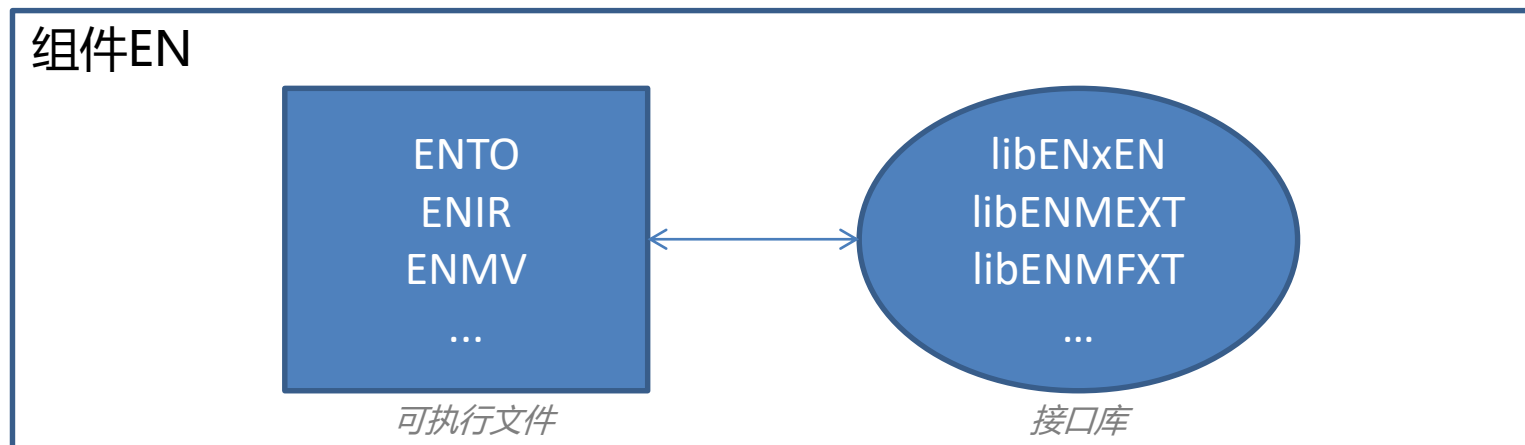


# 一些总结

## • 模块、组件

**模块**：就是一个待解析.idb文件，有的.idb文件以“lib” 开头，这种文件是接口库文件，不带“lib” 的文件通常为可执行文件。通常以其文件名（去除后缀及lib）为模块名，如模块为EHHO、ENMEXT。

**组件**：模块名头两个字母相同的软件模块（包括可执行文件、接口库）的组合。通常以这两个字母（一般都是大写）为“组件名”，如组件EN。



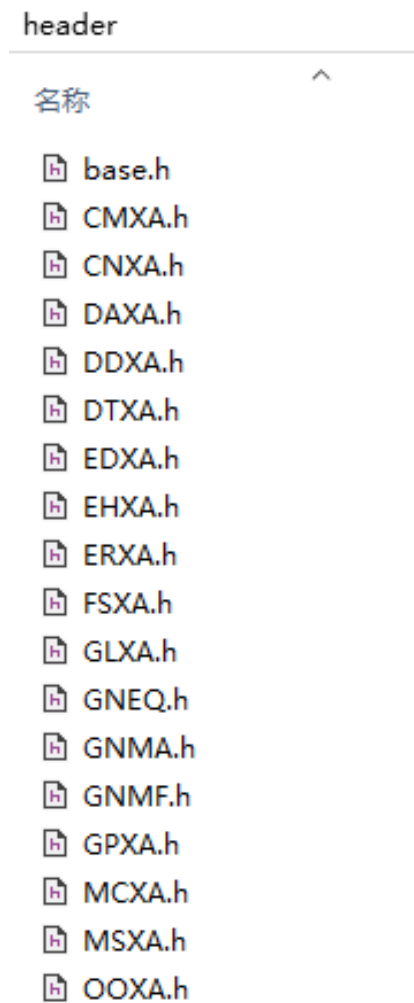
各模块间通过接口库进行交互。

**约定**：CC表示任一组件，CCXX表示任一模块

## • 关于接口库头文件

header文件夹存放了一些已解析出的接口库头文件，供解析时参考。

各头文件（base.h除外）中的结构体名、函数名都以模块名开头，当解析到外部函数调用或外部结构体时，请根据函数名或结构体名去对应的头文件中寻找定义，如：函数THXAtrace的声明在头文件THXA.h中。



```
GNMA.h  TMXA.h  THXA.h  SBXA.h
js Files  (Global Scope)

int THXAcheckSimMode(const char *psComponent,
    THXA_SIM_MODE eSimMode);    //1

int THXAsetMode(const char *psComponent,    //"ZD"
    THXA_SIM_MODE eSimMode,
    THXA_TRACE_MODE eTraceMode,
    THXA_REQ_MODE eReqMode);

int THXAgetMode(const char *psComponent,    //"ZD"
    THXA_SIM_MODE *peSimMode,
    THXA_TRACE_MODE *peTraceMode,
    THXA_REQ_MODE *peReqMode);

//数据打印
void THXAtrace(const char *psComponent,    //组件名
    int iMode,    //0/1/2
    const char *psFunction,    //文件名，解析时用__F__
    const char *psFormat,
```

