15.2.—Interfaz de programación de IDA

Las API de IDA están definidas por los contenidos de los archivos **header** que se encuentran en **<SDKDIR>\include**. No existe ni un simple índice de las funciones disponibles, sin embargo Steve Micallef ha reunido un buen conjunto en su libro. Muchos de los posibles programadores con SDK, encuentran dicho inconveniente como una dificultad para llegar a conseguir sus fines. ¿Cómo lo hago para que x haga algo, utilizando SDK? Las dos opciones principales como respuesta a esta pregunta son; postear cualquier duda en algún foro sobre IDA esperando recibir la solución o contestarse uno mismo buscando en la documentación de las API de IDA ¿Qué documentación, nos preguntamos? Pues los archivos header, por supuesto. Estos no son los documentos más buscados para solucionar el problema, pero son los que contienen todo el conjunto completo de características API. En este caso **grep**, o un sustituto adecuado como editor de programación, será nuestro aliado. Para encontrar hay que saber lo que se está buscando, cosa que no siempre será obvio.

Existen unas cuantas formas de acotar la búsqueda a través de las API. La primera forma es especular con el conocimiento del lenguaje script IDC e intentar localizar alguna funcionalidad similar dentro del SDK, utilizando palabras clave y nombres de función derivadas de IDC. Sin embargo, y es un hecho frustrante, SDK puede contener funciones que ejecuten tareas idénticas a las funciones IDC pero raramente los nombres de estas son idénticos. Esto da como resultado que los programadores tendrán que aprender dos conjuntos de llamadas a las API, uno para utilizar IDC y el otro para utilizar con el SDK. A fin de controlar dicha situación, realizaremos una lista completa de funciones IDC con sus correspondientes acciones en el SDK 5.2, para poderlas realizar y ejecutar.

La siguiente tabla sirve para combinar las funciones de script IDC con su ejecución con las del SDK. Esta tabla es para los que estén familiarizados con las funciones IDC, comprender como acciones similares se realizan con funciones del SDK. La necesidad de esta tabla viene dada por dos razones: La primera, los nombres de funciones IDC no tienen contrapartida con los del SDK y segunda, en algunos casos una simple función en IDC se compone de varias acciones del SDK. Esta tabla también expone alguna de las formas en que el SDK utiliza los **netnodes** como medios para guardar información en la base de datos de IDA. Específicamente la manera en que los netnodes son utilizados para ejecutar arrays IDC.

Para intentar ser breve en las descripciones SDK, se ha omitido el código de descripción de errores, debido a los distintos elementos sintácticos de C++ sobretodo por los {}. Muchas de las funciones SDK retornan resultados copiándolos en los buffer suministrados por el llamador. Estos buffer para ser breves, también se han dejado de declarar. Dichos buffer se nombran como **buf** y su tamaño en la mayoría de los casos se asume como 1.014 byte, que es el valor de la constante del SDK 5.2 **MAXSTR**. Finalmente la declaración de variables han sido usadas sólo donde dicho uso aumenta la comprensión del ejemplo. Los parámetros de entrada de la función IDC no se nombran, ya que pueden referenciarse en el sistema de ayuda de IDA.

Función IDC	Ejecución en SDK
AddBpt	//macro para AddBptEx(ea, 0, BPT_SOFT)
AddBptEx	add_bpt(ea, size, bpttype);
AddCodeXref	add_cref(From, To, flowtype);
AddConstEx	add_const(enum_id, name, value, bmask);
AddEntryPoint	add_entry(ordinal, ea, name, makecode);
AddEnum	add_enum(idx, name, flag);
AddHotkey	add_idc_hotkey(hotkey, idcfunc);
AddSourceFile	add_sourcefile(ea1, ea2, filename);
AddStrucEx	add_struc(index, name, is_union);

AddStrucMember	typeinfo_t mt;
	//Llama a una función interna mt para inicializarla utilizando typeid
	add_struc_member(get_struc(id), name, offset, flag, &mt,nbytes);

AltOp	get_forced_operand(ea, n, buf, sizeof(buf)); return qstrdup(buf);

AnalyzeArea	analyze_area(sEA, eEA);
Analysis	//macro
AppendFchunk	append_func_tail(get_func(funcea), ea1, ea2);
ApplySig	plan_to_apply_idasgn(name);
AskAddr	ea_t addr = defval; askaddr(&addr, "%s", prompt): return addr;
AskFile	return qstrdup(askfile_c(forsave, mask, "%s", prompt));
AskIdent	return qstrdup(askident(defval, "%s", prompt));
AskSeg	sel_t seg = defval; askseg(&sel, "%s", prompt): return val;
AskLong	sval_t val = defval; asklong(&val, "%s", prompt): return val;
AskSelector	return ask_selector(sel);
AskStr	return qstrdup(askstr(HIST_CMT, defval, "%s", prompt));
AskYN	return askyn_c(defval, "%s", prompt);

AttachProcess	return attach_process(pid, event_id);
AutoMark	//macro, ver AutoMark2
AutoMark2	auto_mark_range(start, end, queuetype);
AutoShow	//macro, ver SetCharPrm
AutoUnmark	//*** función indocumentada autoUnmark(start, end, type);
Batch	::batch = batch;
BeginEA	//macro, ver GetLongPrm
Byte	return get_full_byte(ea);
CanExceptionContinue	return get_debug_event()->can_cont;
ChooseFunction	return choose_func(ea, -1)->startEA;
CmtIndent	//macro, ver SetCharPrm
Comments	//macro, ver SetCharPrm

