

ODBJscript

javascript plugin

[Main Page](#)

[Related Pages](#)

[Files](#)

Related Pages

Here is a list of all related documentation pages:

- [Todo List](#)

Generated on Fri Jan 27 2012 17:54:30 for ODBJscript by [doxygen](#) 1.7.5.1

ODBJavascript

javascript plugin

Main Page

Related Pages

Files

Todo List

Global BPD (char *procname)

Add BPD API support.

Global BPGOTO (ulong addr, FUNC *proc)

Add BPGOTO support.

Global BUF (var)

Add BUF API support.

Global CLOSE (window)

Add CLOSE API support.

Global DBH (void)

Add DBH API support.

Global DBS (void)

Add DBS API support.

Global DPE (char *filename, ulong ep)

Add DPE API support.

Global ENDE (void)

Add ENDE API support.

Global ENDSEARCH (void)

Add ENDSEARCH API support.

Global EXEC (void)

Add EXEC API support.

Global GAPI (ulong addr)

Add GAPI API support.

Global GBPM ()

Add GBPM API support.

Global GMEXP (ulong moduleaddr, int info,[int num])

Add GMEXP API support.

Global GN (ulong addr)

Add GN API support.

Global GREF ([int line])

Add GREF API support.

Global GRO (ulong addr)

Add GRO API support.

Global KEY (int vkcode, bool shift=false, bool ctrl=false, int origin=PM_DISASM)

Fix some issues with this api.

Global LM (ulong addr, int size, char *filename)

Add LM API support.

Global LOADLIB (char *dllname)

Add LOADLIB API support.

Global MEMCPY (ulong dest, ulong src, int size)

Add MEMCPY API support.

Global NAMES (ulong addr)

Add NAMES API support.

Global OLLY (int info)

Add OLLY API support.

Global OPENDUMP (ulong addr, ulong base, ulong size, int type=Hex/ASCII)

Fix OPENDUMP API close after open.

Global PAUSE (void)

Add PAUSE API support.

Global POP ()

fix pop something

Global POPA (void)

Add POPA API support.

Global PUSHA (void)

Add PUSHA API support.

Global STR (var)

Add STR API support.

ODBJscript

javascript plugin

Main Page

Related Pages

Files

File List

Globals

File List

Here is a list of all files with brief descriptions:

[odbgscriphelper.cpp](#) 

[ODBJScript.h](#) [code] 

Generated on Fri Jan 27 2012 17:54:31 for ODBJscript by [**doxygen**](#) 1.7.5.1

ODBJavascript

javascript plugin

[Main Page](#)

[Related Pages](#)

[Files](#)

[File List](#)

[Globals](#)

odbgscriphelper.cpp File Reference

Generated on Fri Jan 27 2012 17:54:31 for ODBJavascript by [doxygen](#) 1.7.5.1

ODBJavascript

javascript plugin

[Main Page](#)

[Related Pages](#)

[Files](#)

[File List](#)

[Globals](#)

[Enumerations](#) | [Functions](#)

ODBJScript.h File Reference

[Go to the source code of this file.](#)

Enumerations

```
enum e_exceptmask {
    IGNORE_NONE = 0x00000000, IGNORE_ACCESS_KER32 =
    IGNORE_ACCESS = 0x00000010, IGNORE_DIV_ZERO = 0x00000020,
    IGNORE_CUSTOM_EXCEPT = 0x00000200, IGNORE_MANU =
    IGNORE_ACCESS_KER32|IGNORE_INT3|IGNORE_TRAP|IGI
}

enum e_memoryinfo { MEMORYBASE, MEMORIESIZE, MEMORYO

enum t_moduleinfo {
    MODULEBASE, MODULESIZE, CODEBASE, CODESIZE,
    ENTRY, NSECT, DATABASE, EDATATABASE,
    EDATASIZE, IDATATABASE, RESBASE, RESSIZE,
    RELOCTABLE, RELOCSIZE
}

enum e_cmdinfo {
    COMMAND, CONDITION, DESTINATION, FINALDEST,
    SIZE, TYPE
}

enum e_dumptype {
    DU_HEXTTEXT = 0x01000L, DU_TEXT = 0x02000L, DU_UNIC
    DU_UINT = 0x05000L, DU_IHEX = 0x06000L, DU_FLOAT = 0
    DU_DISASM = 0x09000L, DU_HEXUNI = 0xA0000L, DU_ADF
}

enum e_processinfo {
    HPROCESS = 20, PROCESSID, HMAINTHREAD, MAINTHR
    MAINBASE, PROCESSNAME, EXEFILENAME, CURRENTDI
    SYSTEMDIR
}

enum e_origin { CPUDASM, CPUDUMP, CPUSTACK }
```

Functions

ulong	DD (ulong <i>addr</i>)	read dword from address <i>addr</i>
ulong	DW (ulong <i>addr</i>)	read dword from address <i>addr</i>
ulong	DB (ulong <i>addr</i>)	read dword from address <i>addr</i>
ulong	SETOPTION (DWORD <i>mask</i>)	sets/unsets exception to be ignored by ollydbg.
void	EVAL (void)	
void	MSGYN (void)	
void	MSG (void)	
void	REFRESH (bool <i>onoff</i>)	sets on/off refreshing ollydbg MDI windows.
void	LOGBUF (ulong <i>ip</i> , size_t <i>size</i> , int <i>nepl</i> =32, char * <i>separator</i> = "")	logs <i>size</i> bytes of memory beginning from address <i>ip</i> .
void	LOG (...)	logs any data to log window.
void	TICND (char * <i>cond</i> , ulong <i>in0</i> , ulong <i>in1</i> , ulong <i>out0</i> , ulong <i>out1</i>)	Traces into calls until cond is true and/or on EIP in range [in0-in1] and/or out of range [out0-out1].
void	BPGOTO (ulong <i>addr</i> , FUNC * <i>proc</i>)	Installs a callback function that will be called each time breakpoint at address <i>addr</i> is hit.
void	EOB (FUNC * <i>proc</i>)	installs a callbacks function that will be called every time a breakpoint is hit.
void	EOE (FUNC * <i>proc</i>)	installs a callbacks function that will be called every time an exception occurs.
void	COB ()	

void	COE ()
void	BPX (char *procname, DWORD addr=0) set bp on every caller of label <i>procname</i>
void	BPD (char *procname)
void	GBPM ()
ulong	FIND (ulong addr, char *what) search for a pattern of bytes starting from address <i>addr</i>
ulong	MOV (ulong addr, char *what, int n=length_of_what) writes <i>n</i> or all element of what to address <i>addr</i>
ulong	MOVS (ulong addr, char *str, int n=length_of_what) writes <i>n</i> or all characters of string <i>str</i> to address <i>addr</i>
ulong	FINDO (ulong addr, char *what, int blocklen, func callback,...) stands for find-do, it allows you to call a callback function each time the pattern is found.
ulong	PREOP (ulong addr) Get asm command line address just before specified address.
ulong	GCI (ulong addr, int info) stands for get command information
void	KEY (int vkcode, bool shift=false, bool ctrl=false, int origin=PM_DISASM)
void	OPENDUMP (ulong addr, ulong base, ulong size, int type=Hex/ASCII)
ulong	BUF (var)
ulong	CLOSE (window)
ulong	DBH (void)
ulong	DBS (void)
ulong	DPE (char *filename, ulong ep)
ulong	ENDE (void)
void	ENDSEARCH (void)
void	EXEC (void)
ulong	GAPI (ulong addr)
ulong	GMA (char *name, int info) returns information about module named <i>name</i>
ulong	GMEXP (ulong moduleaddr, int info,[int num])

char *	GN (ulong addr)
ulong	GREF ([int line])
ulong	GRO (ulong addr)
char *	GSTR (ulong addr[, int len]) return string of length <i>len</i> from memory <i>addr</i>
uchar *	GSTRW (ulong addr[, int len]) return unicode string of length <i>len</i> from memory <i>addr</i>
ulong	LM (ulong addr, int size, char *filename)
ulong	LOADLIB (char *dllname)
ulong	MEMCPY (ulong dest, ulong src, int size)
ulong	NAMES (ulong addr)
ulong	OLLY (int info)
void	PAUSE (void)
void	POPA (void)
void	PUSHA (void)
void	STR (var)
void	AI (void)
void	AO (void)
void	ERUN (void)
void	ESTI (void)
void	GO (ulong addr)
void	RTR (void)
void	RTU (void)
void	RUN (void)
void	STI (void)
void	STO (void)
void	TI (void)
void	TO (void)
void	TOCND (char *cond) Traces over calls until cond is true and/or on EIP in range [in0-in1] and/or out of range [out0-out1].
void	CMT (char *comment)
void	AN (ulong addr)
void	BC (ulong addr)
void	BD (ulong addr)

```
void BE (ulong addr)
void BP (ulong addr)
void BPCND (ulong addr, char *cond)
void BPLCND (ulong addr, char *cond, char *expr)
void BPWM (ulong addr, int size)
void BPHWS (ulong addr, char *mode="x", int size=1)
void BPHWC (ulong addr)
void BPL (ulong addr, char *expr)
void BPMC ()
void BPRM (ulong addr, int size)
void GBPR ()
ulong GMEMI (ulong addr, int info)
ulong GPA (char *proc, char *lib, bool bKeepInMem)
ulong REPL (ulong addr, char *find, char *repl, int len)
ulong WRT (char *file, char *data)
ulong WRTA (char *file, char *data[, char *separator])
ulong OPCODE (ulong addr)
ulong FILL (ulong addr, int len, char *value)
void LBL (ulong addr, char *text)
void LC ()
void OPENTRACE ()
void TC ()
ulong ALLOC (int size)
ulong ASK (char *question)
void BACKUP (ulong addr)
ulong DM (ulong addr, int size, file)
ulong DMA (ulong addr, int size, file)
void ESTEP (void)
void FREE (ulong addr)
ulong GFO (ulong addr)
ulong GPI (info)
ulong GSL ([int where])
ulong LEN (char *str)
ulong POP ()
void PUSH (ulong dw)
char * READSTR (char *str, int len)
```

ulong	REV (ulong what)
ulong	ROL (ulong op, ulong ulong count)
ulong	ROR (ulong op, ulong count)
ulong	XCHG (object *regdest, object *regsrc) exchanges values of 2 registers of same size
void	MOVB (object *objaddr, object *objdata) batch mov
void	ASMB (object *objaddr, object *objdata) batch assemble
ulong	TICK (void) returns value of GetTickCount()

