Code CRC for actual decoding char *hittrace; // Hit tracing data or NULL char *hittracecopy; // Copy of INT3-substituted code char *datadec; // Decoded data features or NULL t table namelist; // List of module names t symvar *symvar; // Descriptions of symbolic variables int nsymvar; // Actual number of elements in symvar int maxsymvar; // Maximal number of elements in symvar char *globaltypes; // Global types from debug info ulong mainentry; // Address of WinMain() etc. in dbg data ulong realsfxentry; // Entry of packed code or NULL

int updatenamelist; // Request to update namelist ulong origcodesize; // Original size of module code block ulong sfxbase; // Base of memory block with SFX ulong sfxsize; // Size of memory block with SFX int issystemdll; // Whether system DLL int processed; // 0: not processed, 1: good, -1: bad int dbghelpsym; // 1: symbols loaded by dbghelp.dll char version[NVERS]; // Version of executable file t_jdest *jddata; // Recognized jumps within the module int njddata; // Number of recognized jumps ulong reserved[15]; // Reserved for plugin compatibility } t_module;

Members (members that intended stricly for internal use are not explained):

base - base address of module in the memory space of debugged process;

size - total size occupied by module, not necessarily contiguous memory;

type - service information, combination of bits TY_xxx;

codebase - base address of executable code, as stays in COFF header. In some cases, OllyDbg may correct definitely invalid code base;

codesize - size of executable code, as stays in COFF header. In some cases, OllyDbg may correct definitely invalid code size;

resbase - base address of resources;

ressize - size of resources;

entry - address of module's entry point, as stays in COFF header;

database - base address of module's data block. OllyDbg uses heuristics to locate data;

idatatable - base address of import data table, as stays in COFF header;

idatabase - base address of import data block, as stays in COFF header;

edatatable - base address of export data table, as stays in COFF header;

edatasize - size of export data table, as stays in COFF header;

reloctable - base address of relocation table, as stays in COFF header;

relocsize - size of relocation table, as stays in COFF header;

name - short name of the module, not necessarily NULL-terminated;

path - full name of executable file;

nsect - number of sections in the module;

sect - pointer to copy of section headers from the COFF header;

headersize - total size of headers in executable file;

fixupbase - base of image in executable file;

nfixup - number of fixups in executable file;

fixup - pointer to list of extracted fixups or NULL;

mainentry - address of WinMain or DllEntryPoint from debugging data or 0;

realsfxentry - real entry of unpacked SFX code or 0;

updatenamelist - request to update namelist;

issystemdll - 1 if module is system DLL (i.e. DLL residing in Windows' system directory) and 0 otherwise;

dbghelpsym - 1 if debugging information in one of Microsoft formats is available and 0 otherwise;

version - zero-terminated ASCII string containing version of executable file, NVERS-1 bytes long;

reserved - reserved for future use exclusively by OllyDbg.

See also: <u>Findmodule</u>, <u>Findfileoffset</u>

Findmodule

Given address of memory in debugged application, returns pointer to module descriptor that this address belongs to, or NULL if address is outside any module.

t_module* Findmodule(ulong addr);

Parameters:

addr - address of memory in the memory space of debugged application.

See also: <u>Findfixup</u>, <u>Finddecode</u>, <u>Findfileoffset</u>, <u>t_module</u>

Findfixup

If supplied address belongs to some module, function checks whether there are fixups including or exceeding this address and returns pointer to first such fixup. Otherwise, it returns NULL. Fixups are sorted in ascending order and terminated by element (0,0), so calling procedure may use returned pointer to walk through all subsequent fixups.

t_fixup *Findfixup(t_module *pmod,ulong addr);

Parameters:

pmod - optional pointer to module descriptor. If pmod is NULL, Findfixup looks for module descriptor by itself;

addr - address in memory space of debugged application where search for fixups will start.

See also: <u>Findmodule</u>, <u>Finddecode</u>, <u>Findfileoffset</u>, <u>t_module</u>

Analysecode

Analyzes executable code of specified module. Among other tasks, analysis includes:

- ·Recognition of commands and embedded data;
- •Recognition of 1- and 2-stage switches;
- •Recognition of procedures and loops;
- ·Decoding of arguments of known functions;
- ·Prediction of contents of registers;
- ·Forming of call tree.

One very important assumption: code is valid and is not counterfeit: knowing how this analysis works, one may write a program that will be analyzed totally incorrectly. Function is highly heuristical, so never assume that results are 100% reliable. Returns 0 on success and -1 on error.

int Analysecode(t_module *pmod);

Parameters:

pmod - pointer to module descriptor.

Finddecode

Searches for decoding data that starts on specified address. On success, sets *psize to size of located data and returns pointer to decoding information. If there is no decoding information, sets *psize to 0 and returns NULL. For each byte of analysed code, corresponding byte of decoding data contains combination of type, procedure and analysis fields:

Type field, use DEC_TYPEMASK to extract it from decoding data:

DEC_UNKNOWN	Unknown type
DEC_BYTE	Byte
DEC_WORD	First byte of 16-bit integer
DEC_NEXTDATA	Subsequent byte of data
DEC_DWORD	First byte of 32-bit integer
DEC_FLOAT4	First byte of 32-bit float
DEC_FWORD	First byte of descriptor or long pointer
DEC_FLOAT8	First byte of 64-bit double
DEC_QWORD	First byte of 64-bit integer
DEC_FLOAT10	First byte of 80-bit long double
DEC_TBYTE	First byte of 10-byte BCD integer
DEC_STRING	First byte of ASCII string
DEC_UNICODE	First byte of UNICODE string
DEC_3DNOW	First byte of 3DNow! operand
DEC_SSE	First byte of SSE operand
DEC_BYTESW	Byte which is a second-level switch index
DEC_NEXTCODE	Subsequent byte of command
DEC_COMMAND	First byte of command
DEC_JMPDEST	First byte of command that is jump destination
DEC_CALLDEST	First byte of command that is call (and maybe jump) destination

Procedure field, use DEC_PROCMASK to extract it from decoding data:

DEC_PROCStart of procedureDEC_PBODYBody of procedureDEC_PENDEnd of procedure

Bit DEC_CHECKED, if set, reports that byte was analyzed.

char *Finddecode(ulong addr,ulong *psize);

Parameters:

addr - address of the first byte in the memory space of debugged process for which decoding information is requested;

psize - pointer to variable that will receive size of found decoding data or NULL.

See also: Findmodule, Findfixup, Findfileoffset

Findfileoffset

Converts address belonging to some module into offset in executable file. Returns offset or 0 if offset cannot be calculated (for example, address belongs to the gap between two sections).

ulong Findfileoffset(<u>t_module</u> *pmod,ulong addr);

Parameters:

mod - optional pointer to module descriptor. If pmod is NULL, Findfileoffset looks for module descriptor by itself;

addr - address in memory space of debugged application where search for fixups will start.

See also: <u>Findmodule</u>, <u>Findfixup</u>, <u>Finddecode</u>, <u>t_module</u>

Data conversion functions

ulong <u>Compress</u>(char *bufin,ulong nbufin,char *bufout,ulong nbufout); ulong <u>Decompress</u>(char *bufin,ulong nbufin,char *bufout,ulong nbufout); ulong <u>Getoriginaldatasize</u>(char *bufin,ulong nbufin);

Compress

Compresses binary data. This function uses patent-free form of Lempel-Ziv compression algorithm. Returns length of compressed data or 0 if some error was detected during compression. First longword in the output buffer is the identifier of compressed data and second is the length of original data.

ulong Compress(char *bufin,ulong nbufin,char *bufout,ulong nbufout);

Parameters:

bufin - pointer to uncompressed data;

nbufin - size of uncompressed data;

bufout - pointer to buffer that will receive compressed data;

nbufout - size of bufout.

See also: <u>Decompress</u>

Decompress

Unpacks data compressed by <u>Compress</u>. Returns length of unpacked data or 0 if some error was detected during decompression.

ulong Decompress(char *bufin,ulong nbufin,char *bufout,ulong nbufout);

Parameters:

bufin - pointer to compressed data;

nbufin - size of compressed data;

- bufout pointer to buffer that will receive unpacked data;
- nbufout size of bufout.

See also: <u>Compress</u>, <u>Getoriginaldatasize</u>

Getoriginaldatasize

For the data compressed by <u>Compress</u>, returns size of the original data. Returns 0 on error.

ulong Getoriginaldatasize(char *bufin,ulong nbufin);

Parameters:

bufin - pointer to compressed data;

nbufin - size of compressed data;

See also: <u>Decompress</u>

Plugin functions

int <u>Registerpluginclass(char *classname,char *iconname,HINSTANCE</u> dllinst,WNDPROC classproc);

- void <u>Unregisterpluginclass(char *classname);</u>
- int <u>Pluginwriteinttoini(HINSTANCE dllinst, char *key, int value);</u>
- int <u>Pluginwritestringtoini(HINSTANCE dllinst, char *key, char *s);</u>
- int <u>Pluginreadintfromini(HINSTANCE dllinst, char *key, int def);</u>
- int <u>Pluginreadstringfromini(HINSTANCE dllinst,char *key,char *s,char *def);</u>
- int <u>Pluginsaverecord(ulong tag,ulong size,void *data);</u>
- int <u>Plugingetvalue(int type);</u>
- t_status Getstatus(void);

Registerpluginclass

Generates unique class name and registers new class of plugin windows. If iconname is NULL, uses standard plugin icon (letter 'P'). On success, returns 0 and fills classname (at least 32 bytes long) with unique class name. If registration failed, returns -1. Windows belonging to registered class has 8 longwords of extra memory, plugin is free to use longwords 2..7 (offsets 8..28 in calls to GetWindowLong and SetWindowLong). ODBG_Plugininit is the best place to call this function.

int Registerpluginclass(char *classname,char *iconname,HINSTANCE dllinst,WNDPROC classproc);

Parameters:

classname - pointer to buffer of length at least 32 characters that will receive unique class name;

iconname - name of icon resource in plugin DLL;

dllinst - plugin's instance;

classproc - pointer to window procedure of new class.

See also: <u>Unregisterpluginclass</u>

Unregisterpluginclass

Unregisters window class previously registered by <u>Registerpluginclass</u>. Call this function for each registered class from <u>ODBG_Plugindestroy</u>.

void Unregisterpluginclass(char *classname);

Parameters:

classname - class name returned by call to <u>Registerpluginclass</u>.

See also: <u>Registerpluginclass</u>

Pluginwriteinttoini

Stores an integer associated with a key in the plugin's personal section of the ollydbg.ini. Returns 1 on success and 0 on error.

int Pluginwriteinttoini(HINSTANCE dllinst,char *key,int value);

Parameters:

dllinst - plugin's instance;

key - name of the key to be associated with an integer;

value - integer to be written to ollydbg.ini.

See also: <u>Pluginreadintfromini</u>, <u>Pluginwritestringtoini</u>, <u>Pluginreadstringfromini</u>

Pluginreadintfromini

Reads integer associated with a key from the plugin's personal section of the ollydbg.ini. On success, returns integer from the initializations file. On error, returns specified default value.

int Pluginreadintfromini(HINSTANCE dllinst,char *key,int def);

Parameters:

dllinst - plugin's instance;

key - name of the key associated with an integer;

def - default value.

See also: <u>Pluginwriteinttoini</u>, <u>Pluginwritestringtoini</u>, <u>Pluginreadstringfromini</u>

Pluginwritestringtoini

Stores ASCII string associated with a key in the plugin's personal section of the ollydbg.ini. Returns 1 on success and 0 on error.

int Pluginwritestringtoini(HINSTANCE dllinst,char *key,char *s);

Parameters:

dllinst - plugin's instance;

key - name of the key to be associated with a string;

s - string to be stored in ollydbg.ini.

See also: <u>Pluginreadstringfromini</u>, <u>Pluginwriteinttoini</u>, <u>Pluginreadintfromini</u>

Pluginreadstringfromini

Reads string associated with a key from the plugin's personal section of the ollydbg.ini. On success, returns string from the initializations file. On error, returns specified default string.

int Pluginreadstringfromini(HINSTANCE dllinst,char *key,char *s,char *def);

Parameters:

dllinst - plugin's instance;

- key name of the key associated with the string;
- s pointer to buffer that receives string;

def - pointer to a null-terminated default string.

See also: <u>Pluginwritestringtoini</u>, <u>Pluginwriteinttoini</u>, <u>Pluginreadintfromini</u>,

Pluginsaverecord

Writes single record to .udd file. Returns 1 on success and 0 on error. Call this function only from <u>ODBG_Pluginsaveudd</u>, any other call will fail.

int Pluginsaverecord(ulong tag,ulong size,void *data);

Parameters:

- tag unique plugin-specific tag;
- size size of data to be written to .udd file, maximally USERLEN;
- data pointer to data of specified size to be written to .udd file.