CommentEx	get_cmt(ea, repeatable, buf, sizeof(buf)); return qstrdup(buf);
Compile	CompileEx(filename, true, errbuf, sizeof(errbuf));

CreateArray	qsnprintf(buf, sizeof(buf), "$ idc_array %s", name);
	netnode n(buf, 0, true);
	return (nodeidx_t)n;

DelArrayElement	netnode n(id).supdel(idx, tag);
DelBpt	del_bpt(ea);
DelCodeXref	del_cref(From, To, undef);
DelConstEx	del_const(enum_id, value, serial, bmask);
DelEnum	del_enum(enum_id);
DelExtLnA	netnode n(ea).supdel(n + 1000);
DelExtLnB	netnode n(ea).supdel(n + 2000);
DelFixup	del_fixup(ea);
DelFunction	del_func(ea);

DelXML	del_xml(path);
DelHashElement	netnode n(id); n.hashdel(idx);
DelHiddenArea	hidden_areas->del_area(ea, true);
DelHotkey	del_idc_hotkey(hotkey);
DelLineNumber	del_source_linnum(ea);
DelSelector	del_selector(sel);
DelSourceFile	del_sourcefile(ea);
DelStruc	del_struc(get_struc(id));
DelStrucMember	del_struc_member(get_struc(id), offset);

DeleteAll	while (segs->get_area_qty())
	  del_segm(segs->getn_area(0), 0);
	FlagsDisable(0, inf.ominEA);
	FlagsDisable(inf.omaxEA, 0xFFFFFFFF);

DeleteArray	netnode n(id).kill();

Demangle	demangle_name(buf, sizeof(buf), name, disable_mask); return qstrdup(buf);
DetachProcess	detach_process();
Dfirst	return get_first_dref_from(From);
DfirstB	return get_first_dref_to(To);
Dnext	return get_next_dref_from(From, current);
DnextB	return get_next_dref_to(To, current);
Dword	return get_full_long(ea);
EnableBpt	enable_bpt(ea, enable);

EnableTracing	if (trace_level == 0)
	  return enable_step_trace(enable);
	else if (trace_level == 1)
	  return enable_insn_trace(enable);
	else if (trace_level == 2)
	  return enable_func_trace(enable);

Eval	idc_value_t v; calcexpr(-1, expr, &v, errbuf, sizeof(errbuf));
Exec	call_system(command);

Exit	qexit(code);
ExtLinA	netnode n(ea).supset(n + 1000, line); setFlbits(ea, FF_LINE);
ExtLinB	netnode n(ea).supset(n + 2000, line); setFlbits(ea, FF_LINE);
Fatal	error(format, ...);
FindBinary	ea_t endea = (flag & SEARCH_DOWN) ? inf.maxEA : inf.minEA; return find_binary(ea, endea, str, getDefaultRadix(), flag);
FindCode	return find_code(ea, flag);
FindData	return find_data(ea, flag);
FindExplored	return find_defined(ea, flag);
FindFuncEnd	func_t f; find_func_bounds(ea, &f, FIND_FUNC_DEFINE); return f->endEA;
FindImmediate	return find_imm(ea, flag, value);
FindSelector	return find_selector(val);

```
FindText                return find_text(ea, y, x, str, flag);

FindUnexplored          return find_unknown(ea, flag);

FindVoid                return find_void(ea, flag);

FirstFuncFchunk         get_func(funcea)->startEA;

FirstSeg                return segs->getn_area(0)->startEA;

GenCallGdl          gen_simple_call_chart(outfile, "Building graph",  title, flags);

GenFuncGdl              func_t *f = get_func(ea1);
                        gen_flow_graph(outfile, title, f, ea1, ea2, flags);

GenerateFile            gen_file(type, file_handle, ea1, ea2, flags);

GetArrayElement         netnode n(id);
                        if (tag == 'A') return n.altval(idx);
                        else if (tag == 'S')
                           n.supstr(idx, buf, sizeof(buf));
                           return qstrdup(buf);

GetArrayId              qsnprintf(buf, sizeof(buf), "$ idc_array %s", name);
                        netnode n(buf);
                        return (nodeidx_t)n;

GetBmaskCmt         get_bmask_cmt(enum_id, bmask, repeatable, buf, sizeof(buf));
                        return qstrdup(buf);
```

GetBmaskName

```
get_bmask_name(enum_id, bmask, buf, sizeof(buf));
return qstrdup(buf);
```

GetBptAttr

```
bpt_t bpt;
if (get_bpt(ea, &bpt) == 0) return -1;
if (bpattr == BPTATTR_EA) return bpt.ea;
else if (bpattr == BPTATTR_SIZE) return bpt.size;
else if (bpattr ==BPTATTR_TYPE) return bpt.type;
else if (bpattr == BPTATTR_COUNT) return
bpt.pass_count;
else if (bpattr == BPTATTR_FLAGS) return bpt.flags;
else if (bpattr == BPTATTR_COND) return
qstrdup(bpt.condition);
```