## • 基础头文件base.h

base.h中定义了表示位置的向量数据、复数、时间戳结构体：

xyvect

xyzvect

xyavect

zrxryvect

horvervect

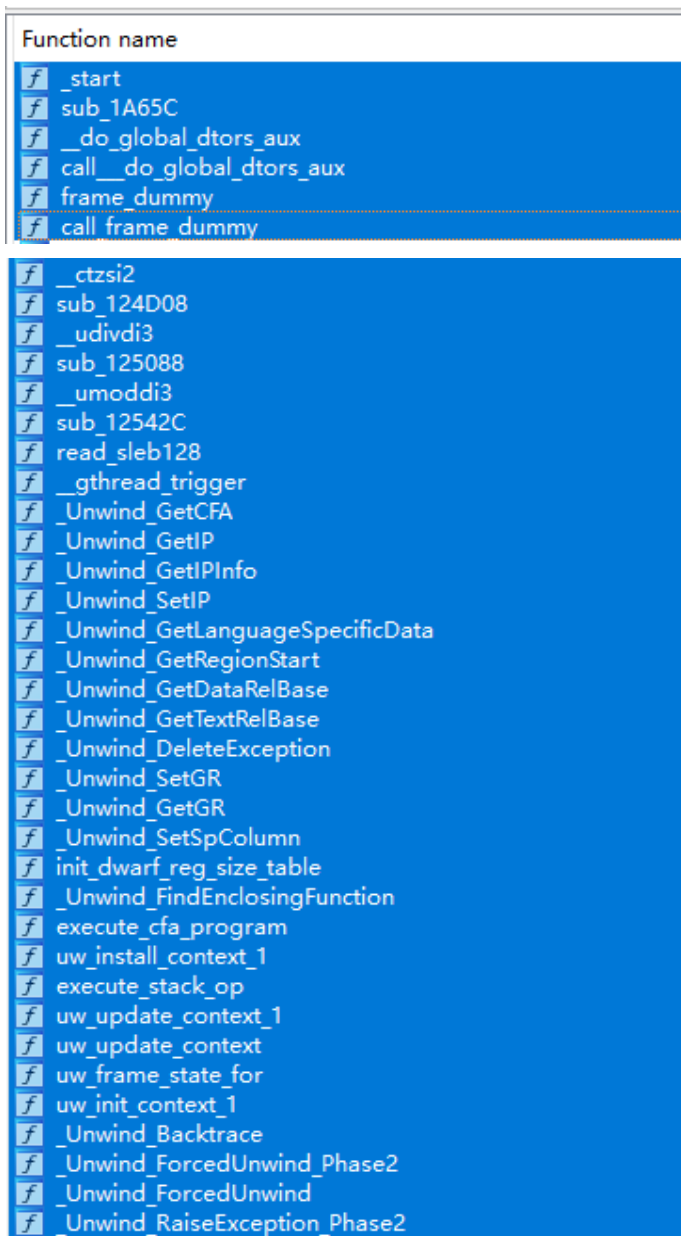
complex

Timestamp

这些结构体大量的在待解析模块中使用。

```
#define _VOID_PTR_ void * //未确定指针定义。
#define _SHM_PTR_ void * //指向共享内存的指针。
typedef int bool;
typedef struct
{
    double x;
    double y;
} xyvect;
typedef struct
{
    double x;
    double y;
    double z;
} xyzvect;
typedef struct
{
    double x;
    double y;
    double a;
} xyavect;
typedef struct
{
    double z;
    double rx;
    double ry;
} zrxryvect;
typedef struct
{
    double x;
    double y;
    double rz;
    double z;
    double rx;
    double ry;
} horvervect;
typedef struct
{
    double re;
    double im;
} complex;
typedef struct
{
    int iSecond; // 1s
    int iMicrosecond; // 1/1000000 s
} timestamp;
```

## • 一个.idb文件内是不是所有的函数都需要解析？



模块中左边这样的函数是程序加载和退出的代码，**无需解析**成C语言。

一般的，任务发放方会给一个excel列出需要解析的函数。

函数名	长度	是否分
enghxap_functions_install_custom_services_cb	32	1
enghxap_functions_execute_service_cb	227	1
enghxap_functions_validate_input_parameters_cb	295	1
enghxap_functions_get_time_limits_for_step_cb	94	1
enghxap_functions_should_step_be_executed_cb	95	1
enghxap_functions_execute_step_and_update_results_cb	2812	1
enghxap_functions_get_result_check_cb	530	1
enghxap_functions_save_and_activate_mc_cb	30	1
enghxap_functions_clear_result_parameters_cb	181	1
enghxap_functions_clean_up_machine_cb	27	1
enghxap_functions_destroy_data_structure_cb	176	1
sub_19390	2	0
ENGHxAP_graphics	131	0
sub_195A4	2	0

对于只有两行汇编码的sub\_\*\*\*\*函数，这种函数用于计算得到全局变量.got表首地址%l7，无需解析。