See also: <u>ODBG_Pluginsaveudd</u>, <u>ODBG_Pluginuddrecord</u>

Plugingetvalue

Retrieves various OllyDbg settings and variables.

int Plugingetvalue(int type);

Parameters:

type - setting or variable to retrieve:

type	Cast to	Explanation
VAL_HINST	(HINST)	Current OllyDbg instance
VAL_HWMAIN	(HWND)	Handle of the main OllyDbg window
VAL_HWCLIENT	(HWND)	Handle of the MDI client window
VAL_NCOLORS		Number of common colors
VAL_COLORS	(COLORREF *)	RGB values of common colors
VAL_BRUSHES	(HBRUSH *)	Handles of common color brushes
VAL_PENS	(PEN *)	Handles of common color pens
VAL_NFONTS		Number of common fonts
VAL_FONTS	(HFONT *)	Handles of common fonts
VAL_FONTNAMES	(char **)	Internal font names
VAL_FONTWIDTHS	(int *)	Average widths of common fonts
VAL_FONTHEIGHTS	(int *)	Average heigths of common fonts
VAL_NFIXFONTS		Actual number of fixed- pitch fonts
VAL_DEFFONT		Index of default font
VAL_NSCHEMES		Number of color schemes
VAL_SCHEMES	(t_scheme *)	Colour schemes

VAL_DEFSCHEME		Index of default colour scheme
VAL_DEFHSCROLL		Default horizontal scroll
- VAL_RESTOREWINDOWPOS		Restore window positions from .ini
VAL_HPROCESS	(HANDLE)	Handle of debugged process
VAL_PROCESSID		Process ID of debugged process
VAL_HMAINTHREAD	(HANDLE)	Handle of main thread of debugged process
VAL_MAINTHREADID		Thread ID of main thread of debugged process
VAL_MAINBASE		Base of main module in the debugged process
VAL_PROCESSNAME	(char *)	Name of the debugged process
VAL_EXEFILENAME	(char *)	Name of the main debugged file
VAL_CURRENTDIR	(char *)	Current directory for debugged process
VAL_SYSTEMDIR	(char *)	Windows system directory
VAL_DECODEANYIP		Decode registers dependless on EIP
VAL_PASCALSTRINGS		Decode Pascal-style string constants
VAL_ONLYASCII		Only printable ASCII chars in dump
VAL_DIACRITICALS		Allow diacritical symbols in strings
VAL_GLOBALSEARCH		Search from the beginning of block
VAL_ALIGNEDSEARCH	[Search aligned to item's size
VAL_SEARCHMARGIN		Floating search allows error margin

VAL_KEEPSELSIZE		Keep size of hex edit selection
VAL_MMXDISPLAY		MMX display mode in dialog (0:hex, 1:signed, 2:unsigned MMX)
VAL_WINDOWFONT		Use calling window's font in dialog
VAL_TABSTOPS		Distance between tab stops
VAL_MODULES	(<u>t_table</u> *)	Table of modules (.EXE and .DLL)
VAL_MEMORY	(<u>t_table</u> *)	Table of allocated memory blocks
VAL_THREADS	(<u>t_table</u> *)	Table of active threads
VAL_BREAKPOINTS	(<u>t_table</u> *)	Table of active breakpoints
VAL_REFERENCES	(<u>t_table</u> *)	Table with found references
VAL_SOURCELIST	(<u>t_table</u> *)	Table of source files
VAL_WATCHES	(<u>t_table</u> *)	Table of watches
VAL_CPUFEATURES		CPU feature bits as returned by CPUID
VAL_TRACEFILE	(FILE *)	Handle of run trace log file
VAL_ALIGNDIALOGS		Align dialogs
VAL_CPUDASM	(<u>t_dump</u> *)	Dump descriptor of CPU Disassembler pane
VAL_CPUDDUMP	(<u>t_dump</u> *)	Dump descriptor of CPU Dump pane
VAL_CPUDSTACK	(<u>t_dump</u> *)	Dump descriptor of CPU Stack pane
VAL_APIHELP	(char *)	Name of selected API help file
VAL_HARDBP		Whether hardware breakpoints are enabled
VAL_PATCHES	(<u>t_table</u> *)	Table of patches
VAL_HINTS	(<u>t_sorted</u> *)	Sorted data with analysis hints

Getstatus

Returns current status of debugged process (one of STAT_xxx):

STAT_NONE	No process to debug
STAT_STOPPED	Process suspended
STAT_EVENT	Processing debug event, process temporarily paused
STAT_RUNNING	Process is running
STAT_FINISHED	Process terminated
STAT_CLOSING	TerminateProcess() called, waiting for confirmation

t_status Getstatus(void);

See also: <u>Plugingetvalue</u>

Source code support functions

Source debugging is still in development phase. I decided not to describe it in actual version of Plugin API.

CPU-specific functions

void <u>Setcpu</u>(ulong threadid,ulong asmaddr,ulong dumpaddr,ulong stackaddr,int mode);

void <u>Setdisasm(ulong asmaddr,ulong selsize,int mode);</u>

void <u>Redrawdisassembler(void);</u>

void Getdisassemblerrange(ulong *pbase,ulong *psize);

ulong Getcputhreadid(void);

Setcpu

Updates state of panes in CPU window. If necessary, creates or restores CPU window and moves it to top.

void Setcpu(ulong threadid,ulong asmaddr,ulong dumpaddr,ulong stackaddr,int mode);

Parameters:

threadid - identified of thread to display in CPU, or 0 if thread remains unchanged. If threadid id non-zero, parameters asmaddr and stackaddr are ignored and set to contents of EIP and ESP of the specified thread. If threadid is 0 and actual thread is invalid, Setcpu automatically reswitches to main thread;

asmaddr - address to display in Disassembler, or 0 if this address remains unchanged. Ignored if threadid is not 0;

dumpaddr - address to display in CPU Dump, or 0 if this address remains unchanged;

stackaddr - address to display in Stack, or 0 if this address remains unchanged. Ignored if threadid is not 0;

mode - combination of CPU_xxx flags that select update mode:

CPU_ASMHIST	Add change to Disassembler history
CPU_ASMCENTER	Position address in the middle of Disassembler window
CPU_ASMFOCUS	Move focus to Disassembler
CPU_DUMPHIST	Add change to Dump history (currently not available)
CPU_DUMPFIRST	Make dumpaddr the first byte in CPU Dump
CPU_DUMPFOCUS	Move focus to CPU Dump
CPU_REGAUTO	Automatically change Registers mode to FPU/MMX/3DNow!
CPU_RUNTRACE	Show run trace data at offset asmaddr

CPU_NOCREATE	Don't create CPU window if absent
CPU_REDRAW	Redraw CPU window immediately
CPU_NOFOCUS	Don't force focus to main window

See also: <u>Setdisasm</u>, <u>Redrawdisassembler</u>, <u>Getcputhreadid</u>

Setdisasm

Presets CPU Disassembler so that it displays code at address asmaddr. If selsize is greater than 1, selects selsize bytes, otherwise 1 assembler command. Then it creates CPU window (if absent), restores and moves window to the top.

void Setdisasm(ulong asmaddr,ulong selsize,int mode);

Parameters:

asmaddr - address to display in Disassembler, or 0 if this address remains unchanged. Ignored if threadid is not 0;

selsize - if greater than 1, size of selection in bytes, otherwise Setdisasm selects 1 command;

mode - combination of CPU_xxx flags that select update mode:

CPU_ASMHIST	Add change to Disassembler history
CPU_ASMCENTER	Position address in the middle of Disassembler window
CPU_ASMFOCUS	Move focus to Disassembler
CPU_REGAUTO	Automatically change Registers mode to FPU/MMX/3DNow!

See also: <u>Setcpu</u>, <u>Redrawdisassembler</u>, <u>Getcputhreadid</u>

Redrawdisassembler

Redraws Disassembler by calling UpdateWindow, so that all modifications are immediately visible.

void Redrawdisassembler(void);

See also: <u>Setcpu</u>

Getdisassemblerrange

Gets address range of memory block that is currently displayed in Disassembler window.

void Getdisassemblerrange(ulong *pbase,ulong *psize);

Parameters:

pbase - pointer to variable that receives base address of memory block in address space of debugged application;

psize - pointer to variable that receives size of memory block.

See also: <u>Getcputhreadid</u>

t_dump

Type of dump descriptor. typedef struct t_dump { // Current status of dump window t table table; // Treat dump window as custom table int dimmed; // Draw in lowcolor if nonzero ulong threadid; // Use decoding and registers if not 0 int dumptype; // Current dump type, DU_xxx+count+size SPECFUNC *specdump; // Decoder of DU_SPEC dump types int menutype; // Standard menus, MT_xxx int itemwidth; // Length of displayed item, characters int showstackframes; // Show stack frames in address dump int showstacklocals; // Show names of locals in stack int showsource; // Show source as comment in disassembler char filename[MAXPATH]; // Name of displayed or backup file ulong base; // Start of memory block or file ulong size; // Size of memory block or file ulong addr; // Address of first displayed byte ulong lastaddr; // Address of last displayed byte + 1 ulong sel0; // Address of first selected byte ulong sel1; // Last selected byte (not included!)

ulong startsel; // Start of last selection

int captured; // Mouse is captured by dump

ulong reladdr; // Addresses relative to this

char relname[SHORTLEN]; // Symbol for relative zero address base

char *filecopy; // Copy of the file or NULL

char *backup; // Old backup of memory/file or NULL

int runtraceoffset; // Offset back in run trace

ulong reserved[8]; // Reserved for the future extentions

} t_dump;

Members:

table - structure that describes dump window as a custom table;

threadid - if non-zero, window belongs to CPU and shuld use thread's registers when disassembling data;

dumptype - current dump type, combination of dump type (one of DU_xxx), number of items per line ((n<<8) & DU_COUNT) and size of single item (l & DU_SIZE). Additionally can be ORed with onbe of the following bits:

DU_ESCAPABLE	Dump window will close on ESC key
DU_BACKUP	Dump window displays backup data

For variable-length types the size is 1. See description of <u>Createdumpwindow</u> for a list of commonly used dump types;

base - base address of displayed memory in the memory size of debugged process, usually 0 for file dump;

size - size of displayed file or memory area;

addr - address or offset of the first displayed byte;

sel0 - address or offset of the first selected byte (included);

sel1 - address or offset of the last selected byte (not included);

filecopy - pointer to copy of displayed file, or NULL if this is memory dump;

backup - pointer to local backup of dump data, or NULL if backup is absent;

runtraceoffset - step back in run trace, or 0 if inactive.

See also: <u>Createdumpwindow</u>, <u>ODBG_Pluginuddrecord</u>, <u>ODBG_Pluginmenu</u>, <u>ODBG_Pluginaction</u>

t_window

Type of window descriptor - structure describing window or control created by debugged application.

typedef struct t_window { // Description of window

ulong hwnd; // Window's handle

ulong dummy; // Must be 1

ulong type; // Type of window, TY_xxx

ulong parenthw; // Handle of parent or 0

ulong winproc; // Address of WinProc or 0

ulong threadid; // ID of the owning thread

ulong exstyle; // Extended window style

ulong style; // Window style

ulong id; // Identifier or menu handle

ulong classproc; // Address of default (class) WinProc

int child; // Index of next child

int level; // Level in genealogy (0: topmost)

int sibling; // Index of next sibling

int byparent; // Index when sorted by parent

char title[TEXTLEN]; // Window's title

char classname[TEXTLEN]; // Class name

char tree[MAXNEST]; // For internal use by OllyDbg

} t_window;

Members:

hwnd - handle of window (control) created by debugged application, cast to HWND to use as a handle in calls to Windows API routines;

dummy - ust be 1 to obey the rules of <u>sorted data;</u>

type - type of window. The only important flag here is TY_NEW;

parenthw - handle of parent window or NULL. In some case this may be the handle of desktop (obtainable by call to *GetDesktopWindow*();

winproc - address of window procedure associated with window in memory context of debugged application. On NT-based systems, *GetWindowLong*(hwnd,GWL_WNDPROC) returns 0 and OllyDbg uses code injection to obtain this address;

threadid - identifier of thread that owns window;

exstyle - extended style of window, set of WS_EX_xxx and similar flags;

style - style of window, set of WS_xxx and similar flags;

id - control's identifier;

classproc - address of window's class procedure. If classproc differs from winproc, window is subclassed;

title - ASCII string with window's title or text;

classname - ASCII string with window's class name.

t_ref

Type of reference descriptor.

typedef struct t_ref { // Description of reference

ulong addr; // Address of reference

ulong size; // 1: single command, otherwise size

ulong type; // Type of reference, TY_xxx

ulong dest; // Destination of call

} t_ref;

Members:

addr - address of referencing command or data;

size - 1 if single command is referenced, or total size, bytes, of selected commands otherwise;

type - type of reference, combination of TY_xxx flags:

TY_REFERENCE Item is a real reference TY_ORIGIN Item is a search origin

dest - destination of intermodular call, 0 for any other reference.