GetBptEA

```
bpt_t bpt
return getn_bpt(n, &bpt) ? bpt.ea : -1;
```

GetBptQty

```
return get_bpt_qty();
```

GetCharPrm

```
if (offset <= 191)
   return *(unsigned char*)(offset + (char*)&inf);
```

GetColor

```
if (what == CIC_ITEM)
   return get_color(ea);
else if (what == CIC_FUNC)
   return get_func(ea)->color;
else if (what == CIC_SEGM)
   return segs->get_area(ea)->color;
return 0xFFFFFFFF;
```

GetConstBmask

```
return get_const_bmask(const_id);
```

GetConstByName

```
return get_const_by_name(name);
```

GetConstCmt	get_const_cmt(const_id, repeatable, buf, sizeof(buf)); return qstrdup(buf);
GetConstEnum	return get_const_enum(const_id);
GetConstEx	return get_const(enum_id, value, serial, bmask);
GetConstName	get_const_name(const_id, buf, sizeof(buf)); return qstrdup(buf);
GetConstValue	return get_const_value(const_id);
GetCurrentLine	tag_remove(get_curline(), buf, sizeof(buf)) return qstrdup(buf);
GetCurrentThreadId	return get_current_thread();
GetdebuggerEvent	return wait_for_next_event(wfne, timeout);

Getdisasm	generate_disasm_line(ea, buf, sizeof(buf));
	tag_remove(buf, buf, 0);
	return qstrdup(buf);

GetEntryOrdinal	return get_entry_ordinal(index);
GetEnTRyPoint	return get_entry(ordinal);

GetEntryPointQty	return get_entry_qty();
GetEnum	return get_enum(name);
GetEnumCmt	get_enum_cmt(enum_id, repeatable, buf, sizeof(buf)); return qstrdup(buf);
GetEnumFlag	return get_enum_flag(enum_id);
GetEnumIdx	return get_enum_idx(enum_id);
GetEnumName	get_enum_name(enum_id, buf, sizeof(buf)); return qstrdup(buf);
GetEnumQty	return get_enum_qty();
GetEnumSize	return get_enum_size(enum_id);
GetEventBptHardwareEa	return get_debug_event()->bpt.hea;
GetEventEa	return get_debug_event()->ea;
GetEventExceptionCode	return get_debug_event()->exc.code;
GetEventExceptionEa	return get_debug_event()->exc.ea;

GetEventExceptionInfo	return qstrdup(get_debug_event()->exc.info);
GetEventExitCode	return get_debug_event()->exit_code;
GetEventId	return get_debug_event()->eid;
GetEventInfo	return qstrdup(get_debug_event()->info);
GetEventModuleBase	return get_debug_event()->modinfo.base;
GetEventModuleName	return qstrdup(get_debug_event()->modinfo.name);
GetEventModuleSize	return get_debug_event()->modinfo.size;
GetEventPid	return get_debug_event()->pid;
GetEventTid	return get_debug_event()->tid;
GetFchunkAttr	func_t *f = funcs->get_area(ea); return internal_get_attr (f, attr);
GetFirstBmask	return get_first_bmask(enum_id);
GetFirstConst	return get_first_const(enum_id, bmask);

GetFirstHashKey	netnode n(id).hash1st(buf, sizeof(buf)); return qstrdup(buf);
GetFirstIndex	return netnode n(id).sup1st(tag);
GetFirstMember	return get_struc_first_offset(get_struc(id));
GetFirstModule	module_info_t modinfo; get_first_module(&modinfo); return modinfo.base;
GetFirstStrucIdx	return get_first_struc_idx();
GetFixupTgtDispl	fixup_data_t fd; get_fixup(ea, &fd); return fd.displacement;
GetFixupTgtOff	fixup_data_t fd; get_fixup(ea, &fd); return fd.off
GetFixupTgtSel	fixup_data_t fd; get_fixup(ea, &fd); return fd.sel;
GetFixupTgtType	fixup_data_t fd; get_fixup(ea, &fd); return fd.type;
GetFlags	getFlags(ea);

GetFpNum

```
//*** función indocumentada
char buf[16];
union {float f; double d; long double ld} val;
get_many_bytes(ea, buf, len > 16 ? 16 : len);
ph.realcvt(buf, &val, (len >> 1) - 1);
return val;
```

GetFrame

```
//macro, ver GetFunctionAttr
```

GetFrameArgsSize

```
//macro, ver GetFunctionAttr
```

GetFrameLvarSize

```
//macro, ver GetFunctionAttr
```

GetFrameRegsSize

```
//macro, ver GetFunctionAttr
```

GetFrameSize

```
return get_frame_size(get_func(ea));
```

GetFuncOffset

```
int flags =  GNCN_REQFUNC | GNCN_NOCOLOR;
get_nice_colored_name(ea, buf, sizeof(buf),flags);
return qstrdup(buf);
```

GetFunctionAttr

```
func_t *f = get_func(ea);
return internal_get_attr(f, attr);
```