```
.text:00012888 sub_12888:                                ? CODE XREF: EMAZxAP_functions_get_config+8↓p
.text:00012888                                     ? EMAZxAP_functions_create_data_structure+8↓p ...
.text:00012888                                     retl
.text:0001288C                                     add     %07, %l7, %l7
.text:0001288C ? End of function sub_12888
.text:0001288C
.text:00012890
.text:00012890 ? ===== S U B R O U T I N E =====
.text:00012890 ? Attributes: bp-based frame
.text:00012890
.text:00012890 .global EMAZxAP_functions_get_config
.text:00012890 EMAZxAP_functions_get_config:                ? DATA XREF: .got:EMAZxAP_functions_get_config_ptr↓
.text:00012890 save     %sp, -0x60, %sp
.text:00012894 sethi    %hi(loc_22C00), %l7
.text:00012898 call     sub_12888
.text:0001289C set      loc_22C70, %l7
.text:000128A0 set      0x190, %g1
.text:000128A8 ld        [%l7+%g1], %00 ? //%g=0x190, %l7+%g1=35698, %00=233B8, _LLC0_0
.text:000128A8                                     ? %00 = "EMAZ;
.text:000128AC mov      2, %01
.text:000128B0 set      0x17C, %g1
.text:000128B8 ld        [%l7+%g1], %02 ? //%g=0x17C, %l7+%g1=35684, %02=23F10, __FUNCTION_
.text:000128B8                                     ? %02 = "EMAZxAP_functions_get_config;
.text:000128BC set      0x194, %g1
.text:000128C4 call     _THXAttrace
.text:000128C8 ld        [%l7+%g1], %03 ? //%g=0x194, %l7+%g1=3569C, %03=233C0, _LLC1_0
.text:000128C8                                     ? %03 = "> ();
.text:000128CC cmp      %i1, 0
```