Plugin callback functions

Plugin interface includes several callback functions. OllyDbg calls them to install or remove plugin and on important events, like selected menu item or pressed shortcut key. Only two callback are mandatory: <u>ODBG_Plugindata</u> and <u>ODBG_Plugininit</u>, all other are optional. Don't forget to export your callbacks!

int <u>ODBG_Plugindata(char *shortname);</u>

int <u>ODBG_Plugininit(int ollydbgversion,HWND hw,ulong *features);</u>

void ODBG_Pluginmainloop(DEBUG_EVENT *debugevent);

void <u>ODBG_Pluginsaveudd(t_module</u> *pmod,int ismainmodule);

int <u>ODBG_Pluginuddrecord(t_module</u> *pmod,int ismainmodule,ulong tag,ulong size,void *data);

int ODBG_Pluginmenu(int origin,char data[4096],void *item);

void ODBG_Pluginaction(int origin,int action,void *item);

int <u>ODBG_Pluginshortcut(int origin,int ctrl,int alt,int shift,int key,void *item);</u>

void ODBG_Pluginreset(void);

void ODBG_Pluginclose(void);

void ODBG_Plugindestroy(void);

int <u>ODBG_Paused(int reason, t_reg</u> *reg);

int <u>ODBG_Pausedex(int reason, int extdata, t_reg</u> *reg, DEBUG_EVENT *debugevent);

int <u>ODBG_Plugincmd(int reason,t_reg</u> *reg,char *cmd);

ODBG_Paused

Optional callback function. If present, OllyDbg will call it each time the debugged application is paused and after all internal processing is finished. Plugin may, for example, make some modifications and immediately continue execution by caling <u>Go</u>. In this case it may return 1, disabling time-consuming redrawing of windows. In any other case it must return 0.

Note that if plugin exports both ODBG_Paused and <u>ODBG_Pausedex</u>, only the second function will be called.

int ODBG_Paused(int reason, t_reg *reg);

Parameters:

reason - reason why application was paused:

PP_EVENT	Paused on debugging event
PP_PAUSE	Paused on user's request
PP_TERMINATED	Application terminated

reg - pointer to registers of thread that caused application to pause, may be NULL.

See also: <u>ODBG_Pausedex</u>

ODBG_Pausedex

Optional callback function. If present, OllyDbg will call it each time the debugged application is paused and after all internal processing is finished. Plugin may, for example, make some modifications and immediately continue execution by caling <u>Go</u>. In this case it may return 1, disabling time-consuming redrawing of windows. In any other case it must return 0.

Note that if plugin exports both ODBG_Pausedex and <u>ODBG_Paused</u>, the second function will not be called.

int ODBG_Pausedex(int reason, int extdata, t_reg *reg, DEBUG_EVENT *debugevent);

Parameters:

reason - reason why application was paused, use PP_MAIN to extract:

PP_EVENT	Paused on debugging event
PP_PAUSE	Paused on user's request
PP_TERMINATED	Application terminated

The reason may be ORed with one or several of the following clarifiers:

PP_BYPROGRAM	Debugging event caused by program
PP_INT3BREAK	INT3 breakpoint
PP_MEMBREAK	Memory breakpoint
PP_HWBREAK	Hardware breakpoint
PP_SINGLESTEP	Single-step trap
PP_EXCEPTION	Exception, like division by 0
PP_ACCESS	Access violation, like writing to NULL pointer
PP_GUARDED	Guarded page

extdata - reserved, currently always 0;

reg - pointer to registers of thread that caused application to pause, may be NULL;

debugevent - pointer to debug event that caused pause, or NULL if there was no event.

See also: <u>ODBG_Paused</u>

ODBG_Plugincmd

Optional callback function. If present, OllyDbg will call it each time the debugged application pauses on conditional logging breakpoint that specifies commands to be passed to plugins. Each command is passed to every plugin that exports ODBG_Plugincmd, so plugin must decide by itself whether it should execute command or not. For example, sample command line plugin accepts all commands that begin with a point. If plugin recognizes command, it must return 1 to stop OllyDbg from passing it to remaining plugins. Otherwise, it must return 0.

int ODBG_Plugincmd(int reason,<u>t_reg</u> *reg,char *cmd);

Parameters:

reason - reason why program was paused, currently always PP_EVENT;

reg - pointer to registers of thread that caused application to pause, may be NULL;

cmd - null-terminated command to plugin.

ODBG_Plugindata

Mandatory callback function that must be present in any valid OllyDbg plugin. It must fill in plugin name and return version of plugin interface (constant PLUGIN_VERSION). If function is absent, or version is not compatible, plugin will be not installed. Short name identifies plugin in OllyDbg. This name is limited to 31 alphanumerical characters or spaces followed by terminating null character. To keep life easy for users, name should be descriptive and correlate with the name of DLL.

int ODBG_Plugindata(char *shortname);

Parameters:

shortname - pointer to buffer of length at least 32 characters that receives name of plugin. This name may include spaces and punctuators but no special symbols.

ODBG_Plugininit

Mandatory callback function that must be present in any valid OllyDbg plugin. Here you can place all startup initializations and allocate resources. If startup was successfull, function must return 0. On error, it must free allocated resources and return -1, in this case plugin will be removed. Parameter ollydbgversion is the version of OllyDbg, use it to assure that OllyDbg is compatible with your plugin.

int ODBG_Plugininit(int ollydbgversion,HWND hw,ulong *features);

Parameters:

ollydbgversion - version of OllyDbg. Check that your plugin is compatible with this version. I will try to avoid incompatible changes in the future versions of OllyDbg;

hw - handle of main OllyDbg window, keep it if necessary;

features - reserved for future extentions.

See also: <u>ODBG_Pluginreset</u>, <u>ODBG_Pluginclose</u>, <u>ODBG_Plugindestroy</u>

ODBG_Pluginmainloop

Optional callback function. If present, OllyDbg will call it on each pass of main loop. Here you can do all your periodical tasks. Don't assume that calls are equidistant; they aren't. Do not export this function unnecessarily, as this may negatively influence the overall speed!

void ODBG_Pluginmainloop(DEBUG_EVENT *debugevent);

Parameters:

debugevent - pointer to debug event received by call to Windows API function WaitForDebugEvent, or NULL if there was no event.

ODBG_Pluginsaveudd

Optional callback function. If present, OllyDbg calls it when some module requests to save module- or application-related data to .udd file. To save data to .udd file, call <u>Pluginsaverecord</u> for each data item that must be saved. Global, application-oriented data must be saved in teh main .udd file; module-relevant data must be saved in module .udd files. Save all addresses relative to the base of module so that data will be restored correctly even when module is relocated.

void ODBG_Pluginsaveudd(t_module *pmod,int ismainmodule);

Parameters:

pmod - pointer to module descriptor;

ismainmodule - flag indicating whether this is main module of debugged application (.exe).

See also: <u>Pluginsaverecord</u>, <u>t_module</u>

ODBG_Pluginuddrecord

Optional callback function. If present, OllyDbg calls <u>ODBG_Pluginuddrecord</u> when it reads .udd file and encounters unrecognized record. If record belongs to plugin, it must process record and return 1, otherwise it must return 0 to pass record to other plugins. Note that module descriptor pointed to by pmod can be incomplete, i.e. does not necessarily contain information stored in processed .udd file, like decoding data or hit trace bufer.

int ODBG_Pluginuddrecord(t_module *pmod,int ismainmodule,ulong tag,ulong size,void *data);

Parameters:

pmod - pointer to module descriptor;

ismainmodule - flag indicating whether this is main module of debugged application (.exe);

tag - tag that identifies record;

size - size of data;

data - pointer to binary record data.

See also: <u>Pluginsaverecord</u>, <u>t_module</u>

ODBG_Pluginmenu

Optional callback function. If present, OllyDbg calls it to give plugin the possibility to add menu items either to main OllyDbg menu (origin=PM_MAIN) or to popup menu in one of standard OllyDbg windows. To add menu items, plugin must prepare string that describes menu structure and return 1, otherwise it must return 0. As a general OllyDbg rule, do not add inactive items to menu.

int ODBG_Pluginmenu(int origin,char data[4096],void *item);

Parameters:

origin - code of window that calls ODBG_Pluginmenu. OllyDbg supports following codes:

Cast item to	Who calls ODBG_Pluginmenu
item is always NULL	Main window
(<u>t_dump</u> *)	Any Dump window
(<u>t_module</u> *)	Modules window
(t_memory *)	Memory window
(<u>t_thread</u> *)	Threads window
(<u>t_bpoint</u> *)	Breakpoints window
(t_ref *)	References window
(int *)	Run trace window
(1-based index)	Watches window
(<u>t_window</u> *)	Windows window
(<u>t_dump</u> *)	CPU Disassembler
(<u>t_dump</u> *)	CPU Dump
(<u>t_dump</u> *)	CPU Stack
(<u>t_reg</u> *)	CPU Registers
	item is always NULL (t_dump *) (t_module *) (t_memory *) (t_thread *) (t_thread *) (t_thread *) (t_tref *) (t_ref *) (t_ref *) (1-based index) (t_window *) (t_dump *) (t_dump *) (t_dump *)

data - pointer to buffer 4K bytes long that receives description of menu structure.

Ordinary menu item consists of decimal identificator (0 to 63) followed by name. When user selects some menu item, Pluginaction receives identifier of this item. Duplicated identifiers are allowed. Use comma (,) to separate menu items. Vertical line (|) places horizontal dividing line in menu. To create submenu, add its name followed by contents of submenu enclosed into braces. OllyDbg automatically removes unnecessary or duplicated separators and empty submenus. To force horizontal dividing line, use # symbol. Some examples:

0 &Aaa,2 &Bbb 3 &Ccc ,,	Linear menu with 3 items: Aaa, Bbb and Ccc, relative IDs 0, 2 and 3, menu shortcuts A, B and C. Separator between second and third item, last separator and commas are ignored
#A{0Aaa,B{1Bbb 2Ccc}}	Unconditional separator, followed by popup menu A with two elements, second of them is popup B with two elements and separator inbetween

item - pointer either to selected element of sorted data displayed in window or, in case of dump windows, pointer to dump descriptor. Can be NULL. You may need this element to find out which menu items apply to selecetd item.

See also: **ODBG_Pluginaction**, Pluginaction, **Plugingetvalue**

ODBG_Pluginaction

Optional callback function. If present, OllyDbg calls it each time the user selected menu item added to menu by <u>ODBG_Pluginmenu</u>.

void ODBG_Pluginaction(int origin,int action,void *item);

Parameters:

origin - code of window that calls ODBG_Pluginaction. OllyDbg supports following codes:

Code	Cast item to	Who calls ODBG_Pluginmenu
PM_MAIN	item is always NULL	Main window
PM_DUMP	(<u>t_dump</u> *)	Any Dump window
PM_MODULES	(t_module *)	Modules window
PM_MEMORY	(t_memory *)	Memory window
PM_THREADS	(<u>t_thread</u> *)	Threads window
PM_BREAKPOINTS	(<u>t_bpoint</u> *)	Breakpoints window
PM_REFERENCES	(t_ref *)	References window
PM_RTRACE	(int *)	Run trace window
PM_WATCHES	(1-based index)	Watches window
PM_WINDOWS	(t_window *)	Windows window
PM_DISASM	(<u>t_dump</u> *)	CPU Disassembler
PM_CPUDUMP	(<u>t_dump</u> *)	CPU Dump
PM_CPUSTACK	(<u>t_dump</u> *)	CPU Stack
PM_CPUREGS	(<u>t_reg</u> *)	CPU Registers

action - identifier of menu item (0..63), as set by <u>ODBG_Pluginmenu</u>;

item - pointer either to selected element of sorted data displayed in window or, in case of dump windows, pointer to dump descriptor, or NULL. You may need this

element to carry out requested action.

See also: <u>ODBG_Pluginmenu</u>, Pluginaction, <u>Plugingetvalue</u>, <u>Custom messages</u>

ODBG_Pluginshortcut

Optional callback function. If present, OllyDbg calls it each time when user presses combination of keys that is not recognized by standard OllyDbg window. This function is usually called twice: first time with origin=PM_MAIN indicating global shortcut, and second time with origin identifier of window that has keyboard focus. Shortcuts are scarce resource and I will constantly add new to OllyDbg, so use this feature with care and always implement alternative possibilities.

int ODBG_Pluginshortcut(int origin,int ctrl,int alt,int shift,int key,void *item);

Parameters:

origin - code of window that calls ODBG_Pluginshortcut. OllyDbg supports following codes:

Code	Cast item to	Who calls ODBG_Pluginmenu
PM_MAIN	item is always NULL	Main window
PM_DUMP	(<u>t_dump</u> *)	Any Dump window
PM_MODULES	(t_module *)	Modules window
PM_MEMORY	(t_memory *)	Memory window
PM_THREADS	(<u>t_thread</u> *)	Threads window
PM_BREAKPOINTS	(<u>t_bpoint</u> *)	Breakpoints window
PM_REFERENCES	(t_ref *)	References window
PM_RTRACE	(int *)	Run trace window
PM_WATCHES	(1-based index)	Watches window
PM_WINDOWS	(t_window *)	Windows window
PM_DISASM	(<u>t_dump</u> *)	CPU Disassembler
PM_CPUDUMP	(<u>t_dump</u> *)	CPU Dump
PM_CPUSTACK	(<u>t_dump</u> *)	CPU Stack

PM_CPUREGS (t_reg *) CPU Registers

ctrl - state of Ctrl key: 0 - released, 1 - pressed;

alt - state of Alt key: 0 - released, 1 - pressed;

shift - state of Shift key: 0 - released, 1 - pressed;

key - code of pressed virtual key (VK_xxx). See "Virtual Key Codes" in Windows API help for a complete list of virtual key codes;

item - pointer either to selected element of sorted data displayed in window or, in case of dump windows, pointer to dump descriptor, or NULL. You may need this element to carry out requested action.