GetFunctionCmt

```
return funcs->get_area_cmt(get_func(ea), repeatable);
```

GetFunctionFlags

```
//macro, ver GetFunctionAttr
```

GetFunctionName	get_func_name(ea, buf, sizeof(buf)); return qstrdup(buf);
GetHashLong	netnode n(id).hashval_long(idx);
GetHashString	netnode n(id).hashval(idx, buf, sizeof(buf)); return qstrdup(buf);
GetIdaDirectory	qstrncpy(buf, idadir(NULL), sizeof(buf)); return qstrdup(buf);
GetIdbPath	qstrncpy(buf, database_idb, sizeof(buf)); return qstrdup(buf);
GetInputFile	get_root_filename(buf, sizeof(buf)); return qstrdup(buf);
GetInputFilePath	RootNode.valstr(buf, sizeof(buf)); return qstrdup(buf);
GetLastBmask	return get_last_bmask(enum_id);
GetLastConst	return get_last_const(enum_id, bmask);
GetLastHashKey	netnode n(id).hashlast(buf, sizeof(buf)); return qstrdup(buf);
GetLastIndex	return netnode n(id).suplast(tag);

GetLastMember

```
return get_struc_last_offset(get_struc(id));
```

GetLastStrucIdx

```
return get_last_struc_idx();
```

GetLineNumber

```
return get_source_linnum(ea);
```

GetLocalType

```
const type_t *type;
const p_list *fields;
get_numbered_type(idati, ordinal, &type, &fields,NULL,
NULL, NULL);
char *name = get_numbered_type_name(idati, ordinal);
qstring res;
print_type_to_qstring(&res, 0, 2, 40, flags, idati,
type,name, NULL, fields, NULL);
return qstrdup(res.c_str());
```

GetLocalTypeName

```
return qstrdup(get_numbered_type_name(idati, ordinal));
```

GetLongPrm

```
if (offset <= 188)
  return *(int*)(offset + (char*)&inf);
```

GetManualInsn

```
get_manual_insn(ea, buf, sizeof(buf));
return qstrdup(buf);
```

GetMarkComment

```
curloc loc.markdesc(slot, buf, sizeof(buf));
return qstrdup(buf);
```

GetMarkedPos

```
return curloc loc.markedpos(&slot);
```

GetMaxLocalType

```
return get_ordinal_qty(idati);
```

GetMemberComment

```
tid_t m = get_member(get_struc(id), offset)->id;
netnode n(m).supstr(repeatable ? 1 : 0, buf, sizeof(buf));
return qstrdup(buf);
```

GetMemberFlag

```
return get_member(get_struc(id), offset)->flag;
```

GetMemberName

```
tid_t m = get_member(get_struc(id), offset)->id;
get_member_name(m, buf, sizeof(buf));
return qstrdup(buf);
```

GetMemberOffset

```
return get_member_by_name(get_struc(id),
member_name)->soff;
```

GetMemberQty

```
get_struc(id)->memqty;
```

GetMemberSize

```
member_t *m = get_member(get_struc(id), offset);
return get_member_size(m);
```

GetMemberStrId

```
tid_t m = get_member(get_struc(id), offset)->id;
return netnode n(m).altval(3) - 1;
```

GetMnem

```
ua_mnem(ea, buf, sizeof(buf));
return qstrdup(buf);
```

GetModuleName

```
module_info_t modinfo;
if (base == 0)
get_first_module(&modinfo);
else
  modinfo.base = base - 1;
  get_next_module(&modinfo);
return qstrdup(modinfo.name);
```

GetModuleSize

```
module_info_t modinfo;
if (base == 0)
  get_first_module(&modinfo);
else
  modinfo.base = base - 1;
  get_next_module(&modinfo);
return modinfo.size;
```

GetNextBmask

```
return get_next_bmask(eum_id, value);
```

GetNextConst

```
return get_next_const(enum_id, value, bmask);
```

GetNextFixupEA

```
return get_next_fixup_ea(ea);
```

GetNextHashKey

```
netnode n(id).hashnxt(idx, buf, sizeof(buf));
return qstrdup(buf);
```

GetNextIndex

```
return netnode n(id).supnxt(idx, tag);
```

GetNextModule

```
module_info_t modinfo;
modinfo.base = base;
get_next_module(&modinfo);
return modinfo.base;
```

GetNextStrucIdx

```
return get_next_struc_idx();
```

GetOpType

```
*buf = 0;
if (isCode(get_flags_novalue(ea))
  ua_ana0(ea);
  return cmd.Operands[n].type;
```

GetOperandValue	Utiliza ua_ana0 para rellenar command struct entonces retorna un valor apropiado basado en cmd.Operands[n].type

GetOpnd	*buf = 0;
	if (isCode(get_flags_novalue(ea))
	  ua_outop2(ea, buf, sizeof(buf), n);
	tag_remove(buf, buf, sizeof(buf));
	return qstrdup(buf);

GetOriginalByte	return get_original_byte(ea);
GetPrevBmask	return get_prev_bmask(enum_id, value);
GetPrevConst	return get_prev_const(enum_id, value, bmask);
GetPrevFixupEA	return get_prev_fixup_ea(ea);
GetPrevHashKey	netnode n(id).hashprev(idx, buf, sizeof(buf)); return qstrdup(buf);
GetPrevIndex	return netnode n(id).supprev(idx, tag);
GetPrevStrucIdx	return get_prev_struc_idx(index);