确定Got表首地址放入%l7

## • 关于函数名

函数名大都以“模块名”开头：

以大写模块名开头的函数，一般为该模块对外的接口或全局函数

以小写模块名开头的函数，一般为该模块的内部函数（静态函数）

不带模块名的函数，一般为该模块的内部函数（静态函数）

The screenshot displays the IDA Pro interface with three main windows: Functions window, IDA View-A, and Hex View-1.

**Functions window:** Lists functions with their names. Red boxes highlight specific naming conventions:

- 不带模块名, KDAW模块局部函数:** Points to functions like `KDAW_color_dyn` and `icompare`.
- 小写模块名开头, KDAW模块局部函数:** Points to functions like `kdaw_determine_cowa_scan_type`.
- 大写模块名开头, KDAW模块全局函数:** Points to functions like `KDAW_alignement_wizard.c` in the Hex View.

**Hex View-1:** Shows the source code of the selected function, `KDAW_alignement_wizard.c`. The function name is highlighted in yellow.

## • 函数的内部实现

- ✓ 大部分函数返回值为int，返回0表示函数执行成功，返回非0表示函数执行错误，返回值代表了错误类型，因此该值称为“错误码”。

**错误码**为0xYYYYZZZZ的形式，YYYY为某个组件名的16进制ASCII码，ZZZZ为0000-FFFF，如0x454E0000，表示EN组件的某个错误类型编号。

（E的ASCII码为45，N的ASCII码为4E）。

- ✓ 一个函数入口处通常有一条或多条THXAtrace语句调用，打印该函数的输入参数。
- ✓ 一个函数出口处通常有一条或多条THXAtrace语句调用，打印该函数的输出参数及返回值。
- ✓ 一个函数内部实现多以“call function”的形式出现，函数执行失败，则赋返回值为某个错误码，并调用ERXA相关接口记录错误；下一步判断返回值是否为0，为0，则继续下一函数调用。

ASCII 字符表

字符(10#)	16#
A	65 41
B	66 42
C	67 43
D	68 44
E	69 45
F	70 46
G	71 47
H	72 48
I	73 49
J	74 4A
K	75 4B
L	76 4C
M	77 4D
N	78 4E
O	79 4F
P	80 50



# 函数内部实现

```
int CCXX_function(...)  
{  
    int iErrorCode = 0;  
    char *psErrorText = NULL;
```

通常定义返回值为: **int iErrorCode**;  
错误描述信息为: **char \*psErrorText**;

```
//入口trace  
THXAtrace(...);
```

入口**trace**;

```
    iErrorCode = some_function1(...);  
    if (0 != iErrorCode)  
    {  
        psErrorText = ERXAmakeContext(...);  
        ERXalogExceptionSingleLink_id(0x45520072,  
            iErrorCode, &iErrorCode, __FILE__, __LINE__, 0, psErrorText);  
    }
```

函数调用, 返回非**0**, 则调用**ERXA**接口进行记录

```
    if (0 == iErrorCode)  
    {  
        iErrorCode = some_function2(...);  
        if (0 != iErrorCode)  
        {  
            psErrorText = ERXAmakeContext(...);  
            ERXalogExceptionSingleLink_id(0x45520072,  
                iErrorCode, &iErrorCode, __FILE__, __LINE__, 0, psErrorText);  
        }  
    }
```