Enumeration Type Documentation

enum e_cmdinfo

Enumerator:

COMMAND asm text representation of the command

CONDITION 0xFF:unconditional, 0:false, 1:true

DESTINATION Destination of jump/call/return

FINALDEST the last non jump/call address of command in a
 jump->call/jump cascade. \$RESULT holds the
 address and \$RESULT_1 holds the address
 that lands to it (helpfull when you want either
 the called api and the caller from debuggee)

SIZE size of command in bytes

TYPE One of C_xxx

enum e_dumptype

Enumerator:

DU_HEXTEXT Hexadecimal dump with ASCII text

DU_TEXT Character dump

<i>DU_UNICODE</i>	Unicode dump
<i>DU_INT</i>	Integer signed dump
<i>DU_UINT</i>	Integer unsigned dump
<i>DU_IHEX</i>	Integer hexadecimal dump
<i>DU_FLOAT</i>	Floating-point dump
<i>DU_ADDR</i>	Address dump
<i>DU_DISASM</i>	Disassembly
<i>DU_HEXUNI</i>	Hexadecimal dump with UNICODE text
<i>DU_ADRASC</i>	Address dump with ASCII text
<i>DUADRUNI</i>	Address dump with UNICODE text

enum e_exceptmask

ignore means check box of the exception

Enumerator:

unckeck all exception == don't

<i>IGNORE_NONE</i>	ignore any exception
<i>IGNORE_ACCESS_KER32</i>	check ignore access kernel32 exception
<i>IGNORE_INT3</i>	check ignore int3 exception
<i>IGNORE_TRAP</i>	check ignore single step exception
<i>IGNORE_ACCESS</i>	check ignore memory access violation exception
<i>IGNORE_DIV_ZERO</i>	check ignore division by zero exception
<i>IGNORE_ILLEGAL_INST</i>	check ignore invalid or privileged instruction exception
<i>IGNORE_FPU_EXCEPT</i>	check ignore all fpu exception
<i>IGNORE_CUSTOM_EXCEPT</i>	check ignore user costum exception
<i>IGNORE_MANUALLY</i>	set manually
<i>IGNORE_ALL</i>	check to ignore all exceptions bellow

enum e_memoryinfo

Enumerator:

MEMORYBASE Base address of block memory

MEMORYSIZE Size occupied by block memory

MEMORYOWNER owner of block memory

enum e_origin

Enumerator:

CPUDASM

CPUDUMP

CPUSTACK

enum e_processinfo

Enumerator:

HPROCESS Handle of Debuggee

PROCESSID Process ID of Debuggee

HMAINTHREAD Handle of main thread

MAINTHREADID Thread ID of main thread

MAINBASE Base of main module in the process

PROCESSNAME Name of the active process

EXEFILENAME Name of the main debugged file

CURRENTDIR Current directory for debugged process

SYSTEMDIR Windows system directory

enum t_moduleinfo

Enumerator:

MODULEBASE Base address of module

MODULESIZE Size occupied by module

CODEBASE Base address of module code block

CODESIZE Size of module code block

ENTRY Address of <ModuleEntryPoint> or NULL

NSECT Number of sections in the module

<i>DATABASE</i>	Base address of module data block
<i>EDATATABLE</i>	Base address of export data table
<i>EDATASIZE</i>	Size of export data table
<i>IDATATABLE</i>	Base address of import data table
<i>RESBASE</i>	Base address of resources
<i>RESSIZE</i>	Size of resources
<i>RELOCTABLE</i>	Base address of relocation table
<i>RELOCSIZE</i>	Size of relocation table

Function Documentation

void AI (void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

ulong ALLOC (int size)

see ODBGScript documentation for details.

```
var mem = ALLOC(0x1000);
if(mem) {
    MOV(mem, "558BEC81C400FEFFFF68005040008D8500F
EFFFFF50E80F2B000068545740008D8500FEFFFF50E8F22A00
008D8500FEFFFF50FF7508E8732B0000C9C20400");
}
LOG(mem);
// FREE(mem); // if you don't wanna keep allocated memory mem
```

void AN (ulong addr)

see ODBGScript documentation for details.

```
AN(eip.v);
```

void AO (void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

ulong ASK (char * question)

see ODBGScript documentation for details.

```
// @todo give example
```

void ASMB (object * objaddr, object * odjdata)

batch assemble

Parameters:

- objaddr** javascript array containing address
- odjdata** javascript array of strings (assembly commands).
Each element is assembled to corresponding address from objaddr.

asmb stands for batch assemble, i.e. you can assemble to a debugger memory page tons of commands at once. As access of debugger memory is a time consuming operation, this function can accelerate the process hundreds of times as it reads the whole page, modifies it and then rewrite it back one time.

See also:

MOVB

```
function jsmain() {
    var my_array = [[], []]; // array composed by 2 arrays

    // populate address object
    my_array[0].push(0x401000);
    my_array[0].push(0x401005);
    my_array[0].push(0x40100A);
    my_array[0].push(0x40100F);
    my_array[0].push(0x401014);
    my_array[0].push(0x401019);
    my_array[0].push(0x40101E);
    my_array[0].push(0x401023);
    my_array[0].push(0x401028);
    my_array[0].push(0x40102D);

    // populate data object
    my_array[1].push("call 501000");
    my_array[1].push("call 502000");
    my_array[1].push("call 503000");
    my_array[1].push("call 504000");
    my_array[1].push("call 505000");
    my_array[1].push("call 506000");
```

```
    my_array[1].push("call 507000");
    my_array[1].push("call 508000");
    my_array[1].push("call 509000");
    my_array[1].push("call 50A000");

    tc = TICK();
    for (var i = 0; i < my_array[0].length; i++)
        ASM(my_array[0][i], my_array[1][i]);
        LOG("calling asm in loop took:", TICK() - tc, " ms");

    tc = TICK();
    ASMB(my_array[0], my_array[1]);
    LOG("calling asmb took:", TICK() - tc,
        " ms");
}
```

void BACKUP (ulong addr)

see ODBGScript documentation for details.

```
BACKUP(0x40115C);
```

void BC (ulong addr)

see ODBGScript documentation for details.

```
BP(0x401000);
BC(0x401000);
BC(); // to clear all breakpoints
BD(0x401000);
BD(); // to disable all breakpoints
BPCND(0x401000, "eax==4");
```

```
BE(0x401000);
BE(); // to enable all breakpoints
```

void BD (ulong addr)

see ODBGScript documentation for details.

```
BP(0x401000);
BC(0x401000);
BC(); // to clear all breakpoints
BD(0x401000);
BD(); // to disable all breakpoints
BPCND(0x401000, "eax==4");
BE(0x401000);
BE(); // to enable all breakpoints
```

void BE (ulong addr)

see ODBGScript documentation for details.

void BP (ulong addr)

see ODBGScript documentation for details.

```
BP(0x401000);
BC(0x401000);
BC(); // to clear all breakpoints
BD(0x401000);
BD(); // to disable all breakpoints
BPCND(0x401000, "eax==4");
BE(0x401000);
BE(); // to enable all breakpoints
```

void BPCND (ulong addr,

```
    char * cond  
)
```

see ODBGScript documentation for details.

```
BP(0x401000);  
BC(0x401000);  
BC(); // to clear all breakpoints  
BD(0x401000);  
BD(); // to disable all breakpoints  
BPCND(0x401000, "eax==4");  
BE(0x401000);  
BE(); // to enable all breakpoints
```

void BPD (char * procname)

Parameters:

procname

Returns:

Todo:

Add BPD API support.

```
void BPGOTO ( ulong addr,  
               FUNC * proc  
)
```

Installs a callback function that will be called each time breakpoint at address *addr* is hit.

Parameters:

addr address of breakpoint to monitor.

proc javascript function that will be called each time breakpoint at address *addr* is hit.

Returns:

must return true to continue execution, otherwise it must return false;

Todo:

add BPGOTO support.