GetProcessName	process_info_t p;
	pid_t pid = get_process_info(idx, &p);
	return qstrdup(p.name);

GetProcessPid

```
return get_process_info(idx, NULL);
```

GetProcessQty

```
return get_process_qty();
```

GetProcessState

```
return get_process_state();
```

GeTReg

```
return getSR(ea, str2reg(reg));
```

GetregValue

```
regval_t r;
get_reg_val(name, &r);
if (is_reg_integer(name))
  return (int)r.ival;
else
  memcpy(result, r.fval, 12);
```

GetSegmentAttr

```
segment_t *s = segs->get_area(segea);
return internal_get_attr (s, attr);
```

GetShortPrm

```
if (offset <= 190)
  return *(unsigned short*)(offset + (char*)&inf);
```

GetSourceFile

```
return qstrdup(get_sourcefile(ea));
```

GetSpDiff

```
return get_sp_delta(get_func(ea), ea);
```

GetSpd

```
return get_spd(get_func(ea), ea);
```

GetString

```
if (len == -1)
len = get_max_ascii_length(ea, type, true);
get_ascii_contents(ea, len, type, buf, sizeof(buf));
return qstrdup(buf);
```

GetStringType	return netnode n(ea).altval(16) - 1;
GetStrucComment	get_struc_cmt(id, repeatable, buf, sizeof(buf)); return qstrdup(buf);
GetStrucId	return get_struc_by_idx(index);
GetStrucIdByName	return get_struc_id(name);
GetStrucIdx	return get_struc_idx(id);
GetStrucName	get_struc_name(id, buf, sizeof(buf)); return qstrdup(buf);
GetStrucNextOff	return get_struc_next_offset(get_struc(id), offset);
GetStrucPrevOff	return get_struc_prev_offset(get_struc(id), offset);
GetStrucQty	return get_struc_qty();
GetStrucSize	return get_struc_size(id);
GetTHReadId	return getn_thread(idx);
GetTHReadQty	return get_thread_qty();

GettrueName	//macro, ver GetTrueNameEx
GettrueNameEx	return qstrdup(get_true_name(from, ea, buf, sizeof(buf)));

GetType	get_ti(ea, tbuf, sizeof(tbuf), plist, sizeof(plist));
	print_type_to_one_line(buf, sizeof(buf), idati,tbuf, NULL, NULL, plist, NULL);
	return qstrdup(buf);

GetnEnum	return getn_enum(idx);

GetVxdFuncName	//*** función indocumentada
	get_vxd_func_name(vxdnum, funcnum, buf, sizeof(buf));
	return qstrdup(buf);

GetXML	valut_t res; get_xml(path, &res); return res;

GuessType	guess_type(ea, tbuf, sizeof(tbuf), plist, sizeof(plist));
	print_type_to_one_line(buf, sizeof(buf), idati, tbuf,NULL, NULL, plist, NULL);
	return qstrdup(buf);

HideArea	add_hidden_area(start,end,description,header,footer,color);
HighVoids	//macro, ver SetLongPrm
Indent	//macro, ver SetCharPrm

IsBitfield	return is_bf(enum_id);
IsEventHandled	return get_debug_event()->handled;
IsUnion	return get_struc(id)->is_union();
ItemEnd	return get_item_end(ea);
ItemSize	return get_item_end(ea) - ea;
Jump	jumpto(ea);
LineA	netnode n(ea).supstr(1000 + num, buf, sizeof(buf)); return qstrdup(buf);
LineB	netnode n(ea).supstr(2000 + num, buf, sizeof(buf)); return qstrdup(buf);
LoadDebugger	load_debugger(dbgname, use_remote);
LoadTil	return add_til2(name, 0);
LocByName	return get_name_ea(-1, name);
LocByNameEx	return get_name_ea(from, name);

LowVoids	//macro, ver SetLongPrm
MK_FP	return ((seg<<4) + off);
MakeAlign	doAlign(ea, count, align);

MakeArray	typeinfo_t ti;
	flags_t f = get_flags_novalue(ea);
	get_typeinfo(ea, 0, f, &ti);
	asize_t sz = get_data_elsize(ea, f, &ti);
	do_data_ex (ea, f, sz * nitems, ti.tid);

MakeByte	//macro, ver MakeData
MakeCode	ua_code(ea);
MakeComm	set_cmt(ea, cmt, false);
MakeData	do_data_ex(ea, flags, size, tid);
MakeDouble	//macro, ver MakeData
MakeDword	//macro, ver MakeData
MakeFloat	//macro, ver MakeData

MakeFrame

```
func_t *f = get_func(ea);
set_frame_size(f, lvsize, frregs, argsize);
return f->frame;
```

MakeFunction

```
add_func(start, end);
```

MakeLocal

```
func_t *f = get_func(ea);
if (*location != '[')
  add_regvar(f, start, end, location, name, NULL);
else
  struc_t *fr = get_frame(f);
  int start = f->frsize + offset;
  if (get_member(fr, start))
    set_member_name(fr, start, name);
  else
    add_struc_member(fr, name, start,  0x400, 0, 1);
```