下一函数调用前先判断**iErrorCode**是否为**0**

```
    ...
```

```
//出口trace  
THXAtrace(...);  
return iErrorCode;  
}
```

出口**trace**;

## • THXA接口调用解析

此接口类似于C语言的printf函数，支持不确定个数参数。  
用于打印重要参数信息。

//数据打印

```
void THXAtrace(const char *psComponent, //组件名或模块名，如“ZD”  
               int iMode,               //0/1/2  
               const char *psFunction,  //文件名，解析时用__FUNCTION__  
               const char *psFormat,  
               ...);
```

除了函数入口、出口有trace，函数中间也可能有trace。

通过trace语句，可以判断函数参数名及类型、变量参数名及类型。

# 入口trace

enhoxap\_functions\_execute\_step\_and\_update\_results\_cb:

? DATA XREF: .got:enhoxap\_functions\_execute\_step\_and\_update\_res

var\_544 = -0x544  
var\_538 = -0x538  
var\_1C = -0x1C  
iErrorCode = -4

```
save    %sp, -0x5A0, %sp
sethi   %hi(aPScanEmptyScan), %17 ? "p scan : empty scan"
call    sub_12EAC
set     (_LLC147_0+0x50), %17
clr     [%fp+iErrorCode]
add     %fp, var_538, %o0
mov     0, %o1
call    _memset
mov     0x530, %o2
set     0x278, %g1
ld      [%17+%g1], %o0 ? //%g=0x278, %17+%g1=5E7E0, %o0=3C8B8, _LLC0_0
                        ? %o0 = "ENHO";
mov     2, %o1
set     0x270, %g1
ld      [%17+%g1], %o2 ? //%g=0x270, %17+%g1=5E7D8, %o2=3E940, __FUNCTION__7525
                        ? %o2 = "enhoxap_functions_execute_step_and_update_results_cb";
set     0x128, %g1
ld      [%17+%g1], %o3 ? //%g=0x128, %17+%g1=5E690, %o3=3D848, _LLC61
                        ? %o3 = "> (step_id = %d)";
call    _THXAttrace
mov     %i1, %o4
```



```
int enhoxap_functions_execute_step_and_update_results_cb(int i0, int step_id)
{
```

...

```
THXAttrace("ENHO", 2,
__FUNCTION__, //"enhoxap_functions_execute_step_and_update_results_cb",
"> (step_id = %d)",
step_id);
```

入口trace打印该函数的输入参数，  
根据trace信息确定输入参数名及类型。

# 出口trace

```
loc_15390:                                ! CODE XREF: enhoxap_functions_execute_step_and_update_results_
                                           ! enhoxap_functions_execute_step_and_update_results_cb+17C↑j ..
set     0x278, %g1
ld      [%17+%g1], %o0
mov     2, %o1
set     0x270, %g1
ld      [%17+%g1], %o2 ! //g=0x270, %17+%g1=5E7D8, %o2=3E940, __FUNCTION__7525
                                           ! %o2 = "enhoxap_functions_execute_step_and_update_results_cb";
set     0x28C, %g1
ld      [%17+%g1], %o3 ! //g=0x28C, %17+%g1=5E7F4, %o3=3C928, _LLC5_0
                                           ! %o3 = "< () = %R";
call    _THXAtrace
ld      [%fp+iErrorCode], %o4
ld      [%fp+iErrorCode], %i0
return  %i7+8
nop
```



```
int enhoxap_functions_execute_step_and_update_results_cb(int i0, int step_id)
{
    ...

    THXAtrace("ENHO", 2,
        __FUNCTION__, //"enhoxap_functions_execute_step_and_update_results_cb",
        "> (step_id = %d)",
        step_id);

    ...

    THXAtrace("ENHO", 2,
        __FUNCTION__, //"enhoxap_functions_execute_step_and_update_results_cb",
        "< () = %R",
        iErrorCode);
    return iErrorCode;
}
```