Installs a callback function that will be called each time breakpoint at address *addr* is hit.

void BPHWC (ulong addr)

see ODBGScript documentation for details.

```
BPHWC(0x401000);  
BPHWC(); // to clear all hardware breakpoints
```

**void BPHWS (ulong addr,
 char * mode = "x",
 int size = 1
)**

see ODBGScript documentation for details.

```
/* sets hardware breakpoint on execution and on  
reading at address 0x401168  
* of course the size is optional, and only meaningful  
for read and write mode.  
*/  
BPHWS(0x401168,"rx", 2);  
BPHWS(0x401168,"rx"); // could also be written to  
set on write 4 bytes.
```

void BPL (ulong addr,

```
    char * expr  
)
```

see ODBGScript documentation for details.

```
BPL(0x401163, "dword ptr [esp+8]");
```

```
void BPLCND( ulong addr,  
              char * cond,  
              char * expr  
)
```

see ODBGScript documentation for details.

```
BPLCND(0x401000, "eax > 1", "eax");
```

```
void BPMC()
```

see ODBGScript documentation for details.

```
BPMC();
```

```
void BPRM( ulong addr,  
           int      size  
)
```

see ODBGScript documentation for details.

```
BPRM(0x401000, 4);
```

```
void BPWM( ulong addr,  
           int      size  
)
```

see ODBGScript documentation for details.

```
BPWM(0x405750, 0x200);
```

```
void BPX ( char * procname,  
          DWORD addr = 0  
        )
```

set bp on every caller of label *procname*

Parameters:

procname procedure name.

addr address belonging to a memory page where you want to search for all intermodular call (imc) in. this will help you to find imc in memory not belonging to an module. if not given, memory at ACPUDUMP is presumed.

Returns:

nothing

set bp on every caller of label *procname*.

ulong BUF(var)

Parameters:

var

Returns:

Todo:

Add BUF API support.

ulong CLOSE(window)

Parameters:

window

Returns:

Todo:

Add CLOSE API support.

void CMT (char * comment)

see ODBGScript documentation for details.

[CMT\(0x401000, "CMT test"\);](#)

void COB ()

deprecated.

See also:

[EOB](#)

void COE ()

deprecated.

See also:

[EOE](#)

ulong DB (ulong addr)

read dword from address *addr*

Parameters:

addr pointer to address to read memory from.

read dword from address *addr*. Doesn't set \$RESULT

```
var dword = DD(eip.v);
dword += ecx.v;
var byt = DB(0x401000);
var word = DW(dword + (ecx.v >> byt) * 4);
```

ulong DBH (void)

Returns:

Todo:

Add DBH API support.

ulong DBS (void)

Returns:

Todo:

Add DBS API support.

ulong DD (ulong **addr**)

read dword from address *addr*

Parameters:

addr pointer to address to read memory from.

read dword from address *addr*. Doesn't set \$RESULT

```
var dword = DD(eip.v);
dword += ecx.v;
var byt = DB(0x401000);
var word = DW(dword + (ecx.v >> byt) * 4);
```

```
ulong DM( ulong addr,  
          int    size,  
          file  
        )
```

see ODBGScript documentation for details.

```
DM(0x40115C, 0x100, "barba_chater.mem");  
DMA(0x40115C+0x100, 0x100, "barba_chater.mem");
```

```
ulong DMA( ulong addr,  
          int    size,  
          file  
        )
```

see ODBGScript documentation for details.

```
DM(0x40115C, 0x100, "barba_chater.mem");  
DMA(0x40115C+0x100, 0x100, "barba_chater.mem");
```

```
ulong DPE( char * filename,  
           ulong ep  
         )
```

Parameters:

filename
ep

Returns:

Todo:

Add DPE API support.

ulong DW (ulong addr)

read dword from address *addr*

Parameters:

addr pointer to address to read memory from.

read dword from address *addr*. Doesn't set \$RESULT

```
var dword = DD(eip.v);
dword += ecx.v;
var byt = DB(0x401000);
var word = DW(dword + (ecx.v >> byt) * 4);
```

ulong ENDE (void)

Returns:

Todo:

Add ENDE API support.

void ENDSEARCH (void)

Returns:

Todo:

Add ENDSEARCH API support.

void EOB (FUNC * proc)

installs a callbacks function that will be called every time a breakpoint is hit.

Parameters:

proc the callback function.

Returns:

false to disable calling the callback function (in place of cob());, otherwise it must return true.

installs a callbacks function that will be called every time a breakpoint is hit.

```
function breakpoints_handler() {
    if(eip.v == 0x40118F)
        return false; // COB() equivalent
    as COB() is now deprecated
    LOG("this message is logged from eob");
    return true; // keep taking control of breakpoints handling
}

function exceptions_handler() {
    if(eip.v == 0x415C00)
        return false; // COE() equivalent
    as COE() is now deprecated
    LOG("this message is logged from eoe");
    return true; // keep taking control of exceptions handling
}

function jsmain() {
    BP(0x401172);
    BP(0x40115E);
    BP(0x401181);
    BP(0x40118F);
    EOB(breakpoints_handler);
    EOE(exceptions_handler);
    ESTO();
    LOG("this message is logged after eob finished");
```

```
        LOG("this message is logged after eoe finished");
    BC();
}
```

void EOE(FUNC * proc)

installs a callbacks function that will be called every time an exception occurs.

Parameters:

proc the callback function.

Returns:

false to disable calling the callback function (in place of COE()), otherwise it must return true.

installs a callbacks function that will be called every time an exception occurs.

```
function breakpoints_handler() {
    if(eip.v == 0x40118F)
        return false; // COB() equivalent
    as COB() is now deprecated
    LOG("this message is logged from eob");
    return true; // keep taking control of breakpoints handling
}
function exceptions_handler() {
    if(eip.v == 0x415C00)
        return false; // COE() equivalent
    as COE() is now deprecated
    LOG("this message is logged from eoe");
    return true; // keep taking control of exceptions handling
}
```

```
function jsmain() {
    BP(0x401172);
    BP(0x40115E);
    BP(0x401181);
    BP(0x40118F);
    EOB(breakpoints_handler);
    EOE(exceptions_handler);
    ESTO();
    LOG("this message is logged after eob finished");
    LOG("this message is logged after eoe finished");
    BC();
}
```

void ERUN (void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

void ESTEP (void)

see ODBGScript documentation for details.

void ESTI (void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

void EVAL (void)

deprecated, Use javascript built-in string/data concatenation.

```
// Instead of this:
/*
var addr
...
MOV addr, [esp]
EVAL "mov [{eip}], {addr}"
WTRA "patcher.txt", $RESULT, "\r\n"
*/
// Use this:
```

```
var addr;  
//...  
WRTA("patcher.txt", "mov [" + eip + "], "+ DD(esp  
.v), "\r\n");
```

void EXEC (void)

Returns:

Todo:

Add EXEC API support.

```
ulong FILL ( ulong addr,  
             int    len,  
             char * value  
         )
```

see ODBGScript documentation for details.

```
FILL(0x40115C, 30, 0x89);
```

```
ulong FIND ( ulong addr,  
             char * what  
         )
```

search for a pattern of bytes starting from address addr

Parameters:

addr begining of the search

what either a numerique, a string bytes representation or an array of bytes;

Returns:

first address of matching result starting from addr;

search for a pattern of bytes starting from address `addr`

```
while (FIND(addr, "68????????81??24")) {
    tmp = DD($RESULT + 1);

    if (DB($RESULT + 6) == 4) {
        // letting db() set $RESULT will
biased next operations
        MOV($RESULT + 1, tmp);
        MOV($RESULT + 5, "90909090909090"
);
    }
    addr += 12;
}
```

```
ulong FINDO ( ulong addr,
              char * what,
              int   blocklen,
              func  callback,
              ...
)
```

stands for find-do, it allows you to call a callback function each time the pattern is found.

Parameters:

- addr** address to begin search from.
- what** javascript array of bytes, hex string representation of pattern to search for or a dword.
- blocklen** length of block of search starting from address `addr`. set to zero to search from `addr` to the end of memory page.
- callback** javascript function to call each time the pattern `what` is found. This callback must return true to increment counter of your matched conditions, else it must return false.

... arguments to be passed to callback in addition to the address of next found pattern.