MakeNameEx

```
set_name(ea, name, flags);
```

MakeOword

```
//macro, ver MakeData
```

MakePackReal

```
//macro, ver MakeData
```

MakeQword

```
//macro, ver MakeData
```

MakeRptCmt

```
set_cmt(ea, cmt, true);
```

MakeStr

```
int len = endea == -1 ? 0 : endea - ea;
make_ascii_string(ea, len, current_string_type);
```

MakeStructEx	netnode n(strname);
	nodeidx_t idx = (nodeidx_t)n;
	if (size != -1)
	  do_data_ex(ea, FF_STRU, size, idx);
	else
	  size_t sz = get_struc_size(get_struc());
	  do_data_ex(ea, FF_STRU, sz, idx);

MakeTbyte	//macro, ver MakeData
MakeUnkn	do_unknown(ea, flags);
MakeUnknown	do_unknown_range(ea, size, flags);
MakeVar	doVar(ea);
MakeWord	//macro, ver MakeData

MarkPosition	curloc loc;
	loc.ea = ea; loc.lnnum = lnnum; loc.x = x; loc.y = y;
	loc.mark(slot, NULL, comment);

MaxEA	//macro, ver GetLongPrm
Message	msg(format, ...);
MinEA	//macro, ver GetLongPrm
Name	return qstrdup(get_name(-1, ea, buf, sizeof(buf)));

NameEx	return qstrdup(get_name(from, ea, buf, sizeof(buf)));
NextAddr	return nextaddr(ea);
NextFchunk	return funcs->getn_area(funcs->get_next_area(ea))->startEA;
NextFuncFchunk	func_tail_itera tor_t fti(get_func(funcea), tailea); return fti.next() ? fti.chunk().startEA : -1;
NextFunction	return get_next_func(ea)->startEA;
NextHead	return next_head(ea, maxea);
NextNotTail	return next_not_tail(ea);
NextSeg	int n = segs->get_next_area(ea); return segs->getn_area(n)->startEA;
OpAlt	set_forced_operand(ea, n, str);
OpBinary	set_op_type(ea, binflag(), n);
OpChr	set_op_type(ea, charflag(), n);
OpDecimal	set_op_type(ea, decflag(), n);

OpEnumEx	op_enum(ea, n, enumid, serial);
OpHex	set_op_type(ea, hexflag(), n);
OpHigh	return op_offset(ea, n, REF_HIGH16, target);
OpNot	toggle_bnot(ea, n);
OpNumber	set_op_type(ea, numflag(), n);
OpOctal	set_op_type(ea, octflag(), n);
OpOff	if (base != 0xFFFFFFFF) set_offset(ea, n, base); else noType(ea, n);
OpOffEx	op_offset(ea, n, reftype, target, base, tdelta);
OpSeg	op_seg(ea, n);
OpSign	toggle_sign(ea, n);
OpStkvar	op_stkvar(ea, n);
OpStroffEx	op_stroff(ea, n, &strid, 1, delta);

ParseTypes	int hti_flags = (flags & 0x70) << 8);
	if (flags & 1) hti_flags |= HTI_FIL;
	parse_types2(input, (flags & 2) ? NULL :
	printer_func,hti_flags);

PatchByte	patch_byte(ea, value);
PatchDword	patch_long(ea, value);
PatchWord	patch_word(ea, value);
PauseProcess	suspend_process();
PopXML	pop_xml();
PrevAddr	return prevaddr(ea);
PrevFchunk	return funcs->getn_area(funcs->get_prev_area(ea))->startEA;
PrevFunction	return get_prev_func(ea)->startEA;
PrevHead	return prev_head(ea, minea);
PrevNotTail	return prev_not_tail(ea);

PushXML	push_xml(path);

RefreshDebuggerMemory	invalidate_dbgmem_config();
	invalidate_dbgmem_contents(-1, -1);
	if (dbg && dbg->stopped_at_debug_event)dbg->stopped_at_debug_event(true);

Refresh	refresh_idaview_anyway();
RefreshLists	callui(ui_list);
RemoveFchunk	remove_func_tail(get_func(funcea), tailea);
RenameArray	qsnprintf(buf, sizeof(buf), "$ idc_array %s", name); netnode n(id).rename(newname);
RenameEntryPoint	rename_entry(ordinal, name);
Rfirst	return get_first_cref_from(From);
Rfirst0	return get_first_fcref_from(From);
RfirstB	return get_first_cref_to(To);
RfirstB0	return get_first_fcref_to(To);
Rnext	return get_next_cref_from(From, current);

Rnext0	return get_next_fcref_from(From, current);
RnextB	return get_next_cref_to(To, current);
RnextB0	return get_next_fcref_to(To, current);
RunPlugin	run_plugin(load_plugin(name), arg);
RunTo	run_to(ea);
ScreenEA	return get_screen_ea();
SegAddrng	set_segm_addressing(segs->get_area(ea), use32);
SegAlign	//macro, ver SetSegmentAttr

SegBounds	if (segs->get_area(ea))
	  set_segm_end(ea, endea, disable);
	  set_segm_end(ea, startea, disable);

SegByBase	return get_segm_by_sel(base)->startEA;
SegByName	sel_t seg; atos(segname, *seg); return seg;