ODBG_Pluginreset

Optional callback function. If present, OllyDbg calls ODBG_Pluginreset when user opens new or restarts current application. Plugin should reset internal variables and data structures to initial state.

void ODBG_Pluginreset(void);

ODBG_Pluginclose

OllyDbg calls this optional function when user wants to terminate OllyDbg. All MDI windows created by plugin still exist. This is the best possibility to save plugin parameters to .ini file. Function must return 0 if it is safe to terminate OllyDbg. Any non-zero return will stop closing sequence. Do not misuse this possibility! Always inform user about the reasons why termination is not good and ask for his decision!

void ODBG_Pluginclose(void);

See also: <u>ODBG_Plugindestroy</u>, <u>Pluginwriteinttoini</u>, <u>Pluginwritestringtoini</u>

ODBG_Plugindestroy

OllyDbg calls this optional function once on exit. At this moment, all MDI windows created by plugin are already destroyed (received WM_DESTROY messages). Function must free all internally allocated resources, like window classes, files, memory and so on.

void ODBG_Plugindestroy(void);

Breakpoint functions

INT3 breakpoints are briefly explained here.

int <u>Manualbreakpoint(ulong addr,int key,int shiftkey,ulong nametype,int font);</u>

void Tempbreakpoint(ulong addr,int mode);

int <u>Setbreakpoint(ulong addr,ulong type,uchar cmd);</u>

int <u>Setbreakpointext(ulong addr,ulong type,uchar cmd,ulong passcount);</u>

ulong Getbreakpointtypecount(ulong addr,ulong *passcount);

int <u>Setmembreakpoint(int type,ulong addr,ulong size);</u>

Note that hardware breakpoints are not supported by Windows 95 and Windows 98. To assure that you can use functions listed below, call Plugingetvalue(VAL_HARDBP):

int <u>Sethardwarebreakpoint(ulong addr,int size,int type);</u>

int <u>Hardbreakpoints(int closeondelete);</u>

int Deletehardwarebreakpoint(int index);

int Deletehardwarebreakbyaddr(ulong addr);

Setbreakpoint

Simplified (old) version of <u>Setbreakpointext</u>, kept for compatibility reasons. Equivalent to call Setbreakpointext(addr,type,cmd,0).

int Setbreakpoint(ulong addr,ulong type,uchar cmd);

Parameters:

addr - address of breakpoint. If address points to data or in the middle of the command, OllyDbg will ask you for confirmation;

type - combination of bits TY_xxx that specify requested actions and type of breakpoint, see description of <u>Setbreakpointext</u>;

cmd - original command that will be saved to descriptor if bit TY_KEEPCODE is set. Otherwise, this parameter is ignored and command is read from the memory.

Setbreakpointext

Sets new INT3 breakpoint or changes type of existing breakpoint at specified address. Returns 0 on success and -1 on error (i.e. breakpoint was neither set nor restored). If bit TY_KEEPCOND in type is set, condition, explanation and expression associated with breakpoint (explained here) remain unchanged, otherwise they are removed. If bit TY_SETCOUNT is set or breakpoint is absent, sets specified pass count, otherwise pass count remains unchanged.

int Setbreakpointext(ulong addr,ulong type,uchar cmd,ulong passcount);

Parameters:

addr - address of breakpoint. If address points to data or in the middle of the command, OllyDbg will ask you for confirmation;

type - combination of bits TY_xxx that specify requested actions and type of breakpoint:

Flag	Meaning
TY_ACTIVE	Set permanent (user) breakpoint or restore disabled
TY_DISABLED	Temporarily deactivate permanent breakpoint. If TY_ACTIVE and TY_DISABLED are set simultaneously, TY_DISABLED is ignored
TY_ONESHOT	Set one-shot breakpoint that will be automatically removed when hit. Doesn't interfere with active breakpoint
TY_TEMP	Set temporary breakpoint that will be automatically removed when hit. Execution continues automatically. TY_TEMP does not interfere with active breakpoint
TY_STOPAN	Stop animation if breakpoint is hit
TY_KEEPCODE	Force original command (parameter cmd)
TY_SETCOUNT	Force pass count even if breakpoint already exists
	Leave associated names of types NM_BREAK,

TY_KEEPCOND NM_BREAKEXPR, NM_BREAKEXPL and NM_PLUGCMD unchanged. If this bit is not set, breakpoints of types TY_ACTIVE and TY_DISABLED clear these names

cmd - original command that will be saved to descriptor if bit TY_KEEPCODE is set. Otherwise, this parameter is ignored and command is read from the memory;

passcount - pass count, i.e. the number of times this breakpoint should be skipped. If breakpoint already exists and flag TY_SETCOUNT is not set, this parameter is ignored and pass count remains unchanged.

To set conditional breakpoint, consider use of <u>Manualbreakpoint</u>. If breakpoint must be set automatically (i.e. without user's interference), please do the following:

·If debugged program is still running, call <u>Suspendprocess</u> to make following operations atomic;

•Call <u>Setbreakpointext</u>(addr,TY_ACTIVE,0,passcount), thus setting INT3 breakpoint and related pass count. This is enough for ordinary (unconditional) breakpoint;

·If necessary, set condition by call to <u>Insertname</u>(addr,NM_BREAK,condition). This is enough for conditional breakpoint;

•To set conditional logging breakpoint, you must additionally prepare <u>control</u> <u>byte</u>, expression and explanation and set them calling <u>Insertname(NM_BREAKEXPR)</u> and <u>Insertname(NM_BREAKEXPL)</u>;

·If necessary, resume execution (<u>Go</u>).

See also: <u>Breakpoint functions</u>, <u>Manualbreakpoint</u>, <u>Setbreakpoint</u>, <u>Getbreakpointtypecount</u>.

How breakpoint works

OllyDbg supports many kinds of INT3 breakpoints: ordinary, conditional and conditional logging. Of course, internally this is the same breakpoint with different options activated. At the first glance, it looks overcomplicated and illogical; but it is really so. Version 2.0 should make breakpoints better, but now you must live with what you have.

Breakpoint consists of single-byte command INT3 that replaces first byte of the breakpointed command, descriptor of type <u>t_bpoint</u> in <u>table of active breakpoints</u> and several <u>names</u> associated with the same address that specify expressions and necessary actions:

Name type	Meaning
NM_BREAK	<u>Condition</u> associated with breakpoint. If condition is absent or invalid, OllyDbg assumes that it is true;
NM_BREAKEXPL	Explanation - any text that identifies breakpoint to user. Usually has no special meaning. Message breakpoints use special name " <winproc>";</winproc>
NM_BREAKEXPR	Expression that should be estimated and logged. First byte of expression contains flags (set of COND_xxx, explained below) that control behaviour of breakpoint;
NM_PLUGCMD	Commands that will be passed, one by one, to plugins if breakpoint is taken. Command are separated by CR, LF or CRLF.

Ordinary breakpoint (toggled if you press **F2**) has no associated names and zero <u>pass count</u>. Program pauses whenever this breakpoint is hit.

Conditional breakpoint (shortcut **Shift+F2**) has associated name of type NM_BREAK. If breakpoint is hit, OllyDbg estimates value of expression. If result is not 0, or expression is invalid, program pauses. Otherwise, OllyDbg

continues execution.

Conditional logging breakpoint (**Shift+F4**) has at least associated name of type NM_BREAKEXPR. First byte of this name is a set of flags COND_xxx that specify additional options. Strange settings of bits COND_NOBREAK and COND_BRKALWAYS are for backward compatibility with version 1.00. As you see, so deep compatibility is not always good:

Bit	Meaning	Equivalent in dialog
COND_NOBREAK	Don't pause execution if breakpoint is hit. Has higher priority than COND_BRKALWAYS	Pause program: Never
COND_BRKALWAYS	Always pause if breakpoint is hit. If both COND_NOBREAK and COND_BRKALWAYS are zero, pause on condition	Pause program: Always
COND_LOGTRUE	Estimate value of expression NM_BREAKEXPR and log it together with NM_BREAKEXPL if condition is true	Log value: On condition
COND_LOGALWAYS	Always log value of expression	Log value: Always
COND_ARGTRUE	Decode and log arguments of known function if expression is true	Log arguments: On condition
COND_ARGALWAYS	Always log arguments of known function	Log arguments: Always
COND_FILLING	Always set to assure that resulting byte is not 0	

Descriptor of breakpoint contains <u>pass count</u>. This feature is new to OllyDbg 1.10. If breakpoint is hit and conditions (or their absence) indicate that program should be paused, OllyDbg compares pass count with 0. If count is 0, program pauses. Otherwise, OllyDbg decrements counter and continues execution. Pass count does not restore automatically, that is, after it is decremented to zero, it

remains zero until user or plugin will set it again.

See also: <u>Breakpoint functions</u>, <u>Manualbreakpoint</u>, <u>Setbreakpoint</u>, <u>Setbreakpointext</u>, <u>Getbreakpointtypecount</u>.

Getbreakpointtypecount

Returns type (combination of bits <u>TY_xxx</u>) and associated pass count of INT3 breakpoint at specified address. If breakpoint doesn't exist, returns TY_INVALID.

ulong Getbreakpointtypecount(ulong addr,ulong *passcount);

Parameters:

addr - address of breakpoint;

passcount - pointer to variable that will receive pass count, can be NULL.

See also: <u>Breakpoint functions</u>, <u>How breakpoint works</u>, <u>Manualbreakpoint</u>, <u>Setbreakpoint</u>, <u>Setbreakpointext</u>.

t_bpoint

Type of INT3 breakpoint descriptor:

typedef struct t_bpoint { // Description of INT3 breakpoint

ulong addr; // Address of breakpoint

ulong dummy; // Always 1

ulong type; // Type of breakpoint, TY_xxx

char cmd; // Old value of command

ulong passcount; // Actual pass count

} t_bpoint;

Members (members that intended stricly for internal use are not explained):

addr - address of breakpoint;

dummy - length of breakpoint, must be 1;

type - type of breakpoint, combination of bits TY_xxx. Avoid direct modification. Please do not change flags that are not described here:

Flag	Meaning
TY_SET	Code INT3 is in memory. Never change!
TY_ACTIVE	Permanent (user) breakpoint
TY_DISABLED	Temporarily deactivated permanent breakpoint
TY_ONESHOT	One-shot breakpoint set by OllyDbg, automatically removed if breakpoint is hit
TY_TEMP	Temporary breakpoint, used internally by OllyDbg, for example to step over permanent breakpoint. Automatically removed when hit, execution continues

cmd - original command at specified address. If breakpoint is active, this command is replaced in memory by INT3;

passcount - counter that indicates how many times this breakpoint must be skipped. If OllyDbg decides that program should pause at breakpoint and passcount is not 0, it decrements passcount and continues execution. Note that this item is new to OllyDbg 1.10.

To get breakpoint descriptor, you may use the following code:

t_table *bptable;

t_bpoint *bpoint;

```
bptable=(t_table *)Plugingetvalue(VAL_BREAKPOINTS);
```

```
if (bptable!=NULL) {
```

bpoint=(t_bpoint *)Findsorteddata(&(bptable->data),addr);

```
if (bpoint!=NULL) {
```

```
..... any necessary actions .....
```

}

```
}
```

See also: <u>Breakpoint functions</u>, <u>Setbreakpoint</u>, <u>Setbreakpointext</u>, <u>Tempbreakpoint</u>

Manualbreakpoint

Facilitates manual INT3 breakpoint setting, either from menu or keyboard shortcut. Supports standard OllyDbg "look and feel". Returns 0 if some action took place and -1 otherwise. Following combinations are supported:

key	shiftkey	Action
VK_F2	0	Toggle unconditional breakpoint
VK_F2	Pressed (not 0)	Set conditional breakpoint
VK_F4	Pressed (not 0)	Set logging breakpoint

int Manualbreakpoint(ulong addr,int key,int shiftkey,ulong nametype,int font);

Parameters:

addr - memory address in the address space of debugged application where INT3 breakpoint must be set;

key - VK_F2 or VK_F4 (see above);

shiftkey - state of shift key (see above);

nametype - set to 0 when calling Manualbreakpoint from plugin;

font - index of predefined font to be used in invoked dialogs. If not sure, use FIXEDFONT.