Returns:

number of matched conditions over all found pattern/value/array.

this is a super-repl function. It allows you to call a callback function each time the pattern is found, thus, you can do testing/address manipulation/more checking before replacing the pattern (or doing what ever you want). It's also a good replacement to calling find() in loops, calling findo can be reusable and the same callback can be called from different places of the script. Please, have in mind that any failure of any function called from the callback will cause the fail of the callback.

```
function jsmain() {
    var min = GMEMI(eip.v, MEMORYBASE);
    var max = min + GMEMI(eip.v, MEMORIESIZE)
- 1;

    LOG(FINDO(eip.v, "E8??????00", 0, call_callback));
    LOG(FINDO(eip.v, "E9??????60", 0, jump_callback, min, max));
}

function call_callback(addr) {
    var tmpdest = addr;
    tmpdest = GCI(tmpdest, FINALDEST);
    if (tmpdest > 0x60000000) {
        LBL(addr, "this is intermodular call"); // set label at address addr
        return true; // return true to count this
    }
    return false; // else return false
}
```

```
function jump_callback(addr, minaddr, maxaddr) {
    var tmpdest = addr;
    tmpdest = GCI(tmpdest, FINALDEST);
    if (tmpdest > maxaddr || tmpdest < minaddr) {
        LBL(addr, "this is intermodular jump"); // set label at address addr
        return true;
    }
    return false;
}
```

void FREE(ulong addr)

see ODBGScript documentation for details.

```
var mem = ALLOC(0x1000);
if(mem) {
    MOV(mem, "558BEC81C400FEFFFF68005040008D8500F
EFFFF50E80F2B000068545740008D8500FEFFFF50E8F22A00
008D8500FEFFFF50FF7508E8732B0000C9C20400");
}
LOG(mem);
// FREE(mem); // if you don't wanna keep allocated memory mem
```

ulong GAPI(ulong addr)

Parameters:

addr

Returns:

Todo:

Add GAPI API support.

void GBPM()

Returns:

Todo:

Add GBPM API support.

void GBPR()

see ODBGScript documentation for details.

```
EST0();
LOG("break point reason is: ", GBPR().toString(16
));
```

```
ulong GCI( ulong addr,
           int     info
         )
```

stands for get command information

Parameters:

addr pointer to command
info information to get

Returns:

what was requested

See also:

[e_cmdinfo](#)

stands for get command information. info is a member of [e_cmdinfo](#)

```
var txt = GCI(0x402020, COMMAND);
var dest = GCI(eip.v, FINALDEST);
```

```
LOG("The asm text representation is: '", txt, "'  
and the final destination if jumps/calls cascade  
from eip is: ", dest);
```

ulong GFO (ulong addr)

see ODBGScript documentation for details.

```
LOG("offset of va 401000 is: ", GFO(0x401000));
```

ulong GMA (char * name, int info)

returns information about module named *name*

Parameters:

name name of module
info information to get

Returns:

See also:

[t_moduleinfo](#)

returns information about module named *name*

```
LOG(GMA("kernel32", MODULEBASE));  
LOG(GMA("user32", MODULEBASE));
```

ulong GMEMI (ulong addr, int info)

see ODBGScript documentation for details. see [e_memoryinfo](#)

```
LOG(GMEMI(eip.v, MEMORYBASE));
```

```
ulong GMEXP( ulong moduleaddr,  
             int     info  
           );
```

Parameters:

moduleaddr
 info
 num

Returns:

Todo:

Add GMEXP API support.

```
char* GN( ulong addr )
```

Parameters:

addr

Returns:

Todo:

Add GN API support.

```
void GO( ulong addr )
```

see ODBGScript documentation for details.

```
AI();  
AO();  
ERUN();  
ESTI();
```

```
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

```
ulong GPA( char * proc,
            char * lib,
            bool bKeepInMem
        )
```

see ODBGScript documentation for details.

```
if(GPA("GetModuleHandleA", "kernel32.dll")) {
    LOG($RESULT);
    LOG($RESULT_1);
    LOG($RESULT_2);
}
```

```
ulong GPI( info )
```

see ODBGScript documentation for details.

See also:

[e_processinfo](#)

```
LOG("mainbase:", GPI(MAINBASE));
LOG("processname:", GPI(PROPERTYNAME));
LOG("exefilename:", GPI(EXEFILENAME));
```

```
ulong GREF( )
```

Parameters:

line

Returns:

Todo:

Add GREF API support.

ulong GRO (ulong addr)

Parameters:

addr

Returns:

Todo:

Add GRO API support.

ulong GSL ()

see ODBGScript documentation for details.

See also:

e_origin

```
GSL(CPUDASM); // to get selection limits in  
asm window  
GSL(CPUDUMP); // to get selection limits in  
dump window  
GSL(CPUSTACK); // to get selection limits in  
stack window
```

char* GSTR (ulong addr[, int len])

return string of length *len* from memory *addr*

Parameters:

addr pointer to memory of string

len size of characters to copy

Returns:

the string

return string of length *len* from memory *addr*. If called with *len*, it copies zeros also<> it sets only \$RESULT with the string and \$RESULT_1 with its length.

```
// Example 1
var str = GSTR(0x405000, 0xE6);
var my_array = str.split('\0');
for(i = 0; i < my_array.length; i++)
    /* you could also do:
       * LOG("The length of string:'", my_array
       [i], "' is:", my_array[i].length); */
    LOG("The length of string:'", my_array[i]
, "' is:", LEN(my_array[i)));


// Example 2
var str = GSTR(0x405000);
LOG("Using LEN(), the length of string:'", str, "
' is:", LEN(str));
LOG("Using $RESULT_1, the length of string:'", st
r, "' is:", $RESULT_1);
LOG("Using .length, the length of string:'", str,
"' is:", str.length);
```

uchar* GSTRW (ulong addr[, int len])

return unicode string of length *len* from memory *addr*

Parameters:

addr pointer to memory of string
len size of unicode characters to copy

Returns:

the unicode string

return unicode string of length *len* from memory *addr*. If called with *len*, it copies zeros also it sets only \$RESULT with the unicode string and \$RESULT_1 with its length.

```
// Example 1
var str = GSTRU(0x405000, 0xE6);
var my_array = str.split('\0');
for(i = 0; i < my_array.length; i++)
    /* you could also do:
       * LOG("The length of string:'", my_array
       [i], "' is:", my_array[i].length); */
       LOG("The length of string:'", my_array[i]
, "' is:", LEN(my_array[i)));


// Example 2
var str = GSTRU(0x405000);
LOG("Using LEN(), the length of string:'", str, "
' is:", LEN(str));
LOG("Using $RESULT_1, the length of string:'", st
r, "' is:", $RESULT_1);
LOG("Using .length, the length of string:'", str,
"' is:", str.length);
```

```
void KEY( int   vkcode,
          bool  shift = false,
          bool  ctrl = false,
          int   origin = PM_DISASM
)
```

Parameters:

vkcode
shift
ctrl

Returns:

Todo:

Fix some issues with this api.

```
KEY('C'.charCodeAt(0), false, true, PM_DISASM); /  
/ copy current line from asm window
```

```
void LBL(ulong addr,  
         char * text  
     )
```

see ODBGScript documentation for details.

```
LBL(ecx.v, "this is lbl test");
```

```
void LC()
```

see ODBGScript documentation for details.

```
LC();
```

```
ulong LEN(char * str)
```

see ODBGScript documentation for details. Note that either len and LEN are reserved names in ODBJScript.

```
var str = GSTR(0x405000, 0xE6);  
var my_array = str.split('\0');  
for(i = 0; i < my_array.length; i++)
```

```
// note that either len and LEN are reserved  
names in ODBJScript.  
    LOG("The length of string:'", my_array[i]  
, "' is:", LEN(my_array[i)));
```

```
ulong LM( ulong addr,  
          int size,  
          char * filename  
    )
```

Parameters:

addr

size

filename

Returns:

Todo:

Add LM API support.

```
ulong LOADLIB ( char * dllname )
```

Parameters:

dllname

Returns:

Todo:

Add LOADLIB API support.

```
void LOG ( ... )
```

logs any data to log window.

Parameters:

... arguments you wanna log.

logs any data to log window. can be DWORD, doubles, floats, strings, arrays, function names ...

```
LOG("data at: ", eax.v, " must be: ", [0x68, 0x40  
, 0x90], ", or equal to: ", ebx);
```

```
void LOGBUF( ulong ip,  
             size_t size,  
             int    nepl = 32,  
             char * separator = ""  
           )
```

logs *size* bytes of memory beginning from address *ip*.

Parameters:

ip	pbuffer start address of dump
size	size of dumped bytes to be logged
nepl	elemperline number of elements per line
separator	separator between bytes

Returns:

void

logs *size* bytes of memory beginning from address *ip* with options:
nepl element per line and separating element by *separator*.

```
LOGBUF(0x401000, 45, 100, ':' );
```

```
ulong MEMCPY( ulong dest,  
              ulong src,  
              int   size  
            )
```

Parameters:

dest
src
size

Returns:

Todo:

Add MEMCPY API support.