SegClass
```
set_segm_class(segs->get_area(ea), class);
```

SegComb
```
//macro, ver SetSegmentAttr
```

SegCreate
```
segment_t s;
s.startEA = startea;
s.sel = setup_selector(base);
s.bitness = use32;
s.align = align;
s.comb = comb;
return add_segm_ex(&s, NULL, NULL, ADDSEG_NOSREG);
```

SegDefReg
```
SetDefaultRegisterValue(segs->get_area(ea),str2reg(reg), value);
```

SegDelete
```
del_segm(ea, flags);
```

SegEnd
```
//macro, ver GetSegmentAttr
```

SegName
```
segment_t *s = (segment_t*) segs->get_area(ea);
get_true_segm_name(s, buf, sizeof(buf));
return qstrdup(buf);
```

SegRename
```
set_segm_name(segs->get_area(ea), "%s", name);
```

SegStart
```
//macro, ver GetSegmentAttr
```

SelEnd
```
ea_t ea1, ea2;
read_selection(&ea1, &ea2);
return ea2;
```

SelStart	ea_t ea1, ea2; read_selection(&ea1, &ea2); return ea1;
SelectThread	select_thread(tid);
SetArrayLong	netnode n(id).altset(idx, value);
SetArrayString	netnode n(id).supset(idx, str);
SetBmaskCmt	set_bmask_cmt(enum_id, bmask, cmt, repeatable);
SetBmaskName	set_bmask_name(enum_id, bmask, name);

SetBptAttr	bpt_t bpt;
	if (get_bpt(ea, &bpt) == 0) return;
	if (bpattr == BPTATTR_SIZE) bpt.size = value;
	else if (bpattr ==BPTATTR_TYPE) bpt.type = value;
	else if (bpattr == BPTATTR_COUNT) bpt.pass_count = value;
	else if (bpattr == BPTATTR_FLAGS) bpt.flags = value;
	update_bpt(&bpt);

SetBptCnd	bpt_t bpt;
	if (get_bpt(ea, &bpt) == 0) return;
	qstrncpy(bpt.condition, cnd, sizeof(bpt.condition));
	update_bpt(&bpt);

SetCharPrm	if (offset >= 13 && offset <= 191) *(offset + (char*)&inf) = value;

SetColor	if (what == CIC_ITEM)
	  set_item_color(ea, color);
	else if (what == CIC_FUNC)
	  get_func(ea)->color = color;
	  funcs->update(get_func(ea));
	else if (what == CIC_SEGM)
	  segs->get_area(ea)->color = color;
	  segs->update(segs->get_area(ea));

SetConstCmt	set_const_cmt(const_id, cmt, repeatable);
SetConstName	set_const_name(const_id, name);
SetDebuggerOptions	return set_debugger_options(options);
SetEnumBf	set_enum_bf(enum_id, flag ? 1 : 0);
SetEnumCmt	set_enum_cmt(enum_id, cmt, repeatable);
SetEnumFlag	set_enum_flag(enum_id, flag);
SetEnumIdx	set_enum_idx(enum_id, idx);
SetEnumName	set_enum_name(enum_id, name);

SetFchunkAttr	func_t *f = funcs->get_area(ea);
	internal_set_attr (f, attr, value);
	funcs->update(f);

SetFchunkOwner	set_tail_owner(funcs->get_area(tailea), funcea);

SetFixup	fixup_data_t f = {type, targetsel, targe toff, displ}; set_fixup(ea, &f);
SetFlags	setFlags(ea, flags);
SetFunctionAttr	func_t *f = get_func(ea); internal_set_attr (f, attr, value);
SetFunctionCmt	funcs->set_area_cmt(get_func(ea), cmt, repeatable);
SetFunctionEnd	func_setend(ea, end);
SetFunctionFlags	//macro, ver SetFunctionFlags

SetHiddenArea	hidden_area_t *ha = hidden_areas->get_area(ea);
	ha->visible = visible;
	update_hidden_area(ha);

SetManualInsn	set_manual_insn(ea, insn);
SetHashLong	netnode n(id).hashset(idx, value);
SetHashString	netnode n(id).hashset(idx, value);
SetLineNumber	set_source_linnum(ea, lnnum);

SetLocalType

```
if (input == NULL || *input == 0)
  del_numbered_type(idati, ordinal);
else
  qstring name;
  qtype type, fields;
  parse_decl(idati, input, &name, &type, &fields, flags);
  if (ordinal == 0)
    if (!name.empty())
      get_named_type(idati, name.c_str(),
        NTF_TYPE | NTF_NOBASE, NULL, NULL,
        NULL, NULL, NULL, &ordinal);
if (!ordinal)
  ordinal = alloc_type_ordinal(idati);
    set_numbered_type(idati, value, 0, name.c_str(),
                type.c_str(), fields.c_str(),
                NULL, NULL, NULL);
```

SetLongPrm

```
if (offset >= 13 && offset <= 188)
  *(int*)(offset + (char*)&inf) = value;
```

SetMemberComment

```
member_t *m = get_member(get_struc(ea),
member_offset);
set_member_cmt(m, comment, repeatable);
```

SetMemberName

```
set_member_name(get_struc(ea), member_offset, name);
```

SetMemberType

```
typeinfo_t mt;
//llama función interna mt para inicializar utilizando typeid
int size =  get_data_elsize(-1, flag, &mt) *  nitems;
set_member_type(get_struc(id), member_offset,flag, &mt,size);
```