出口trace打印该函数的输出参数及返回值，  
这里trace只打印了函数返回值，%R专用于打印iErrorCode。

## • ERXA接口调用解析

此接口类似于C语言的**sprintf**函数，支持不确定个数参数。通常为下面的记录错误信息接口服务，事先构造好错误描述信息。

//构造错误描述信息字符串

```
char *ERXAmakeContext(const char *psFormat, ...);
```

//记录错误信息

```
void ERXalogExceptionSingleLink(int iErrorCode,  
    int *piErrorLink,          //0  
    const char *psFile,        //文件名, 解析时用__FILE__  
    int iLine,                  //当前行号, 解析时用__LINE__  
    char *psSccsId,             //0  
    char *psContext);           //通常为ERXAmakeContext的返回值
```

//记录错误信息, 执行本函数后\*piErrorCode赋值为iNewErrorCode.

```
void ERXalogExceptionSingleLink_id(int iNewErrorCode,  
    int iErrorCode, int *piErrorCode, //这两个参数通常为调用函数的返回值, 一个传值, 一个传地址  
    const char *psFile,               //文件名, 解析时用__FILE__  
    int iLine,                         //当前行号, 解析时用__LINE__  
    char *psSccsId,                   //0  
    char *psContext);                 //通常为ERXAmakeContext的返回值
```

```

set    0x180, %g1
call   _ERXAmakeContext
ld     [%17+%g1], %o1  ! //g=0x180, %17+%g1=5E6E8, %o1=3DC98, _LLC83
                        ! %o1 = "DQXA_get_values( ENH0xAP_functions_dq_handle_ptr,
                        ! ENH0_TC_STRUCT_DQ_NAME, (void *)&tst_const)";

st     %o0, [%sp+0x5C]
set    0x45520072, %o0
ld     [%fp+iErrorCode], %o1
add    %fp, iErrorCode, %o2
set    0x288, %g1
ld     [%17+%g1], %o3  ! //g=0x288, %17+%g1=5E7F0, %o3=3C910, _LLC4_0
                        ! %o3 = "ENH0xAP_functions.c";

mov    0x436, %o4
call   _ERXAllogExceptionSingleLink_id
mov    0, %o5

```



```

int iErrorCode = 0;
char *psErrorText = NULL;

```

...

```

psErrorText = ERXAmakeContext("Calling %s resulted in an error.",
    "DQXA_get_values( ENH0xAP_functions_dq_handle_ptr, ENH0_TC_STRUCT_DQ_NAME, (void *)&tst_const)");
ERXAllogExceptionSingleLink_id(0x45520072,
    iErrorCode, &iErrorCode,
    __FILE__,    //"ENH0xAP_functions.c",
    __LINE__,    //0x436,
    0,
    psErrorText);

```

通过错误描述信息，可以推断函数执行的目的，有时也能推断出变量名及类型。

## • 充分利用IDA中的字符串信息去解析

通过字符串信息去推断函数名，文件名，结构体名，变量名，变量类型。

```
set     0xDC, %g1
ld      [%17+%g1], %o0    ! //g=0xDC, %17+%g1=5E644, %o0=3D3A8, _LLC43
                        ! %o0 = "ENHO:result_struct";
call    _DDXAxOBJECT_full_create
add     %fp, var_8, %o1
cmp     %o0, 0
be      %icc, loc_15508
st      %o0, [%fp+var_4]
set     0x29C, %g1
ld      [%17+%g1], %o0    ! //g=0x29C, %17+%g1=5E804, %o0=3C970, _LLC9_0
                        ! %o0 = "Calling %s resulted in an error.";

set     0x1A8, %g1
call    _ERXAmakeContext
ld      [%17+%g1], %o1    ! //g=0x1A8, %17+%g1=5E710, %o1=3DE40, _LLC92
                        ! %o1 = "DDXAxOBJECT_full_create( ENHO_RESULT_STRUCT_STR, (void**) &result_struct_ptr)";

st      %o0, [%sp+0x70+var_14]
set     0x45520072, %o0
ld      [%fp+var_4], %o1
add     %fp, var_4, %o2
set     0x288, %g1
ld      [%17+%g1], %o3    ! //g=0x288, %17+%g1=5E7F0, %o3=3C910, _LLC4_0
                        ! %o3 = "ENHOxAP_functions.c";

mov     0x497, %o4
call    _ERXAllogExceptionSingleLink_id
mov     0, %o5
```