```
MEMCPY(0x405024, 0x4011AC, 0x30);
```

```
ulong MOV(ulong addr,  
         char * what,  
         int    n = lenght_of_what  
)
```

writes *n* or all element of *what* to address *addr*

Parameters:

addr pointer to address to write to.
what a string representation of binary data, or a js array of char or a DWORD
n in case you want to write just a part from *what*, *n* is the needed number of elements.

Returns:

number of bytes wrote.

writes *n* or all element of *what* to address *addr*. Doesn't set \$RESULT

```
var my_array = [0x90, 0x90, 0x90, 0x90, 0x90, 0x68];  
MOV(eip.v, my_array);  
MOV(edi.v, ecx.v)  
MOV(0x40116B, "90??3244??????7BA5F3EE??3"); // equivalent to asm like mov [0x4100EA], #90??3244??
```

????7BA5F3EE??3#

```
void MOVB ( object * objaddr,  
            object * odjdata  
        )
```

batch mov

Parameters:

objaddr javascript array containing address
odjdata javascript array of binary data (DWORD/array/string representation of binary data) to be written. Each element is written to corresponding address from objaddr.

movb stands for batch mov, i.e. you can write to a debugger memory page tons of data at once. As access of debugger memory is a time consuming operation, this function can accelerate the process hundreds of times as it reads the whole page, modifies it and then rewrite it back one time.

See also:

[ASMB](#)

```
function jsmain() {  
    var my_array = [[], []]; // array composed by 2 arrays  
  
    // populate address object  
    my_array[0].push(0x401000);  
    my_array[0].push(0x401004);  
    my_array[0].push(0x401008);  
    my_array[0].push(0x40100C);  
    my_array[0].push(0x401010);  
    my_array[0].push(0x401014);  
    my_array[0].push(0x401018);  
    my_array[0].push(0x40101C);
```

```

my_array[0].push(0x401020);
my_array[0].push(0x401024);

// populate data object
my_array[1].push(0);
my_array[1].push(1);
my_array[1].push(2);
my_array[1].push(3);
my_array[1].push(4);
my_array[1].push(5);
my_array[1].push(6);
my_array[1].push(7);
my_array[1].push(8);
my_array[1].push(9);

tc = TICK();
for (var i = 0; i < my_array[0].length; i++)
    MOV(my_array[0][i], my_array[1][i]);
    LOG("calling mov in loop took:", TICK() - tc, " ms");

tc = TICK();
MOVB(my_array[0], my_array[1]);
LOG("calling movb took:", TICK() - tc
, " ms");
}

```

```

ulong MOVS ( ulong addr,
            char * str,
            int      n = lenght_of_what
)

```

writes *n* or all characters of string *str* to address *addr*

Parameters:

addr pointer to address to write to.

what a string.

n in case you want to write just a part from *what*, n is the needed number of characters.

Returns:

number of characters wrote.

writes *n* or all characters of string *str* to address *addr*. Doesn't set \$RESULT

```
MOVS(GSL(CPUDUMP), "string to move to current selected address of cump windows");
```

void MSG (void)

see ODBGScript documentation for details.

```
MSG("No pattern found? Will abort script!");
```

void MSGYN (void)

see ODBGScript documentation for details.

```
var response = MSGYN("Is DebugBlocker enabled");
if (response == IDCANCEL)
    return;
else if (response == IDNO)
    delta = 0x20;
else // IDYES
    delta = 0x520;

LOG(delta);
```

ulong NAMES (ulong addr)

Parameters:

addr

Returns:

Todo:

Add NAMES API support.

ulong OLLY (int info)

Parameters:

info

Returns:

Todo:

Add OLLY API support.

ulong OPCODE (ulong addr)

see ODBGScript documentation for details.

```
OPCODE(0x401000);  
LOG($RESULT_1);
```

```
void OPENDUMP ( ulong addr,  
                ulong base,  
                ulong size,  
                int    type = Hex/ASCII  
            )
```

Todo:

Fix OPENDUMP API close after open.

See also:

[e_dumptype](#)

```
OPENDUMP(0x40115C);
// or
OPENDUMP(0x40115C, 0x401000, 0x200, DU_HEXTEXT
);
```

void OPENTRACE()

see ODBGScript documentation for details.

```
OPENTRACE();
```

void PAUSE(void)

Returns:

Todo:

Add PAUSE API support.

ulong POP()

see ODBGScript documentation for details.

Todo:

fix pop something

```
// pop(something) isn't supported yet
POP();
```

void POPA(void)

Returns:

Todo:

Add POPA API support.

ulong PREOP (ulong addr)

Get asm command line address just before specified address.

Parameters:

addr current position

Returns:

address of command 1 step back from address *addr*. if *addr* == PREOP(*addr*), function returns 0 to facilitate testing.

Get asm command line address just before specified address.

```
/* preop(addr);
 * Normally ollydbg returns the same address if a
 * ddr is the first address of a
 * memory page. But to keep things simple, I cha-
 * nged that to let you test the
 * function without saving the return address in
 * a var. see this example:
 */
while (PREOP(jmpdest)) {
    if (DW($RESULT) != 0)
        break;
    jmpdest = $RESULT;
}
```

void PUSH (ulong dw)

see ODBGScript documentation for details.

```
PUSH(eax.v);  
PUSH(0x401000);
```

void PUSHA(void)

Returns:

Todo:

Add PUSHA API support.

```
char* READSTR ( char * str,  
                 int      len  
               )
```

see ODBGScript documentation for details.

void REFRESH (bool onoff)

sets on/off refreshing ollydbg MDI windows.

Parameters:

onoff if true, plugin will call Broadcast function to let you see visual changes like breakpoint, memory changes ... but this slows down execution a little bit.

Returns:

nothing

sets on/off refreshing ollydbg MDI windows. thus, if you want to give the script a hand, call it with false in the beginning of your script.

```
REFRESH(true);
```

```
ulong REPL ( ulong addr,
            char * find,
            char * repl,
            int     len
        )
```

see ODBGScript documentation for details.

```
LOG(REPL(0x40115C, "0000", "9090", 0x1000));
```

```
ulong REV ( ulong what )
```

see ODBGScript documentation for details.

```
LOG(REV(0x11223344));
```

```
ulong ROL ( ulong      op,
            ulong ulong count
        )
```

see ODBGScript documentation for details.

```
LOG(ROL(0x11223344, 8));
LOG(ROR(0x11223344, 8));
```

```
ulong ROR ( ulong op,
            ulong count
        )
```

see ODBGScript documentation for details.

```
LOG(ROL(0x11223344, 8));
LOG(ROR(0x11223344, 8));
```

void RTR (void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

void RTU (void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

void RUN(void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

ulong SETOPTION (DWORD mask)

sets/unsets exception to be ignored by ollydbg.

Parameters:

mask for exceptions to set/unset

See also:

[**e_exceptmask**](#)

sets/unsets exception to be ignored by ollydbg. can do it manually or internally.

```
SETOPTION(IGNORE_ALL);
// or
SETOPTION(IGNORE_ACCESS_KER32|IGNORE_INT3);
// or
SETOPTION(IGNORE_MANUALLY);
```

void STI(void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

void STO(void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

void STR (var)

Parameters:

var

Returns:

Todo:

Add STR API support.

void TC ()

see ODBGScript documentation for details.

TC();

void TI (void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

ulong TICK (void)

returns value of GetTickCount()

returns value of GetTickCount() which retrieves the number of milliseconds that have elapsed since Windows was started.

```
void TICND ( char * cond,  
             ulong in0,  
             ulong in1,  
             ulong out0,  
             ulong out1  
         )
```

Traces into calls until cond is true and/or on EIP in range [in0-in1] and/or out of range [out0-out1].

Parameters:

cond condition to stop execution if met
in0,in1 - 'in range' request. Run trace will pause if EIP is in this range (in1 not included).
out0,out1 - 'out of range' request. Run trace will pause if EIP is outside this range or equals to out1.

Returns:

nothing

this javascript version of function accepts either 1, 3 or 5 arguments. in case of one argument, it will be treated as string condition. In case of 3, they will be treated as string cond,in0,int1, else it will be the full version of function.

```
TICND( "eax==66D00000" );  
// or  
TICND( "", 0x40117C, 0x401188 ); // pause if eip is
```

```
    in range [0x40117C-0x401188]
// or
TICND("eax==66D00000", 0, 0, 0x40117C, 0x401188);
// pause if either eax is equal to 66D00000 or ei
p is out of range [0x40117C-0x401188]
```

void TO(void)

see ODBGScript documentation for details.

```
AI();
AO();
ERUN();
ESTI();
GO(0x401400);
RTR();
RTU();
RUN();
STI();
STO();
TI();
TO();
ESTEP();
```

void TOCND(char * cond)

Traces over calls until cond is true and/or on EIP in range [in0-in1] and/or out of range [out0-out1].

Parameters:

- cond** condition to stop execution if met
- in0,in1** - 'in range' request. Run trace will pause if EIP is in this range (in1 not included).
- out0,out1** - 'out of range' request. Run trace will pause if EIP is outside this range or equals to out1.

Returns:

nothing

this javascript version of function accepts either 1, 3 or 5 arguments.
in case of one argument, it will be treated as string condition. In case
of 3, they will be treated as string cond,in0,int1, else it will be the full
version of function.

```
TOCND("eax==66D00000");
// or
TOCND("", 0x40117C, 0x401188);
```

```
ulong WRT ( char * file,
            char * data
        )
```

see ODBGScript documentation for details.

```
var my_array = new Array();
my_array[0] = 0x90;
my_array[1] = 0x10;
my_array[2] = 0x11;
my_array[3] = 0x12;
my_array[4] = 0x13;
my_array[5] = 0x14;
my_array[6] = 0x15;
my_array[7] = 0x16;
my_array[8] = 0x17;
my_array[9] = 0x18;
WRT("stoleb.bytes", "; This is wrta/wrt example");
;
WRTA("stoleb.bytes", my_array, "");
WRTA("stoleb.bytes", 0x68909090);
```

```
ulong WRTA ( char * file,
            char * data[, char *separator]
```

)

see ODBGScript documentation for details.