Tempbreakpoint

Sets temporary or one-shot breakpoint on execution. If possible, sets hardware breakpoint, otherwise INT3. OllyDbg automatically removes temporary and one-shot breakpoints.

void Tempbreakpoint(ulong addr,int mode);

Parameters:

addr - code address where temporary breakpoint should be set;

mode - type of breakpoint to set:

Set one-shot breakpoint. OllyDbg automatically TY_ONESHOT|TY_KEEPCOND removes one-shot breakpoint when hit and pauses debugged application Same as above, additionally stops TY_ONESHOT|TY_KEEPCOND|TY_STOPAN any kind of trace or animation when hit Set temporary breakpoint. OllyDbg automatically removes TY_TEMP|TY_KEEPCOND temporary breakpoint when hit and immediately continues

Any other combination

execution Sets INT3 breakpoint of specified type

Setmembreakpoint

Modifies or removes memory breakpoint. OllyDbg supports only one memory breakpoint at a time. Returns 0 on success and -1 on error. Call Setmembreakpoint(0,0,0) to disable memory breakpoint.

int Setmembreakpoint(int type,ulong addr,ulong size);

Parameters:

type - type of memory breakpoint. Use either MEMBP_READ or MEMBP_READ|MEMBP_WRITE;

addr - start of memory breakpoint in the address space of debugged application;

size - size of memory breakpoint, bytes.

Sethardwarebreakpoint

Sets hardware breakpoint and activates it. 80x86 compatible processors support 4 hardware breakpoints. If all available slots are in use, function asks user to delete one of active breakpoints. Returns 0 on success and -1 on error or if user cancelled action. It is allowed to call Sethardwarebreakpoint "on the fly", i.e. when debugged application is running.

Note that hardware breakpoints are not supported by Windows 95 and Windows 98. To assure that you can use this function, call <u>Plugingetvalue</u>(VAL_HARDBP).

int Sethardwarebreakpoint(ulong addr,int size,int type);

Parameters:

addr - address of breakpoint;

size - size of memory covered by hardware breakpoint (1, 2 or 4 bytes). addr must be aligned on the corresponding boundary. This parameter must be 1 in case of breakpoint on execution;

type - type of hardware breakpoint:

HB_CODEActive on command executionHB_ACCESSActive on read/write accessHB_WRITEActive on write access

See also: <u>Hardbreakpoints</u>, <u>Deletehardwarebreakpoint</u>, <u>Deletehardwarebreakbyaddr</u>

Hardbreakpoints

Creates dialog enabling user to view, follow and delete existing hardware breakpoints. If closeondelete is 1, dialog closes after some breakpoint is deleted. Returns -1 on error or if user cancelled action and 0 otherwise.

Note that hardware breakpoints are not supported by Windows 95 and Windows 98. To assure that you can use this function, call <u>Plugingetvalue</u>(VAL_HARDBP).

int Hardbreakpoints(int closeondelete);

Parameters:

closeondelete - if 1, asks user to delete some existing breakpoint and closes dialog window after some hardware breakpoint is deleted.

See also: <u>Sethardwarebreakpoint</u>, <u>Deletehardwarebreakpoint</u>, <u>Deletehardwarebreakbyaddr</u>

Deletehardwarebreakpoint

80x86 processors support up to 4 hardware breakpoints. This function removes hardware breakpoint with specified index previously set by OllyDbg. Returns 0 on success and -1 on error. OllyDbg may use hardware breakpoints to bypass actual command, so use this function with care! Function <u>Deletehardwarebreakbyaddr</u> is easier to use.

Note that hardware breakpoints are not supported by Windows 95 and Windows 98. To assure that you can use this function, call <u>Plugingetvalue</u>(VAL_HARDBP).

int Deletehardwarebreakpoint(int index);

Parameters:

index - index of hardware breakpoint to delete (0..3).

See also: Sethardwarebreakpoint, Hardbreakpoints, Deletehardwarebreakbyaddr

Deletehardwarebreakbyaddr

Deletes hardware breakpoint by address. If there are several breakpoints embracing same addres, deletes all such breakpoints. Returns number of deleted breakpoints or 0 on error.

Note that hardware breakpoints are not supported by Windows 95 and Windows 98. To assure that you can use this function, call Plugingetvalue(VAL_HARDBP).

int Deletehardwarebreakbyaddr(ulong addr);

Parameters:

addr - address of hardware breakpoint. Every hardware breakpoint that covers this address will be removed. For example, if hardware breakpoint has address 0x00123450 and size 4, it covers address range from 0x00123450 to 0x00123453 inclusive.

See also: Sethardwarebreakpoint, Hardbreakpoints, Deletehardwarebreakpoint

Execution and stepping functions

Execution and stepping functions listed in this section check for rough errors but, when improperly used, may bring OllyDbg in unstable state. Please use them with care! For simple tasks, consider use of <u>Sendshortcut</u>.

int <u>OpenEXEfile(char *path,int dropped);</u>

int <u>Attachtoactiveprocess(int processid);</u>

int <u>Go</u>(ulong threadid,ulong tilladdr,int stepmode,int givechance,int backupregs);

void <u>Animate(int animation);</u>

int <u>Suspendprocess(int processevents);</u>

ulong <u>Runsinglethread(ulong threadid);</u>

void <u>Restoreallthreads(void);</u>

Go

Continues execution of the debugged program. Returns -1 if continuation is impossible and 0 on success. Improper use of this function may bring OllyDbg in unstable or undefined state. For simple tasks, consider use of <u>Sendshortcut</u>.

int Go(ulong threadid,ulong tilladdr,int stepmode,int givechance,int backupregs);

Parameters:

threadid - thread ID to continue. If threadid is 0, function assumes thread where last debugging event occured;

tilladdr - if stepmode is STEP_SKIP, function requests skipping of all commands up to tilladdr at once. Calling routine must guarantee that tilladdr is the first byte of some command and that sequence inbetween has no jumps/returns to outside. Otherwise, sets temporary breakpoint on tilladdr so that program will pause at this point (like "Run tol selection" in Disassembler).

stepmode - stepping mode, one of the following:

STEP_SAME	Same action as on previous call to Go
STEP_RUN	Run program
STEP_OVER	Step over (execute calls at once)
STEP_IN	Step in (enter subroutines)
STEP_SKIP	Skip sequence till specified address

givechance - if debugged application was paused on exception and this parameter is not 0, passes exception to exception handler installed by application;

backupregs - if not 0, updates old thread registers (element oldreg of structure <u>t_thread</u>). Disassembler uses backup to highlight modified registers.

See also: <u>OpenEXEfile</u>, <u>Animate</u>, <u>Suspendprocess</u>, <u>Runsinglethread</u>, <u>Restoreallthreads</u>

Animate

Sets animation mode and, if requested in debug options, sets higher priority to debugged process. Notice that this function doesn't start stepping or animation, you must explicitly call <u>Go</u> afterwards. Improper use of Animate may bring OllyDbg in unstable state. For simple tasks, consider use of <u>Sendshortcut</u>.

void Animate(int animation);

Parameters:

animation - animation mode:

ANIMATE_OFF	No animation
ANIMATE_IN	Animate into
ANIMATE_OVER	Animate over
ANIMATE_RET	Execute till RET
ANIMATE SKPRET	Execute till RET, then skip RET
	instruction
ANIMATE_USER	Execute till user code
ANIMATE_TRIN	Run trace in
ANIMATE_TROVER	Run trace over
ANIMATE_STOP	Gracefully stop animation

See also: <u>OpenEXEfile</u>, <u>Go</u>, <u>Suspendprocess</u>, <u>Runsinglethread</u>, <u>Restoreallthreads</u>

Suspendprocess

Suspends all threads of the process being debugged. It may happen (especially when logging breakpoints are set or hit trace is active) that threads will be suspended after some breakpoint is executed but corresponding debug event is not processed. If you want OllyDbg to process events before returning from Suspendprocess, call it with processevents=1. Returns 0 on success and -1 in case of any error. To resume execution, call <u>Go</u>. This function is slow on Win95-bases systems.

int Suspendprocess(int processevents);

Parameters:

processevents - process pending debugging events before return.

See also: <u>OpenEXEfile</u>, <u>Go</u>, <u>Animate</u>, <u>Runsinglethread</u>, <u>Restoreallthreads</u>

Runsinglethread

Suspends all threads except for specified, and resumes specified thread even if it was suspended. If threadid is 0 or invalid, suspends all threads. Returns thread ID of the thread that was the only one running, thread ID of the main thread if there were none/more than 1 active threads, and 0 on error. To reverse effect of this function, call <u>Restoreallthreads</u>. Improper use of this function may bring OllyDbg in unstable or undefined state.

ulong Runsinglethread(ulong threadid);

Parameters:

threadid - identifier (not handle!) of thread to run, or 0 to suspend all threads.

See also: OpenEXEfile, Go, Animate, Suspendprocess, Restoreallthreads

OpenEXEfile

Closes actuall process and starts new executable or link specified in path. Returns 0 if executable file is successfully started. Displays error message and returns -1 if file is not a 32-bit Portable Executable or OllyDbg was unable to create new process.

int OpenEXEfile(char *path,int dropped);

Parameters:

path - pointer to ASCII string with name of executable file (.exe) or Explorer link file (.lnk);

dropped - set to 1 if executable file was drag-and-dropped to OllyDbg or plugin, otherwise set it to 0. Currently, the only action of this flag is to clear command line.

See also: <u>Go</u>, <u>Animate</u>, <u>Suspendprocess</u>, <u>Runsinglethread</u>, <u>Restoreallthreads</u>

Restoreallthreads

Restores original thread states (as before the sequence of calls to <u>Runsinglethread</u>). Warns if all threads are suspended.

void Restoreallthreads(void);

See also: <u>OpenEXEfile</u>, <u>Go</u>, <u>Animate</u>, <u>Suspendprocess</u>, <u>Runsinglethread</u>

Trace and profiling functions

char *<u>Findhittrace(ulong addr,char **ptracecopy,ulong *psize);</u>

int Modifyhittrace(ulong addr0,ulong addr1,int mode);

int <u>Runtracesize(void);</u>

int Findprevruntraceip(ulong ip,int startback);

int Findnextruntraceip(ulong ip,int startback);

int <u>Startruntrace(t_reg *preg);</u>

```
void <u>Deleteruntrace(void);</u>
```

void Settracecondition(char *cond,int onsuspicious,ulong in0,ulong in1,ulong out0,ulong out1);

void <u>Settracecount(ulong count);</u>

int Getruntraceregisters(int nback,t_reg *preg,t_reg *pold,char *cmd,char
*comment);

int Getruntraceprofile(ulong addr,ulong size,ulong *profile);

HWND <u>Creatertracewindow</u>(void);

void <u>Scrollruntracewindow(int back);</u>

HWND <u>Createprofilewindow</u>(ulong base,ulong size);

Settracecount

Sets number of commands to trace. After specified number of commands is logged to trace buffer, trace pauses. Usually you may call this function after <u>Settracecondition</u>.

void Settracecount(ulong count);

Parameters:

count - number of commands to execute before run trace pauses.

See also: <u>Settracecondition</u>

Findhittrace

Looks whether hit trace information is available starting from specified address. Returns pointer to hit trace information corresponding to given address and optionally sets *ptracecopy to copy of original code and *psize to size of remaining data. Returns NULL and sets *psize to 0 if there is no decoding information. Hit trace information is an array of bytes that are the combination of bits TR_xxx.

char *Findhittrace(ulong addr,char **ptracecopy,ulong *psize);

Parameters:

addr - address of first byte of the code in the address space of debugged application;

ptracecopy - pointer to variable that receives pointer to statical copy of original code, may be NULL;

psize - pointer to variable that receives size of hit trace and copy data, may be NULL.

See also: Modifyhittrace, Runtracesize

Modifyhittrace

Function adds, resets, removes or restores specified range in the combined hit/run trace data buffer. This buffer contains flags specifying which actions should be undertaken when corresponding command is reached, don't mix it with the run trace log buffer that contains results of run trace. If necessary, buffer is created. Returns 0 on success (even partial) and -1 on error.

Warning: Setting hit trace or forced run trace on data may have disastrous effects on your program!

int Modifyhittrace(ulong addr0,ulong addr1,int mode);

Parameters:

addr0 - address of the first byte of the code range in the address space of debugged application;

addr1 - address of the last byte of the code range in the address space of debugged application (not included);

mode - action to perform, one of the following:

ATR_ADD	Hit trace specified range
ATR_ADDPROC	Hit trace only recognized procedures in the range
ATR_RESET	Mark range as not traced
ATR_REMOVE	Remove range and breakpoints
ATR_REMOVEALL	Destroy range and breakpoints
ATR_RESTORE	Restore breakpoints in memory
ATR_RTRADD	Hit trace range and force run trace
ATR_RTRJUMPS	Hit trace and run trace jumps only
ATR_RTRENTRY	Hit trace and run trace entries only
ATR_RTREMOVE	Remove trace from range
ATR_RTSKIP	Skip range from run trace

See also: Findhittrace, Runtracesize

Runtracesize

Returns number of records in run trace data, including record added during <u>initialization</u>, or 0 if run trace data is absent. This function is very fast.

int Runtracesize(void);

Findprevruntraceip

Searches for the previous (older) appearance of command with specified EIP in the run trace buffer, starting from the specified backward step (not included in search). Returns backward step or -1 if command is not in trace or if run trace is inactive.

int Findprevruntraceip(ulong ip,int startback);

Parameters:

ip - address of the command to search;

startback - backward step where the search starts. This step is not included in search. Use startback=0 to search for the youngest appearance.