SetProcessorType

```
set_processor_type(processor, level);
```

SetReg

```
splitSRarea1(ea, str2reg(reg), value, SR_user, false);
```

SetRemoteDebugger	set_remote_debugger(hostname, password, portnum);

SetRegValue	regval_t r;
	if (is_reg_integer(name))
	  r.ival = unsigned int)VarLong(value);
	else
	  memcpy(r.fval, VarFloat(value), 12);
	set_reg_val(name, &r);

SetSegmentAttr	segment_t *s = segs->get_area(segea);
	internal_set_attr(s, attr, value);
	segs->update(s);

SetSegmentType	//macro, ver SetSegmentAttr
SetSelector	set_selector(sel, value);
SetShortPrm	if (offset >= 13 && offset <= 190) *(short*)(offset + (char*)&inf) = value;
SetSpDiff	add_user_stkpnt(ea, delta);
SetStatus	setStat(status);
SetStrucComment	set_struc_cmt(id, cmt, repeatable);
SetStrucIdx	set_struc_idx(get_struc(id), index);
SetStrucName	set_struc_name(id, name);

SetType	apply_cdecl(ea, type)
	if (get_aflags(ea) & AFL_TILCMT)
	  set_ti(ea, "", NULL);

SetXML	set_xml(path, name, value);
StartDebugger	start_process(path, args, sdir);
StepInto	step_in to();
StepOver	step_over();
StepUntilRet	step_until_ret();
StopDebugger	exit_process();
StringStp	//macro, ver SetCharPrm
Tabs	//macro, ver SetCharPrm
TakeMemorySnapshot	take_memory_snapshot(only_loader_segs);
TailDepth	//macro, ver SetLongPrm
Til2Idb	return til2idb(idx, type_name);

Voids	//macro, ver SetCharPrm
Wait	autoWait();
Warning	warning(format, ...);
Word	return get_full_word(ea);
XrefShow	//macro, ver SetCharPrm
XrefType	Retorna el valor de una variable global interna
add_dref	add_dref(From, To, drefType);
atoa	ea2str(ea, buf, sizeof(buf)); return qstrdup(buf);
atol	return atol(str);
byteValue	//macro
del_dref	del_dref(From, To);
fclose	qfclose(handle);

fgetc	return qfgetc(handle);
filelength	return efilelength(handle);
fopen	return qfopen(file, mode);
form	internal_snprintf (buf, sizeof(buf), format, ...); return qstrdup(buf);
fprintf	qfprintf(handle, format, ...);
fputc	qfputc(byte, handle);
fseek	qfseek(handle, offset, origin);
ftell	return qftell(handle);
hasName	//macro
hasValue	//macro
isBin0	//macro
isBin1	//macro
isChar0	//macro
isChar1	//macro
isCode	//macro
isData	//macro

isDec0 //macro

isDec1 //macro

isDefArg0 //macro

isDefArg1 //macro

isEnum0 //macro

isEnum1 //macro

isExtra //macro

isFlow //macro

isFop0 //macro

isFop1 //macro

isHead //macro

isHex0 //macro

isHex1 //macro

isLoaded //macro

isOct0 //macro

isOct1 //macro

isOff0 //macro

isOff1 //macro

isRef //macro

isSeg0 //macro

isSeg1 //macro

isStkvar0 //macro

isStkvar1 //macro

isStroff0 //macro

isStroff1 //macro

isTail	//macro
isUnknown	//macro
isVar	//macro

loadfile	linput_t *li = make_linput(handle);
	file2base(li, pos, ea, ea + size, false);
	unmake_linput(li);

ltoa	Llamada interna de una rutina de conversión
ord	return str[0];
readlong	unsigned int res; freadbytes(handle, &res, 4, mostfirst); return res;
readshort	unsigned short res; freadbytes(handle, &res, 2, mostfirst); return res;
readstr	qfgets(buf, sizeof(buf), handle); return qstrdup(buf);
rotate_left	return rotate_left(value, count, nbits, offset);
savefile	base2file(handle, pos, ea, ea + size);
set_start_cs	//macro, ver SetLongPrm

set_start_ip	//macro, ver SetLongPrm
strlen	return strlen(str);
strstr	return strstr(str, substr);
substr	Llamada interna de una rutina de corte
writelong	fwritebytes(handle, &dword, 4, mostfirst);
writeshort	fwritebytes(handle, &word, 2, mostfirst);
writestr	qfputs(str, handle);
xtol	return strtoul(str, NULL, 16);

____	//*** función indocumentada (cuatro guiones)
	//retorna a la base de datos su fecha y hora de creación
	return RootNode.altval(RIDX_ALT_CTIME);

La segunda técnica para acotar la búsqueda en SDK es familiarizarse con el contenido y con el propósito de cada archivo header. Por lo general las funciones relacionadas y asociadas a tipos de estructuras de datos están agrupadas dentro de los archivos header. Por ejemplo las funciones SDK que permiten una interacción con el usuario están agrupadas dentro de **kernwin.hpp**. Cuando en grep realicemos una búsqueda y no logre localizar la pedido, tener el conocimiento de en qué archivo header se puede relacionar nos ayudará a encontrar lo buscado.

Performance Bigundill@