```
var my_array = new Array();
my_array[0] = 0x90;
my_array[1] = 0x10;
my_array[2] = 0x11;
my_array[3] = 0x12;
my_array[4] = 0x13;
my_array[5] = 0x14;
my_array[6] = 0x15;
my_array[7] = 0x16;
my_array[8] = 0x17;
my_array[9] = 0x18;
WRT("stoleb.bytes", "; This is wrta/wrt example")
;
WRTA("stoleb.bytes", my_array, "");
WRTA("stoleb.bytes", 0x68909090);
```

```
ulong XCHG ( object * regdest,
              object * regsrc
            )
```

exchanges values of 2 registers of same size

Parameters:

regdest is first register

regsrc is second register

exchanges values of 2 registers of same size. Note that you dont have to use .v to refer to register value, it's done internally.

```
XCHG(al, cl);
// or
XCHG(eax, edi);
```

Generated on Fri Jan 27 2012 17:54:31 for ODBJscript by [doxygen](#) 1.7.5.1

ODBJscript

javascript plugin

Main Page	Related Pages	Files																	
File List	Globals																		
All	Functions	Enumerations	Enumerator																
a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	r	s	t	w	x

Here is a list of all functions, variables, defines, enums, and typedefs with links to the files they belong to:

- a -

- AI() : [ODBJScript.h](#)
- ALLOC() : [ODBJScript.h](#)
- AN() : [ODBJScript.h](#)
- AO() : [ODBJScript.h](#)
- ASK() : [ODBJScript.h](#)
- ASMB() : [ODBJScript.h](#)

- b -

- BACKUP() : [ODBJScript.h](#)
- BC() : [ODBJScript.h](#)
- BD() : [ODBJScript.h](#)
- BE() : [ODBJScript.h](#)
- BP() : [ODBJScript.h](#)
- BPCND() : [ODBJScript.h](#)
- BPD() : [ODBJScript.h](#)
- BPGOTO() : [ODBJScript.h](#)
- BPHWC() : [ODBJScript.h](#)
- BPHWS() : [ODBJScript.h](#)
- BPL() : [ODBJScript.h](#)
- BPLCND() : [ODBJScript.h](#)
- BPMC() : [ODBJScript.h](#)
- BPRM() : [ODBJScript.h](#)
- BPWM() : [ODBJScript.h](#)

- BPX() : **ODBJScript.h**
- BUF() : **ODBJScript.h**

- c -

- CLOSE() : **ODBJScript.h**
- CMT() : **ODBJScript.h**
- COB() : **ODBJScript.h**
- CODEBASE : **ODBJScript.h**
- CODESIZE : **ODBJScript.h**
- COE() : **ODBJScript.h**
- COMMAND : **ODBJScript.h**
- CONDITION : **ODBJScript.h**
- CPUDASM : **ODBJScript.h**
- CPUDUMP : **ODBJScript.h**
- CPUSTACK : **ODBJScript.h**
- CURRENTDIR : **ODBJScript.h**

- d -

- DATABASE : **ODBJScript.h**
- DB() : **ODBJScript.h**
- DBH() : **ODBJScript.h**
- DBS() : **ODBJScript.h**
- DD() : **ODBJScript.h**
- DESTINATION : **ODBJScript.h**
- DM() : **ODBJScript.h**
- DMA() : **ODBJScript.h**
- DPE() : **ODBJScript.h**
- DU_ADDR : **ODBJScript.h**
- DU_ADRASC : **ODBJScript.h**
- DUADRUNI : **ODBJScript.h**
- DU_DISASM : **ODBJScript.h**
- DU_FLOAT : **ODBJScript.h**
- DU_HEXTTEXT : **ODBJScript.h**
- DU_HEXUNI : **ODBJScript.h**
- DU_IHEX : **ODBJScript.h**
- DU_INT : **ODBJScript.h**
- DU_TEXT : **ODBJScript.h**

- DU_UINT : **ODBJScript.h**
- DU_UNICODE : **ODBJScript.h**
- DW() : **ODBJScript.h**

- e -

- e_cmdinfo : **ODBJScript.h**
- e_dumptype : **ODBJScript.h**
- e_exceptmask : **ODBJScript.h**
- e_memoryinfo : **ODBJScript.h**
- e_origin : **ODBJScript.h**
- e_processinfo : **ODBJScript.h**
- EDATASIZE : **ODBJScript.h**
- EDATATABLE : **ODBJScript.h**
- ENDE() : **ODBJScript.h**
- ENDSEARCH() : **ODBJScript.h**
- ENTRY : **ODBJScript.h**
- EOB() : **ODBJScript.h**
- EOE() : **ODBJScript.h**
- ERUN() : **ODBJScript.h**
- ESTEP() : **ODBJScript.h**
- ESTI() : **ODBJScript.h**
- EVAL() : **ODBJScript.h**
- EXEC() : **ODBJScript.h**
- EXEFILENAME : **ODBJScript.h**

- f -

- FILL() : **ODBJScript.h**
- FINALDEST : **ODBJScript.h**
- FIND() : **ODBJScript.h**
- FINDO() : **ODBJScript.h**
- FREE() : **ODBJScript.h**

- g -

- GAPI() : **ODBJScript.h**
- GBPM() : **ODBJScript.h**
- GBPR() : **ODBJScript.h**

- GCI() : **ODBJScript.h**
- GFO() : **ODBJScript.h**
- GMA() : **ODBJScript.h**
- GMEMI() : **ODBJScript.h**
- GMEXP() : **ODBJScript.h**
- GN() : **ODBJScript.h**
- GO() : **ODBJScript.h**
- GPA() : **ODBJScript.h**
- GPI() : **ODBJScript.h**
- GREF() : **ODBJScript.h**
- GRO() : **ODBJScript.h**
- GSL() : **ODBJScript.h**
- GSTR() : **ODBJScript.h**
- GSTRW() : **ODBJScript.h**

- h -

- HMAINTHREAD : **ODBJScript.h**
- HPROCESS : **ODBJScript.h**

- i -

- IDATATABLE : **ODBJScript.h**
- IGNORE_ACCESS : **ODBJScript.h**
- IGNORE_ACCESS_KER32 : **ODBJScript.h**
- IGNORE_ALL : **ODBJScript.h**
- IGNORE_CUSTOM_EXCEPT : **ODBJScript.h**
- IGNORE_DIV_ZERO : **ODBJScript.h**
- IGNORE_FPU_EXCEPT : **ODBJScript.h**
- IGNORE_ILLEGAL_INST : **ODBJScript.h**
- IGNORE_INT3 : **ODBJScript.h**
- IGNORE_MANUALLY : **ODBJScript.h**
- IGNORE_NONE : **ODBJScript.h**
- IGNORE_TRAP : **ODBJScript.h**

- k -

- KEY() : **ODBJScript.h**

- l -

- LBL() : **ODBJScript.h**
- LC() : **ODBJScript.h**
- LEN() : **ODBJScript.h**
- LM() : **ODBJScript.h**
- LOADLIB() : **ODBJScript.h**
- LOG() : **ODBJScript.h**
- LOGBUF() : **ODBJScript.h**

- m -

- MAINBASE : **ODBJScript.h**
- MAINTHREADID : **ODBJScript.h**
- MEMCPY() : **ODBJScript.h**
- MEMORYBASE : **ODBJScript.h**
- MEMORYOWNER : **ODBJScript.h**
- MEMORIESIZE : **ODBJScript.h**
- MODULEBASE : **ODBJScript.h**
- MODULESIZE : **ODBJScript.h**
- MOV() : **ODBJScript.h**
- MOVB() : **ODBJScript.h**
- MOVS() : **ODBJScript.h**
- MSG() : **ODBJScript.h**
- MSGYN() : **ODBJScript.h**

- n -

- NAMES() : **ODBJScript.h**
- NSECT : **ODBJScript.h**

- o -

- OLLY() : **ODBJScript.h**
- OPCODE() : **ODBJScript.h**
- OPENDUMP() : **ODBJScript.h**
- OPENTRACE() : **ODBJScript.h**

- p -

- PAUSE() : **ODBJScript.h**
- POP() : **ODBJScript.h**

- POPA() : **ODBJScript.h**
- PREOP() : **ODBJScript.h**
- PROCESSID : **ODBJScript.h**
- PROCESSNAME : **ODBJScript.h**
- PUSH() : **ODBJScript.h**
- PUSHA() : **ODBJScript.h**

- r -

- READSTR() : **ODBJScript.h**
- REFRESH() : **ODBJScript.h**
- RELOCSIZE : **ODBJScript.h**
- RELOCTABLE : **ODBJScript.h**
- REPL() : **ODBJScript.h**
- RESBASE : **ODBJScript.h**
- RESSIZE : **ODBJScript.h**
- REV() : **ODBJScript.h**
- ROL() : **ODBJScript.h**
- ROR() : **ODBJScript.h**
- RTR() : **ODBJScript.h**
- RTU() : **ODBJScript.h**
- RUN() : **ODBJScript.h**

- s -

- SETOPTION() : **ODBJScript.h**
- SIZE : **ODBJScript.h**
- STI() : **ODBJScript.h**
- STO() : **ODBJScript.h**
- STR() : **ODBJScript.h**
- SYSTEMDIR : **ODBJScript.h**

- t -

- t_moduleinfo : **ODBJScript.h**
- TC() : **ODBJScript.h**
- TI() : **ODBJScript.h**
- TICK() : **ODBJScript.h**
- TICND() : **ODBJScript.h**

- TO() : **ODBJScript.h**
- TOCND() : **ODBJScript.h**
- TYPE : **ODBJScript.h**

- W -

- WRT() : **ODBJScript.h**
- WRTA() : **ODBJScript.h**

- X -

- XCHG() : **ODBJScript.h**

Generated on Fri Jan 27 2012 17:54:31 for ODBJscript by  1.7.5.1

ODBJscript

javascript plugin

Main Page

Related Pages

Files

File List

Globals

ODBJScript.h

Go to the documentation of this file.