See also: Findhittrace, Runtracesize, Findnextruntraceip, Getruntraceregisters

Findnextruntraceip

Searches for the next (younger) appearance of command with specified EIP in the run trace buffer, starting from the specified backward step (not included in search). Returns backward step or -1 if command is not in trace or if run trace is inactive.

int Findnextruntraceip(ulong ip,int startback);

ip - address of the command to search;

startback - backward step where the search starts. This step is not included in search.

See also: <u>Findhittrace</u>, <u>Runtracesize</u>, <u>Findprevruntraceip</u>, <u>Getruntraceregisters</u>

Getruntraceregisters

Extracts registers that are nback steps back in the run trace data (nback=0 means actual registers) and optionally registers on the previous step (so one can check for modifications). Optionally extracts original command and comment. Returns -1 of error, length of command if cmd!=NULL and original command is available and 0 if original command is absent. If record contains skipped sequence, returns 0 and sets cmd[0] to 0x01.

int Getruntraceregisters(int nback,<u>t_reg</u> *preg,t_reg *pold,char *cmd,char *comment);

Parameters:

nback - backward step in run trace buffer, 0 means actual step;

preg - pointer to <u>t_reg</u> structure that receives registers restored to the state *after* this command was executed;

pold - pointer to <u>t_reg</u> structure that receives registers restored to the state *before* this command was executed, can be NULL;

cmd - buffer at least <u>MAXCMDSIZE</u> bytes long that receives original command, or NULL. If record contains skipped sequence and cmd is not NULL, function sets cmd[0] to 0x01 and returns 0;

comment - buffer at least <u>TEXTLEN</u> bytes long that receives comment from the run trace buffer, can be NULL.

See also: Runtracesize, Findprevruntraceip, Findnextruntraceip

Getruntraceprofile

Calculates number of times that each address in range from addr to addr+size (not included) appears in the run trace data. Parameter profile points to array of size elements that receives profile data. Returns 0 on success or when run trace data is unavailable, and -1 on error. Function can be rather slow if run trace data is long.

int Getruntraceprofile(ulong addr,ulong size,ulong *profile);

Parameters:

addr - base address of the profiled code;

size - size of the profiled code;

profile - pointer to array of size doublewords that receives profile data.

See also: <u>Findhittrace</u>, <u>Runtracesize</u>, <u>Findprevruntraceip</u>, <u>Findnextruntraceip</u>, <u>Getruntraceregisters</u>

Scrollruntracewindow

Selects specified line and scrolls run trace window so that selection is visible. If option "Synchronize CPU and Run trace" is active, Disassembler also scrolls to this command.

void Scrollruntracewindow(int back);

Parameters:

back - backward step in run trace buffer, 0 means actual step.

See also: <u>Runtracesize</u>, <u>Findprevruntraceip</u>, <u>Findnextruntraceip</u>, <u>Getruntraceregisters</u>

Startruntrace

Reinitializes trace data and reallocates trace buffer. Previous trace is deleted. Returns 0 on success and -1 on error.

int Startruntrace(t_reg *preg);

Parameters:

preg - pointer to actual <u>registers</u> that will be used as the oldest record in the run trace buffer. Function fails if preg is NULL.

See also: <u>Runtracesize</u>, <u>Findprevruntraceip</u>, <u>Findnextruntraceip</u>, <u>Getruntraceregisters</u>, <u>Settracecondition</u>

Deleteruntrace

Closes run trace and destroys trace data.

void Deleteruntrace(void);

See also: <u>Startruntrace</u>, <u>Runtracesize</u>, <u>Findprevruntraceip</u>, <u>Findnextruntraceip</u>, <u>Getruntraceregisters</u>

Settracecondition

OllyDbg can pause run trace on a set of conditions. This function quickly sets pause on expression, on suspicious command and/or on EIP range and deactivates pause on command.

void Settracecondition(char *cond,int onsuspicious,ulong in0,ulong in1,ulong out0,ulong out1);

Parameters:

cond - pointer to character string containing expression. Run trace will pause if expression is invalid or estimates to non-zero value;

onsuspicious - activates (1) or deactivates (0) pause on suspicious command;

in0, in1 - 'in range' request. Run trace will pause if EIP is in this range (in1 not included). To disable pause on 'in range', set both in0 and in1 to 0;

out0, out1 - 'out of range' request. Run trace will pause if EIP is outside this range or equals to out1. To disable pause on 'out of range', set both out0 and out1 to 0.

See also: Startruntrace, Issuspicious

Createprofilewindow

Creates new or brings to top existing profile window and displays actual profile for the specified piece of code. Only one profile window may exist at a time. Returns handle of the window or NULL on error. Note that in order to actualize profile, this function attempts to allocate temporary buffer of size 4*size bytes, and will fail if you specify too large or non-contiguous code block.

HWND Createprofilewindow(ulong base,ulong size);

base - base address of the profiled code;

size - size of the profiled code.

See also: Startruntrace, Getruntraceprofile

t_reg

Structure that keeps the values of all relevant 80x86 registers. Note that length of this structure in version 1.10 is increased by 4 bytes. This may lead to incompatibilities with previous versions.

typedef struct t_reg { // Excerpt from context

int modified; // Some regs modified, update context

int modifiedbyuser; // Among modified, some modified by user

int singlestep; // Type of single step, SS_xxx

ulong r[8]; // EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI

ulong ip; // Instruction pointer (EIP)

ulong flags; // Flags

int top; // Index of top-of-stack

long double f[8]; // Float registers, f[top] - top of stack

uchar tag[8]; // Float tags (0x3 - empty register)

ulong fst; // FPU status word

ulong fcw; // FPU control word

ulong s[6]; // Segment registers ES,CS,SS,DS,FS,GS

ulong base[6]; // Segment bases

ulong limit[6]; // Segment limits

uchar big[6]; // Default size (0-16, 1-32 bit)

ulong dr6; // Debug register DR6

```
ulong threadid; // ID of thread that owns registers
ulong lasterror; // Last thread error or 0xFFFFFFFF
int ssevalid; // Whether SSE registers valid
int ssemodified; // Whether SSE registers modified
char ssereg[8][16]; // SSE registers
ulong mxcsr; // SSE control and status register
int selected; // Reports selected register to plugin
ulong drlin[4]; // Debug registers DR0..DR3
ulong dr7; // Debug register DR7
```

} t_reg;

Members:

modified - non-zero value indicates that some registers were modified and OllyDbg should update CONTEXT structure of the corresponding thread before continuing execution;

modifiedbyuser - among modified registers, some registers were modified by user;

singlestep - used internally by OllyDbg, do not modify directly!

r - 32-bit general-purpose registers EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI (in the listed order, use constants REG_xxx to access);

ip - 32-bit Instruction Pointer (EIP register);

flags - 32-bit EFLAGS register, do not modify single-step trap bit!

top - index of the register that is the top of the FPU stack;

f - 80-bit floating-point/MMX/3DNow! registers;

tag - two-bit tags associated with floating point registers;

fst - 16-bit FPU status word;

fcw - 16-bit FPU control word;

s - segment registers ES,CS,SS,DS,FS,GS (in the listed order, use constants SEG_xxx to access);

base - base addresses of segment descroptors;

limit - limits of segment descriptors;

big - default segment size (0 - 16-bit segment, seldom in flat mode; 1 - 32-bit segment);

dr6 - debug register dr6, please do not modify!

threadid - identifier of the thread that owns registers;

lasterror - last error in the thread as returned by call to GetlastError, or -1 (0xFFFFFFF) if exact value of the error is unknown;

ssevalid - non-zero if ssereg contain valid data;

ssereg - 16-byte SSE registers;

mxcsr - SSE control and status register;

selected - currently selected register, defined only if t_reg is passed to one of ODBG_Plugin... callback functions, otherwise undefined. AND this value with RS_GROUP to obtain the group of registers RS_xxx; to get index of register within the group, AND it with RS_INDEX. For example, code 0013 is a general-purpose register EBX (0013 & RS_GROUP = RS_INT, 0013 & RS_INDEX = REG_EBX);

drlin - debug registers dr0..dr3, please do not modify!

dr7 - debug register dr7, please do not modify!

Procedure functions

Group of functions that facilitate handling of procedures recognized by Analyzer.

ulong Findprocbegin(ulong addr);

ulong Findprocend(ulong addr);

ulong Findprevproc(ulong addr);

ulong Findnextproc(ulong addr);

int Getproclimits(ulong addr,ulong *start,ulong *end);

Findprocbegin

Returns start address of the procedure that encloses addr, or 0 on error, for example, when module is not analyzed or address points to no procedure.

ulong Findprocbegin(ulong addr);

Parameters:

addr - address of any command within the procedure.

See also: Findprocend, Findprevproc, Findnextproc, Getproclimits

Findprocend

Returns address of the last command of the procedure that encloses addr, or 0 on error, for example, when module is not analyzed or address points to no procedure.

ulong Findprocend(ulong addr);

Parameters:

addr - address of any command within the procedure.

See also: <u>Findprocbegin</u>, <u>Findprevproc</u>, <u>Findnextproc</u>, <u>Getproclimits</u>

Findprevproc

Returns start address of the procedure that precedes or encloses addr, or 0 on error, for example, when module is not analyzed or address doesn't point to executable code.

ulong Findprevproc(ulong addr);

Parameters:

addr - address of reference command.

See also: <u>Findprocbegin</u>, <u>Findprocend</u>, <u>Findnextproc</u>, <u>Getproclimits</u>

Findnextproc

Returns start address of the procedure that is next to addr, or 0 on error, for example, when module is not analyzed or address doesn't point to executable code.

ulong Findnextproc(ulong addr);

Parameters:

addr - address of reference command.

See also: <u>Findprocbegin</u>, <u>Findprocend</u>, <u>Findprevproc</u>, <u>Getproclimits</u>

Getproclimits

Calculates limits of the procedure that includes specified address. Returns 0 on success and -1 on error, for example, when module is not analyzed or address points to no procedure.

int Getproclimits(ulong addr,ulong *start,ulong *end);

Parameters:

addr - address of any command within the procedure;

start - pointer to variable that receives start address of the procedure;

end - pointer to variable that receives address of the last command in the procedure.

See also: <u>Findprocbegin</u>, <u>Findprocend</u>, <u>Findprevproc</u>, <u>Findnextproc</u>

Search functions

The functions described in this section have little value for plugin developer and exported mainly for use in command line plugin. They search for specified sort of data and display results in the reference window.

int Findallcommands(t_dump *pd,t_asmmodel *model,ulong origin,char *title);

int <u>Findalldllcalls(t_dump</u> *pd,ulong origin,char *title);

int Findallsequences(t_dump *pd,t_extmodel model[NSEQ][NMODELS],ulong
origin,char *title);

int Findreferences(ulong base,ulong size,ulong addr0,ulong addr1,ulong origin,int recurseonjump,char *title);

int Findstrings(ulong base,ulong size,ulong origin,char *title);

Findalldllcalls

Searches for all calls (including indirect) to different modules from the code section described by dump structure, places them into the reference table as a set of t_ref records and displays in reference window. Address of origin, if not 0, is also included into the table (marked as TY_ORIGIN). Returns number of found references or -1 on error. Notice that this function doesn't work on file dump.

int Findalldllcalls(t_dump *pd,ulong origin,char *title);

Parameters:

pd - pointer to <u>dump descriptor</u> of code section;

origin - address of search origin or 0 if none. Search origin gives easy way to return to initial point after browsing through the found items;

title - title of reference window.

Note concerning functions that access .ini file

I hate registry! Many times I was forced to reinstall software that was still on my hard disk only because registry crashed after some hazardous experiments with hardware, or because I reinstalled Windows to get rid of trash from removed installations. Do YOU know which of your personal data resides in registry? Can you check it? Can you easily backup settings of some program and easily restore them? Or edit? In my opinion, the overcomplication of the software in the last time either comes from the fact that programmers first write and then think, or is a (rather successfull) way to make product inaccessible for a concurrent. Dixi.

Sample program

This is the annotated code of sample bookmark plugin. I place it here so that you can get quick help on all referenced functions.

|| ||

// SAMPLE PLUGIN FOR OLLYDBG //

|| ||

// This plugin allows to set up to 10 code bookmarks using keyboard shortcuts //

// or popup menus in Disassembler and then quickly return to one of the //

// bookmarks using shortcuts, popup menu or Bookmark window. Bookmarks //

 $\prime\prime$ are kept between sessions in .udd file. $\prime\prime$

// //

// VERY IMPORTANT NOTICE: COMPILE THIS DLL WITH BYTE ALIGNMENT OF STRUCTURES

// AND UNSIGNED CHAR!