```
00001 #ifndef __ODBJ_DOCUMENTATION_H__
00002 #define __ODBJ_DOCUMENTATION_H__
00003
00004
00008 enum e_exceptmask {
00009     IGNORE_NONE             = 0x00000000,
00010     IGNORE_ACCESS_KER32      = 0x00000001,
00011     IGNORE_INT3              = 0x00000002,
00012     IGNORE_TRAP              = 0x00000004,
00013     IGNORE_ACCESS             = 0x00000010,
00014     IGNORE_DIV_ZERO           = 0x00000020,
00015     IGNORE_ILLEGAL_INST       = 0x00000040,
00016     IGNORE_FPU_EXCEPT         = 0x00000100,
00017     IGNORE_CUSTOM_EXCEPT      = 0x00000200,
00018     IGNORE_MANUALLY           = 0xFFFFFFFF,
00019     IGNORE_ALL                = IGNORE_ACCESS_
KER32|IGNORE_INT3|IGNORE_TRAP|IGNORE_ACCESS|IGNORE_
_DIV_ZERO|IGNORE_ILLEGAL_INST|IGNORE_FPU_EXCEPT|IG
NORE_CUSTOM_EXCEPT
00020 };
00021
00022 enum e_memoryinfo {
00023     MEMORYBASE,
00024     MEMORIESIZE,
00025     MEMORYOWNER
00026 };
```

```
00027
00028
00029 enum t_moduleinfo {
00030     MODULEBASE,
00031     MODULESIZE,
00032     CODEBASE ,
00033     CODESIZE ,
00034     ENTRY ,
00035     NSECT ,
00036     DATABASE ,
00037     EDATATABASE,
00038     EDATASIZE ,
00039     IDATATABASE,
00040     RESBASE ,
00041     RESSIZE ,
00042     RELOCTABLE,
00043     RELOCSIZE
00044 };
00045
00046 enum e_cmdinfo {
00047     COMMAND ,
00048     CONDITION ,
00049     DESTINATION,
00050     FINALDEST ,
00051     SIZE ,
00052     TYPE ,
00053 };
00054
00055 enum e_dumptype {
00056     DU_HEXTTEXT = 0x01000L,
00057     DU_TEXT      = 0x02000L,
00058     DU_UNICODE   = 0x03000L,
00059     DU_INT       = 0x04000L,
00060     DU_UINT      = 0x05000L,
00061     DU_IHEX      = 0x06000L,
00062     DU_FLOAT     = 0x07000L,
00063     DU_ADDR      = 0x08000L,
```

```
00064     DU_DISASM    =      0x09000L,
00065     DU_HEXUNI   =      0x0A000L,
00066     DU_ADRASC    =      0x0B000L,
00067     DU_ADRUNI   =      0x0C000L
00068 };
00069
00070 enum e_processinfo {
00071     HPROCESS = 20,
00072     PROCESSID ,
00073     HMAINTHREAD ,
00074     MAINTHREADID,
00075     MAINBASE ,
00076     PROCESSNAME ,
00077     EXEFILENAME ,
00078     CURRENTDIR ,
00079     SYSTEMDIR
00080 };
00081
00082 enum e_origin {
00083     CPUDASM,
00084     CPUDUMP,
00085     CPUSTACK
00086 };
00087
00095 ulong DD(ulong addr);
00103 ulong DW(ulong addr);
00111 ulong DB(ulong addr);
00121 ulong SETOPTION(DWORD mask);
00126 void EVAL(void);
00131 void MSGYN(void);
00136 void MSG(void);
00145 void REFRESH(bool onoff);
00157 void LOGBUF(ulong ip, size_t size, int nepl=
 32, char* separator = "");
00165 void LOG(...);
00176 void TICND(char* cond, ulong in0, ulong in1, u
long out0, ulong out1);
```

```
00188 void BPGOTO(ulong addr, FUNC* proc);
00197 void EOB(FUNC* proc);
00206 void EOE(FUNC* proc);
00211 void COB();
00216 void COE();
00226 void BPX(char* procname, DWORD addr = 0);
00237 void BPD(char* procname);
00247 void GBPM();
00257 ulong FIND(ulong addr, char* what);
00268 ulong MOV(ulong addr, char* what, int n = length_of_what);
00279 ulong MOVS(ulong addr, char* str, int n = length_of_what);
00294 ulong FINDO(ulong addr, char* what, int bloc_klen, func callback, ...);
00303 ulong PREOP(ulong addr);
00314 ulong GCI(ulong addr, int info);
00327 void KEY(int vkcode, bool shift = false, bool ctrl = false, int origin = PM_DISASM);
00333 void OPENDUMP(ulong addr, ulong base, ulong size, int type = Hex/ASCII){}
00344 ulong BUF(var);
00355 ulong CLOSE(window);
00365 ulong DBH(void);
00375 ulong DBS(void);
00387 ulong DPE(char* filename, ulong ep);
00397 ulong ENDE(void);
00407 void ENDSEARCH(void);
00417 void EXEC(void);
00428 ulong GAPI(ulong addr);
00438 ulong GMA(char* name, int info);
00451 ulong GMEXP(ulong moduleaddr, int info, [int num]);
00461 char* GN(ulong addr);
00472 ulong GREF([int line]);
00483 ulong GRO(ulong addr);
00493 char* GSTR(ulong addr[, int len]);
```

```
00503 uchar* GSTRW(ulong addr[, int len]);
00516 ulong LM(ulong addr, int size, char* filename);
00527 ulong LOADLIB(char* dllname);
00540 ulong MEMCPY(ulong dest, ulong src, int size);
00551 ulong NAMES(ulong addr);
00562 ulong OLLY(int info);
00572 void PAUSE(void);
00582 void POPA(void);
00592 void PUSHA(void);
00603 void STR(var);
00608 void AI(void);
00613 void AO(void);
00618 void ERUN(void);
00623 void ESTI(void);
00628 void GO(ulong addr);
00633 void RTR(void);
00638 void RTU(void);
00643 void RUN(void);
00648 void STI(void);
00653 void STO(void);
00658 void TI(void);
00663 void TO(void);
00674 void TOCND(char* cond);
00679 void CMT(char* comment);
00684 void AN(ulong addr);
00689 void BC(ulong addr);
00694 void BD(ulong addr);
00699 void BE(ulong addr);
00704 void BP(ulong addr);
00709 void BPCND(ulong addr, char* cond);
00714 void BPLCND(ulong addr, char* cond, char* expr);
00719 void BPWM(ulong addr, int size);
00724 void BPHWS(ulong addr, char* mode = "x", int size = 1);
```

```
00729 void BPHWC(ulong addr);
00734 void BPL(ulong addr, char* expr);
00739 void BPMC();
00744 void BPRM(ulong addr, int size);
00749 void GBPR();
00755 ulong GMEMI(ulong addr, int info);
00760 ulong GPA(char* proc, char* lib, bool bKeepIn
Mem);
00765 ulong REPL(ulong addr, char* find, char* rep
l, int len);
00770 ulong WRT(char* file, char* data);
00775 ulong WRTA(char* file, char* data[, char* se
parator]);
00780 ulong OPCODE(ulong addr);
00785 ulong FILL(ulong addr, int len, char* value)
;
00790 void LBL(ulong addr, char* text);
00795 void LC();
00800 void OPENTRACE();
00805 void TC();
00810 ulong ALLOC(int size);
00815 ulong ASK(char* question);
00820 void BACKUP(ulong addr);
00825 ulong DM(ulong addr, int size, file);
00830 ulong DMA(ulong addr, int size, file);
00835 void ESTEP(void);
00840 void FREE(ulong addr);
00845 ulong GFO(ulong addr);
00851 ulong GPI(info);
00857 ulong GSL([int where]);
00862 ulong LEN(char* str);
00868 ulong POP();
00873 void PUSH(ulong dw);
00878 char* READSTR(char* str, int len);
00883 ulong REV(ulong what);
00888 ulong ROL(ulong op, ulong ulong count);
00893 ulong ROR(ulong op, ulong count);
```

```
00900 ulong XCHG(object* regdest, object* regsrc);
00908 void MOVB(object* objaddr, object* odjdata);
00916 void ASMB(object* objaddr, object* odjdata);
00921 ulong TICK(void);
00922
00923
00924
00925 #endif // __ODBJ_DOCUMENTATION_H__
```

Generated on Fri Jan 27 2012 17:54:31 for ODBJscript by [doxygen](#) 1.7.5.1

ODBJscript

javascript plugin

Main Page	Related Pages	Files															
File List	Globals																
All	Functions	Enumerations	Enumerator														
a	b	c	d	e	f	g	k	l	m	n	o	p	r	s	t	w	x

- a -

- AI() : **ODBJScript.h**
- ALLOC() : **ODBJScript.h**
- AN() : **ODBJScript.h**
- AO() : **ODBJScript.h**
- ASK() : **ODBJScript.h**
- ASMB() : **ODBJScript.h**

- b -

- BACKUP() : **ODBJScript.h**
- BC() : **ODBJScript.h**
- BD() : **ODBJScript.h**
- BE() : **ODBJScript.h**
- BP() : **ODBJScript.h**
- BPCND() : **ODBJScript.h**
- BPD() : **ODBJScript.h**
- BPGOTO() : **ODBJScript.h**
- BPHWC() : **ODBJScript.h**
- BPHWS() : **ODBJScript.h**
- BPL() : **ODBJScript.h**
- BPLCND() : **ODBJScript.h**
- BPMC() : **ODBJScript.h**
- BPRM() : **ODBJScript.h**
- BPWM() : **ODBJScript.h**
- BPX() : **ODBJScript.h**

- BUF() : **ODBJScript.h**

- c -

- CLOSE() : **ODBJScript.h**
- CMT() : **ODBJScript.h**
- COB() : **ODBJScript.h**
- COE() : **ODBJScript.h**

- d -

- DB() : **ODBJScript.h**
- DBH() : **ODBJScript.h**
- DBS() : **ODBJScript.h**
- DD() : **ODBJScript.h**
- DM() : **ODBJScript.h**
- DMA() : **ODBJScript.h**
- DPE() : **ODBJScript.h**
- DW() : **ODBJScript.h**

- e -

- ENDE() : **ODBJScript.h**
- ENDSEARCH() : **ODBJScript.h**
- EOB() : **ODBJScript.h**
- EOE() : **ODBJScript.h**
- ERUN() : **ODBJScript.h**
- ESTEP() : **ODBJScript.h**
- ESTI() : **ODBJScript.h**
- EVAL() : **ODBJScript.h**
- EXEC() : **ODBJScript.h**

- f -

- FILL() : **ODBJScript.h**
- FIND() : **ODBJScript.h**
- FINDO() : **ODBJScript.h**
- FREE() : **ODBJScript.h**

- g -

- GAPI() : **ODBJScript.h**
- GBPM() : **ODBJScript.h**
- GBPR() : **ODBJScript.h**
- GCI() : **ODBJScript.h**
- GFO() : **ODBJScript.h**
- GMA() : **ODBJScript.h**
- GMEMI() : **ODBJScript.h**
- GMEXP() : **ODBJScript.h**
- GN() : **ODBJScript.h**
- GO() : **ODBJScript.h**
- GPA() : **ODBJScript.h**
- GPI() : **ODBJScript.h**
- GREF() : **ODBJScript.h**
- GRO() : **ODBJScript.h**
- GSL() : **ODBJScript.h**
- GSTR() : **ODBJScript.h**
- GSTRW() : **ODBJScript.h**

- k -

- KEY() : **ODBJScript.h**

- l -

- LBL() : **ODBJScript.h**
- LC() : **ODBJScript.h**
- LEN() : **ODBJScript.h**
- LM() : **ODBJScript.h**
- LOADLIB() : **ODBJScript.h**
- LOG() : **ODBJScript.h**
- LOGBUF() : **ODBJScript.h**

- m -

- MEMCPY() : **ODBJScript.h**
- MOV() : **ODBJScript.h**
- MOVB() : **ODBJScript.h**
- MOVS() : **ODBJScript.h**
- MSG() : **ODBJScript.h**

- MSGYN() : **ODBJScript.h**

- n -

- NAMES() : **ODBJScript.h**

- o -

- OLLY() : **ODBJScript.h**
- OPCODE() : **ODBJScript.h**
- OPENDUMP() : **ODBJScript.h**
- OPENTRACE() : **ODBJScript.h**

- p -

- PAUSE() : **ODBJScript.h**
- POP() : **ODBJScript.h**
- POPA() : **ODBJScript.h**
- PREOP() : **ODBJScript.h**
- PUSH() : **ODBJScript.h**
- PUSHA() : **ODBJScript.h**

- r -

- READSTR() : **ODBJScript.h**
- REFRESH() : **ODBJScript.h**
- REPL() : **ODBJScript.h**
- REV() : **ODBJScript.h**
- ROL() : **ODBJScript.h**
- ROR() : **ODBJScript.h**
- RTR() : **ODBJScript.h**
- RTU() : **ODBJScript.h**
- RUN() : **ODBJScript.h**

- s -

- SETOPTION() : **ODBJScript.h**
- STI() : **ODBJScript.h**
- STO() : **ODBJScript.h**
- STR() : **ODBJScript.h**

- t -

- TC() : **ODBJScript.h**
- TI() : **ODBJScript.h**
- TICK() : **ODBJScript.h**
- TICND() : **ODBJScript.h**
- TO() : **ODBJScript.h**
- TOCND() : **ODBJScript.h**

- w -

- WRT() : **ODBJScript.h**
- WRTA() : **ODBJScript.h**

- x -

- XCHG() : **ODBJScript.h**

ODBJavascript

javascript plugin

Main Page	Related Pages	Files
File List	Globals	
All	Functions	Enumerations
Enumerator		

- e_cmdinfo : [ODBJScript.h](#)
- e_dumptype : [ODBJScript.h](#)
- e_exceptmask : [ODBJScript.h](#)
- e_memoryinfo : [ODBJScript.h](#)
- e_origin : [ODBJScript.h](#)
- e_processinfo : [ODBJScript.h](#)
- t_moduleinfo : [ODBJScript.h](#)

ODBJavascript

javascript plugin

Main Page	Related Pages	Files		
File List	Globals			
All	Functions	Enumerations	Enumerator	
c d e f h i m n p r s t				

- c -

- CODEBASE : [ODBJScript.h](#)
- CODESIZE : [ODBJScript.h](#)
- COMMAND : [ODBJScript.h](#)
- CONDITION : [ODBJScript.h](#)
- CPUDASM : [ODBJScript.h](#)
- CPUDUMP : [ODBJScript.h](#)
- CPUSTACK : [ODBJScript.h](#)
- CURRENTDIR : [ODBJScript.h](#)

- d -

- DATABASE : [ODBJScript.h](#)
- DESTINATION : [ODBJScript.h](#)
- DU_ADDR : [ODBJScript.h](#)
- DUADRASC : [ODBJScript.h](#)
- DUADRUNI : [ODBJScript.h](#)
- DUDISASM : [ODBJScript.h](#)
- DUFLOAT : [ODBJScript.h](#)
- DUHEXTEXT : [ODBJScript.h](#)
- DUHEXUNI : [ODBJScript.h](#)
- DUIHEX : [ODBJScript.h](#)
- DUINT : [ODBJScript.h](#)
- DUTEXT : [ODBJScript.h](#)
- DUUINT : [ODBJScript.h](#)
- DUUNICODE : [ODBJScript.h](#)

- e -

- EDATASIZE : **ODBJScript.h**
- EDATATABLE : **ODBJScript.h**
- ENTRY : **ODBJScript.h**
- EXEFILENAME : **ODBJScript.h**

- f -

- FINALDEST : **ODBJScript.h**

- h -

- HMAINTHREAD : **ODBJScript.h**
- HPROCESS : **ODBJScript.h**

- i -

- IDATATABLE : **ODBJScript.h**
- IGNORE_ACCESS : **ODBJScript.h**
- IGNORE_ACCESS_KER32 : **ODBJScript.h**
- IGNORE_ALL : **ODBJScript.h**
- IGNORE_CUSTOM_EXCEPT : **ODBJScript.h**
- IGNORE_DIV_ZERO : **ODBJScript.h**
- IGNORE_FPU_EXCEPT : **ODBJScript.h**
- IGNORE_ILLEGAL_INST : **ODBJScript.h**
- IGNORE_INT3 : **ODBJScript.h**
- IGNORE_MANUALLY : **ODBJScript.h**
- IGNORE_NONE : **ODBJScript.h**
- IGNORE_TRAP : **ODBJScript.h**

- m -

- MAINBASE : **ODBJScript.h**
- MAINTHREADID : **ODBJScript.h**
- MEMORYBASE : **ODBJScript.h**
- MEMORYOWNER : **ODBJScript.h**
- MEMORIESIZE : **ODBJScript.h**
- MODULEBASE : **ODBJScript.h**
- MODULESIZE : **ODBJScript.h**

- n -

- NSECT : **ODBJScript.h**

- p -

- PROCESSID : **ODBJScript.h**
- PROCESSNAME : **ODBJScript.h**

- r -

- RELOCSIZE : **ODBJScript.h**
- RELOCTABLE : **ODBJScript.h**
- RESBASE : **ODBJScript.h**
- RESSIZE : **ODBJScript.h**

- s -

- SIZE : **ODBJScript.h**
- SYSTEMDIR : **ODBJScript.h**

- t -

- TYPE : **ODBJScript.h**