#include <windows.h>

#include <stdio.h>

#include <string.h>

#include <dir.h>

#include "plugin.h"

HINSTANCE hinst; // DLL instance

HWND hwmain; // Handle of main OllyDbg window

char bookmarkwinclass[32]; // Name of bookmark window class

// OllyDbg supports and makes extensive use of special kind of data collections

// called sorted tables. A table consists of descriptor (t_table) and data. All

// data elements has same size and begin with a 3-dword header: address, size

 $\ensuremath{\prime\prime}$ and type. Table automatically sorts items by address, overlapping is not

// allowed. Our bookmark table consists of elements of type t_bookmark.

typedef struct t_bookmark {

ulong index; // Bookmark index (0..9)

ulong size; // Size of index, always 1 in our case

ulong type; // Type of entry, always 0

ulong addr; // Address of bookmark

} t_bookmark;

t_table bookmark; // Bookmark table

// Functions in this file are placed in more or less "chronological" order,

// i.e. order in which they will be called by OllyDbg. This requires forward

// referencing.

int Bookmarksortfunc(t_bookmark *b1,t_bookmark *b2,int sort);

LRESULT CALLBACK Bookmarkwinproc(HWND hw,UINT msg,WPARAM wp,LPARAM lp);

int Bookmarkgettext(char *s,char *mask,int *select,<u>t_sortheader</u> *ph,int column);

void Createbookmarkwindow(void);

// Entry point into a plugin DLL. Many system calls require DLL instance

// which is passed to DllEntryPoint() as one of parameters. Remember it.

// Preferrable way is to place initializations into <a>ODBG_Plugininit() and

// cleanup in <u>ODBG_Plugindestroy()</u>.

BOOL WINAPI DllEntryPoint(HINSTANCE hi,DWORD reason,LPVOID reserved) {

if (reason==DLL_PROCESS_ATTACH)

hinst=hi; // Mark plugin instance

return 1; // Report success

};

// ODBG_Plugindata() is a "must" for valid OllyDbg plugin. It must fill in // plugin name and return version of plugin interface. If function is absent, // or version is not compatible, plugin will be not installed. Short name // identifies it in the Plugins menu. This name is max. 31 alphanumerical // characters or spaces + terminating '\0' long. To keep life easy for users, // this name should be descriptive and correlate with the name of DLL. extc int _export cdecl ODBG_Plugindata(char shortname[32]) { strcpy(shortname,"Bookmarks"); // Name of plugin return PLUGIN_VERSION; };

// OllyDbg calls this obligatory function once during startup. Place all
// one-time initializations here. If all resources are successfully allocated,
// function must return 0. On error, it must free partially allocated resources
// and return -1, in this case plugin will be removed. Parameter ollydbgversion
// is the version of OllyDbg, use it to assure that it is compatible with your
// plugin; hw is the handle of main OllyDbg window, keep it if necessary.
// Parameter features is reserved for future extentions, do not use it.
extc int _export cdecl ODBG_Plugininit(
int ollydbgversion,HWND hw,ulong *features) {

// Check that version of OllyDbg is correct.

if (ollydbgversion<PLUGIN_VERSION)

return -1;

// Keep handle of main OllyDbg window. This handle is necessary, for example, // to display message box.

hwmain=hw;

// Initialize bookmark data. Data consists of elements of type t_bookmark,

// we reserve space for 10 elements. If necessary, table will allocate more

// space, but in our case maximal number of bookmarks is 10. Elements do not

// allocate memory or other resources, so destructor is not necessary.

if (Createsorteddata(&(bookmark.data),"Bookmarks",

sizeof(t_bookmark),10,(<u>SORTFUNC</u> *)Bookmarksortfunc,NULL)!=0)

return -1; // Unable to allocate bookmark data

// Register window class for MDI window that will display plugins. Please

// note that formally this class belongs to instance of main OllyDbg program,

// not a plugin DLL. String bookmarkwinclass gets unique name of new class.

// Keep it to create window and unregister on shutdown.

if (<u>Registerpluginclass</u>(bookmarkwinclass,NULL,hinst,Bookmarkwinproc)<0) {

// Failure! Destroy sorted data and exit.

Destroysorteddata(&(bookmark.data));

return -1; };

// Plugin successfully initialized. Now is the best time to report this fact

// to the log window. To conform OllyDbg look and feel, please use two lines.

// The first, in black, should describe plugin, the second, gray and indented

// by two characters, bears copyright notice.

<u>Addtolist(0,0,"Bookmarks sample plugin v1.10 (plugin demo)");</u>

Addtolist(0,-1," Copyright (C) 2001-2004 Oleh Yuschuk");

// OllyDbg saves positions of plugin windows with attribute TABLE_SAVEPOS to

// the .ini file but does not automatically restore them. Let us add this

// functionality here. I keep information whether window was open when

// OllyDbg terminated also in ollydbg.ini. This information is saved in

// ODBG_Pluginclose. To conform to OllyDbg norms, window is restored only
// if corresponding option is enabled.

if (<u>Plugingetvalue</u>(VAL_RESTOREWINDOWPOS)!=0 &&

Pluginreadintfromini(hinst,"Restore bookmarks window",0)!=0)

Createbookmarkwindow();

return 0;

};

// To sort sorted data by some criterium, one must supply sort function that

// returns -1 if first element is less than second, 1 if first element is

// greater and 0 if elements are equal according to criterium sort. Usually

// this criterium is the zero-based index of the column in window.

int Bookmarksortfunc(t_bookmark *b1,t_bookmark *b2,int sort) {

int i=0;

if (sort==1) { // Sort by address of bookmark

if (b1->addr<b2->addr) i=-1;

else if (b1->addr>b2->addr) i=1; };

// If elements are equal or sorting is by the first column, sort by index.

```
if (i==0) {
```

```
if (b1->index<b2->index) i=-1;
```

else if (b1->index>b2->index) i=1; };

return i;

};

 $\ensuremath{\textit{//}}\xspace$ Each window class needs its own window procedure. Both standard and custom

// OllyDbg windows must pass some system and OllyDbg-defined messages to

// Tablefunction(). See description of Tablefunction() for more details.

LRESULT CALLBACK Bookmarkwinproc(HWND hw,UINT msg,WPARAM wp,LPARAM lp) {

int i,shiftkey,controlkey;

HMENU menu;

t_bookmark *pb;

switch (msg) {

// Standard messages. You can process them, but - unless absolutely sure -

// always pass them to Tablefunction().

case WM_DESTROY:

case WM_MOUSEMOVE:

case WM_LBUTTONDOWN:

case WM_LBUTTONDBLCLK:

case WM_LBUTTONUP:

case WM_RBUTTONDOWN:

case WM_RBUTTONDBLCLK:

case WM_HSCROLL:

case WM_VSCROLL:

case WM_TIMER:

case WM_SYSKEYDOWN:

Tablefunction(&bookmark,hw,msg,wp,lp);

break; // Pass message to DefMDIChildProc()

// Custom messages responsible for scrolling and selection. User-drawn

// windows must process them, standard OllyDbg windows without extra

// functionality pass them to Tablefunction().

case <u>WM_USER_SCR</u>:

case <u>WM_USER_VABS</u>:

case <u>WM_USER_VREL</u>:

case <u>WM_USER_VBYTE</u>:

case <u>WM_USER_STS</u>:

case <u>WM_USER_CNTS</u>:

case <u>WM_USER_CHGS</u>:

return Tablefunction(&bookmark,hw,msg,wp,lp);

// If window should support TABLE_ONTOP ("<u>Always on top</u>" mode), it must
pass

// WM_WINDOWPOSCHANGED to Tablefunction().

case WM_WINDOWPOSCHANGED:

return Tablefunction(&bookmark,hw,msg,wp,lp);

case <u>WM_USER_MENU</u>:

menu=CreatePopupMenu();

// Find selected bookmark. Any operations with bookmarks make sense only

// if at least one bookmark exists and is selected. Note that sorted data

// has special sort index table which is updated only when necessary.

// Getsortedbyselection() does this; some other sorted data functions

// don't and you must call <u>Sortsorteddata()</u>. Read documentation!

pb=(t_bookmark *)Getsortedbyselection(

&(bookmark.data),bookmark.data.selected);

if (menu!=NULL && pb!=NULL) {

AppendMenu(menu,MF_STRING,1,"&Follow\tEnter");

AppendMenu(menu,MF_STRING,2,"&Delete\tDel"); };

// Even when menu is NULL, call to <u>Tablefunction</u> is still meaningful.

i=<u>Tablefunction(&bookmark,hw,WM_USER_MENU,0,(LPARAM)menu);</u>

if (menu!=NULL) DestroyMenu(menu);

if (i==1) // Follow bookmark in Disassembler

<u>Setcpu</u>(0,pb->addr,0,0,CPU_ASMHIST|CPU_ASMCENTER|CPU_ASMFOCUS);

else if (i==2) { // Delete bookmark

Deletesorteddata(&(bookmark.data),pb->index);

// There is no automatical window update, do it yourself.

InvalidateRect(hw,NULL,FALSE); };

return 0;

```
case WM_KEYDOWN:
```

```
// Processing of WM_KEYDOWN messages is - surprise, surprise - very
```

// similar to that of corresponding menu entries.

shiftkey=GetKeyState(VK_SHIFT) & 0x8000;

controlkey=GetKeyState(VK_CONTROL) & 0x8000;

if (wp==VK_RETURN && shiftkey==0 && controlkey==0) {

// Return key follows bookmark in Disassembler.

pb=(t_bookmark *)Getsortedbyselection(

&(bookmark.data),bookmark.data.selected);

```
if (pb!=NULL)
```

```
Setcpu(0,pb-
>addr,0,0,CPU_ASMHIST|CPU_ASMCENTER|CPU_ASMFOCUS);
```

```
else if (wp==VK_DELETE && shiftkey==0 && controlkey==0) {
```

// DEL key deletes bookmark.

pb=(t_bookmark *)Getsortedbyselection(

&(bookmark.data),bookmark.data.selected);

if (pb!=NULL) {

Deletesorteddata(&(bookmark.data),pb->index);

```
InvalidateRect(hw,NULL,FALSE);
```

}; }

else

// Add all this arrow, home and pageup functionality.

```
Tablefunction(&bookmark,hw,msg,wp,lp);
```

break;

case <u>WM_USER_DBLCLK</u>:

// Doubleclicking row follows bookmark in Disassembler.

pb=(t_bookmark *)Getsortedbyselection(

&(bookmark.data),bookmark.data.selected);

if (pb!=NULL)

```
Setcpu(0,pb-
>addr,0,0,CPU_ASMHIST|CPU_ASMCENTER|CPU_ASMFOCUS);
```

return 1; // Doubleclick processed

```
case <u>WM_USER_CHALL</u>:
```

```
case <u>WM_USER_CHMEM</u>:
```

// Something is changed, redraw window.

InvalidateRect(hw,NULL,FALSE);

return 0;

```
case WM_PAINT:
```

```
// Painting of all OllyDbg windows is done by Painttable(). Make custom
```

// drawing only if you have important reasons to do this.

Painttable(hw,&bookmark,Bookmarkgettext);

return 0;

default: break;

};

return DefMDIChildProc(hw,msg,wp,lp);

};

// If you define <u>ODBG_Pluginmainloop</u>, this function will be called each time

// from the main Windows loop in OllyDbg. If there is some debug event from

// the debugged application, debugevent points to it, otherwise it is NULL. Do

// not declare this function unnecessarily, as this may negatively influence

// the overall speed!

extc void _export cdecl ODBG_Pluginmainloop(DEBUG_EVENT
*debugevent) {

};

// Record types must be unique among OllyDbg and all plugins. The best way to

// assure this is to register record type by OllDbg (Oleh Yuschuk). Registration

// is absolutely free of charge, except for email costs :)

#define TAG_BOOKMARK 0x236D420AL // Bookmark record type in .udd file

// Time to save data to .udd file! This is done by calling Pluginsaverecord()

 $\ensuremath{\textit{//}}$ for each data item that must be saved. Global, process-oriented data must

// be saved in main .udd file (named by .exe); module-relevant data must be

// saved in module files. Don't forget to save all addresses relative to

// module's base, so that data will be restored correctly even when module is

// relocated.

extc void _export cdecl ODBG_Pluginsaveudd(t_module *pmod,int
ismainmodule) {

int i;

ulong data[2];

t_bookmark *pb;

```
if (ismainmodule==0)
```

return; // Save bookmarks to main file only

pb=(t_bookmark *)bookmark.data.data;

```
for (i=0; i<bookmark.data.n; i++,pb++) {
```

data[0]=pb->index;

```
data[1]=pb->addr;
```

Pluginsaverecord(TAG_BOOKMARK,2*sizeof(ulong),data);

};

};

// OllyDbg restores data from .udd file. If record belongs to plugin, it must // process record and return 1, otherwise it must return 0 to pass record to // other plugins. Note that module descriptor pointed to by pmod can be // incomplete, i.e. does not necessarily contain all informations, especially // that from .udd file.

extc int _export cdecl <u>ODBG_Pluginuddrecord(t_module</u> *pmod,int ismainmodule,

ulong tag,ulong size,void *data) {

t_bookmark mark;

if (ismainmodule==0)

return 0; // Bookmarks saved in main file only

if (tag!=TAG_BOOKMARK)

```
return 0; // Tag is not recognized
```

```
mark.index=((ulong *)data)[0];
```

mark.size=1;

```
mark.type=0;
```

```
mark.addr=((ulong *)data)[1];
```

Addsorteddata(&(bookmark.data),&mark);

return 1; // Record processed

};

// Function adds items either to main OllyDbg menu (origin=PM_MAIN) or to
popup

// menu in one of standard OllyDbg windows. When plugin wants to add own menu

// items, it gathers menu pattern in data and returns 1, otherwise it must

// return 0. Except for static main menu, plugin must not add inactive items.

// Item indices must range in 0..63. Duplicated indices are explicitly allowed.

extc int _export cdecl ODBG_Pluginmenu(int origin, char data[4096], void
*item) {

int i,n;

t_bookmark *pb;

t_dump *pd;

switch (origin) {

// Menu creation is very simple. You just fill in data with menu pattern.

// Some examples:

// 0 Aaa,2 Bbb|3 Ccc|,, - linear menu with 3 items, relative IDs 0, 2 and

// 3, separator between second and third item, last

// separator and commas are ignored;

// #A{0Aaa,B{1Bbb|2Ccc}} - unconditional separator, followed by popup menu

// A with two elements, second is popup with two

// elements and separator inbetween.

case PM_MAIN: // Plugin menu in main window

strcpy(data,"0 &Bookmarks|1 &About");

// If your plugin is more than trivial, I also recommend to include Help.

return 1;

case PM_DISASM: // Popup menu in Disassembler

// First check that menu applies.

pd=(t_dump *)item;

if (pd==NULL || pd->size==0)

return 0; // Window empty, don't add

// Start second-level popup menu.

n=sprintf(data,"Bookmark{");

// Add item "Insert bookmark n" if there are free bookmarks and some part

// of Disassembler is selected. Note that OllyDbg correctly interpretes

// superfluos commas, separators and, to some extent, missed braces.

pb=(t_bookmark *)bookmark.data.data;

for (i=0; i<bookmark.data.n; i++)

if (pb[i].index!=(ulong)i) break;

if (i<10 && pd->sel1>pd->sel0)

n+=sprintf(data+n,"%i &Insert bookmark %i\tAlt+Shift+%i,",i,i,i);

// Add item "Delete bookmark n" for each available bookmark. Menu

// identifiers are not necessarily consecutive.

for (i=0; i<bookmark.data.n; i++) {

n+=sprintf(data+n,"%i Delete bookmark %i,",pb[i].index+10,pb[i].index);

};

// Add separator to menu.

data[n++]='|';

// Add item "Go to bookmark n" for each available bookmark. Bookmarks

```
// set at selected command are not shown.
```

```
for (i=0; i<bookmark.data.n; i++) {
```

```
if (pb[i].addr==pd->sel0) continue;
```

```
n+=sprintf(data+n,"%i Go to bookmark %i\tAlt+%i,",
```

```
pb[i].index+20,pb[i].index,pb[i].index);
```

```
;
```

```
};
```

```
// Close popup. If you forget to do this, OllyDbg will try to correct
```

// your error.

```
sprintf(data+n,"}");
```

return 1;

```
default: break; // Any other window
```

};

return 0; // Window not supported by plugin

};

// This optional function receives commands from plugin menu in window of type

// origin. Argument action is menu identifier from ODBG_Pluginmenu(). If user

// activates automatically created entry in main menu, action is 0.

extc void _export cdecl ODBG_Pluginaction(int origin, int action, void *item) {
 t_bookmark mark, *pb;

t_dump *pd;

```
if (origin==PM_MAIN) {
```

switch (action) {

case 0:

// Menu item "Bookmarks", creates bookmark window.

Createbookmarkwindow();

break;

case 1:

// Menu item "About", displays plugin info.

MessageBox(hwmain,

"Bookmark plugin v1.10\n"

"(demonstration of plugin capabilities)\n"

"Copyright (C) 2001-2004 Oleh Yuschuk",

"Bookmark plugin", MB_OK | MB_ICONINFORMATION);

break;

default: break;

```
}; }
```

else if (origin==PM_DISASM) {

pd=(t_dump *)item;

if (action>=0 && action<10) { // Insert bookmark

mark.index=action;

```
mark.size=1;
```

mark.type=0;

```
mark.addr=pd->sel0;
```

Addsorteddata(&(bookmark.data),&mark);

```
if (bookmark.hw!=NULL) InvalidateRect(bookmark.hw,NULL,FALSE); }
```

```
else if (action>=10 && action<20) {// Delete bookmark
```

pb=(t_bookmark *)Findsorteddata(&(bookmark.data),action-10);

if (pb!=NULL) {

Deletesorteddata(&(bookmark.data),action-10);

```
if (bookmark.hw!=NULL) InvalidateRect(bookmark.hw,NULL,FALSE);
```

}; }

```
else if (action>=20 && action<30) {//Go to bookmark
```

```
pb=(t_bookmark *)Findsorteddata(&(bookmark.data),action-20);
```

```
if (pb!=NULL) {
```

```
Setcpu(0,pb-
>addr,0,0,CPU_ASMHIST|CPU_ASMCENTER|CPU_ASMFOCUS);
```

```
};
};
};
};
```

// Standard function <u>Painttable()</u> makes most of OllyDbg windows redrawing. You // only need to supply another function that prepares text strings and

// optionally colours them. Case of custom windows is a bit more complicated,

// please read documentation.

int Bookmarkgettext(char *s,char *mask,int *select,

```
t_sortheader *ph,int column) {
```

int n;

ulong cmdsize, decodesize;

char cmd[<u>MAXCMDSIZE</u>],*pdecode;

t_memory *pmem;

t_disasm da;

t_bookmark *pb=(t_bookmark *)ph;

if (column==0) { // Name of bookmark

// Column 0 contains name of bookmark in form "Alt+n", where n is the

// digit from 0 to 9. Mainly for demonstration purposes, I display prefix

// "Alt+" in grayed and digit in normal text. Standard table windows do

// not need to bother about selection.

n=sprintf(s,"Alt+%i",pb->index);

*select=DRAW_MASK;

memset(mask,DRAW_GRAY,4);

mask[4]=DRAW_NORMAL; }

else if (column==1) // Address of bookmark

n=sprintf(s,"%08X",pb->addr);

else if (column==2) { // Disassembled command

// Function <u>Disasm()</u> requires that calling routine supplies code to be

// disassembled. Read this code from memory. First determine possible

// code size.

pmem=Findmemory(pb->addr); // Find memory block containing code

if (pmem==NULL) {

*select=DRAW_GRAY; return sprintf(s,"???"); };

cmdsize=pmem->base+pmem->size-pb->addr;

if (cmdsize><u>MAXCMDSIZE</u>)

cmdsize=<u>MAXCMDSIZE;</u>

```
if (<u>Readmemory</u>(cmd,pb-
>addr,cmdsize,MM_RESTORE|MM_SILENT)!=cmdsize) {
```

```
*select=DRAW_GRAY; return sprintf(s,"???"); };
```

```
pdecode=<u>Finddecode(pb->addr,&decodesize);</u>
```

```
if (decodesize<cmdsize) pdecode=NULL;</pre>
```

Disasm(cmd,cmdsize,pb->addr,pdecode,&da,DISASM_CODE,0);

strcpy(s,da.result);

n=strlen(s); }

else if (column==3) // Comment

// Only user-defined comments are displayed here.

n=<u>Findname(pb->addr,NM_COMMENT,s);</u>

else n=0; // s is not necessarily 0-terminated

return n;

};

// OllyDbg makes most of work when creating standard MDI window. Plugin must

// only describe number of columns, their properties and properties of window

// as a whole.

void Createbookmarkwindow(void) {

// Describe table columns. Note that column names are pointers, so strings

// must exist as long as table itself.

if (bookmark.bar.nbar==0) {

// Bar still uninitialized.

bookmark.bar.name[0]="Bookmark"; // Name of bookmark

bookmark.bar.defdx[0]=9;

bookmark.bar.mode[0]=0;

bookmark.bar.name[1]="Address"; // Bookmark address

bookmark.bar.defdx[1]=9;

bookmark.bar.mode[1]=0;

bookmark.bar.name[2]="Disassembly";// Disassembled command

bookmark.bar.defdx[2]=32;

bookmark.bar.mode[2]=BAR_NOSORT;

bookmark.bar.name[3]="Comment"; // Comment

bookmark.bar.defdx[3]=256;

bookmark.bar.mode[3]=BAR_NOSORT;

bookmark.bar.nbar=4;

bookmark.mode= // Note: new option <u>TABLE_ONTOP</u>

TABLE_COPYMENU|TABLE_SORTMENU|TABLE_APPMENU|TABLE_SAV

bookmark.drawfunc=Bookmarkgettext; };

// If window already exists, <u>Quicktablewindow()</u> does not create new window,

// but restores and brings to top existing. This is the simplest way,

// <u>Newtablewindow()</u> is more flexible but more complicated. I do not recommend

// custom (plugin-drawn) windows without very important reasons to do this.

Quicktablewindow(&bookmark,15,4,bookmarkwinclass,"Bookmarks");

};

// This function receives possible keyboard shortcuts from standard OllyDbg

// windows. If it recognizes shortcut, it must process it and return 1,

// otherwise it returns 0.

extc int _export cdecl ODBG_Pluginshortcut(

int origin,int ctrl,int alt,int shift,int key,void *item) {

t_dump *pd;

t_bookmark mark,*pm;

// Plugin accepts shortcuts in form Alt+x or Shift+Alt+x, where x is a key

// '0'..'9'. Shifted shortcut sets bookmark (only in Disassembler),

// non-shifted jumps to bookmark from everywhere.

if (ctrl==0 && alt!=0 && key>='0' && key<='9') {

if (shift!=0 && origin==PM_DISASM && item!=NULL) {

// Set new or replace existing bookmark.

pd=(t_dump *)item;

mark.index=key-'0';

mark.size=1;

mark.type=0;

```
mark.addr=pd->sel0;
```

Addsorteddata(&(bookmark.data),&mark);

if (bookmark.hw!=NULL) InvalidateRect(bookmark.hw,NULL,FALSE);

return 1; } // Shortcut recognized

else if (shift==0) {

// Jump to existing bookmark (from any window).

pm=<u>Findsorteddata(</u>&(bookmark.data),key-'0');

if (pm==NULL)

Flash("Undefined bookmark");

else

```
Setcpu(0,pm-
>addr,0,0,CPU_ASMHIST|CPU_ASMCENTER|CPU_ASMFOCUS);
return 1; // Shortcut recognized
};
return 0; // Shortcut not recognized
};
// Function is called when user opens new or restarts current application.
```

// Plugin should reset internal variables and data structures to initial state.

extc void _export cdecl ODBG_Pluginreset(void) {

Deletesorteddatarange(&(bookmark.data),0,0xFFFFFFF;;

};

// OllyDbg calls this optional function when user wants to terminate OllyDbg.

// All MDI windows created by plugins still exist. Function must return 0 if

// it is safe to terminate. Any non-zero return will stop closing sequence. Do

// not misuse this possibility! Always inform user about the reasons why

// termination is not good and ask for his decision!

extc int _export cdecl ODBG_Pluginclose(void) {

// For automatical restoring of open windows, mark in .ini file whether

// Bookmarks window is still open.

Pluginwriteinttoini(hinst,"Restore bookmarks window",bookmark.hw!=NULL);

return 0;

};

// OllyDbg calls this optional function once on exit. At this moment, all MDI

// windows created by plugin are already destroyed (and received WM_DESTROY

// messages). Function must free all internally allocated resources, like

// window classes, files, memory and so on.

extc void _export cdecl ODBG_Plugindestroy(void) {

Unregisterpluginclass(bookmarkwinclass);

Destroysorteddata(&(bookmark.data));

};

Attachtoactiveprocess

Attaches OllyDbg to active (running) process with known process identifier. If another process is debugged, asks for permission to close it. Returns 0 on success and -1 on error.

int Attachtoactiveprocess(int processid);

Parameters:

processid - identifier of running process.

See also: **OpenEXEfile**

Creatertracewindow

Creates new or brings to top existing window displaying run trace history. Only one such window may exist at a time. Returns handle of the window or NULL on error.

HWND Creatertracewindow(void);

Demanglename

Demangles or undecorates name. Currently supports Borland and Microsoft mangling schemes. Returns 0 if name is not mangled (in this case buffer pointed to by undecorated is invalid and probably modified) and length of unmangled name on success. Attention, no guarantee that demangled name is unique!

int Demanglename(char *name,int type,char *undecorated);

Parameters:

name - pointer to mangled name;

type - type of name. Function treats names of types NM_IMPORT and NM_IMPNAME in a special way;

undecorated - pointer to output buffer of length at least <u>TEXTLEN</u> characters.