结构体: ENHO\_result\_struct

call ...error  
嗯，调用某函数发生错误

函数DDXAxOBJECT\_full\_create有两个参数：  
一个是表示" ENHO\_result\_struct"的字符串  
一个是void \*\*result\_struct\_ptr

该函数所在文件名: "ENHOxAP\_functions.c"

## • 结构体名字的确定

解析对象会涉及到一些DDXA接口的接口调用，这些接口第一个参数就是结构体名字的字符串，后面的参数是对应的结构体变量指针（一维/二维）

```
int DDXA_C_object_full_alloc(char *def_name, void **C_object_p);  
int DDXA_C_object_sub_alloc(char *def_name, void *C_object_p);  
int DDXA_C_object_sub_free(char *def_name, void *C_object_p);  
int DDXA_C_object_full_free(char *def_name, void **C_object_p);  
int DDXAxOBJECT_full_create(char *def_name, void **C_object_p);  
int DDXAxOBJECT_full_destroy(char *def_name, void **C_object_p);  
int DDXAxOBJECT_sub_destroy(char *def_name, void *C_object_p);  
int DDXAxOBJECT_clone(char *def_name, void *original, void **clone);
```

结构体类型名

结构体变量指针（一维/二维）



通常传的第三个参数%00是一个带 “:” 的字符串，对应的结构体名把 “:” 替换成 “\_”，如下图，我们可判断出var\_28/var\_2C/var\_30均为一个结构体指针以及对应的结构体名：

```
.text:0005B104      add     %fp, var_28, %i4
.text:0005B108      set     0x1934, %g1
.text:0005B110      ld      [%i7+%g1], %o0    ! //%g=0x1934, %i7+%g1=18417C, %o0=14E6E0, _LLC2_12
.text:0005B110                                     ! %o0 = "KVMAxLOG:zmap_grid_table;
.text:0005B114      call    DDXAxOBJECT_full_create
.text:0005B118      mov     %i4, %o1
.text:0005B11C      cmp     %o0, 0
.text:0005B120      bne     %icc, loc_5B7A4
.text:0005B124      mov     %o0, %i0
.text:0005B128      set     0x16E4, %g1
.text:0005B130      ld      [%i7+%g1], %o0    ! //%g=0x16E4, %i7+%g1=183F2C, %o0=14E7E8, _LLC10_10
.text:0005B130                                     ! %o0 = "KVMAxLOG:pdgc_grid_table;
.text:0005B134      call    DDXAxOBJECT_full_create
.text:0005B138      add     %fp, var_2C, %o1
.text:0005B13C      cmp     %o0, 0
.text:0005B140      bne, pn %icc, loc_5B7A4
.text:0005B144      mov     %o0, %i0
.text:0005B148      set     0x1710, %g1
.text:0005B150      ld      [%i7+%g1], %o0    ! //%g=0x1710, %i7+%g1=183F58, %o0=14E9B0, _LLC21_9
.text:0005B150                                     ! %o0 = "KVMAxLOG:valid_spots;
.text:0005B154      call    DDXAxOBJECT_full_create
.text:0005B158      add     %fp, var_30, %o1
.text:0005B15C      cmp     %o0, 0
```

```
KVMAxLOG_zmap_grid_table *var_28;
KVMAxLOG_pdgc_grid_table *var_2C;
KVMAxLOG_valid_spots *var_30;

DDXAxOBJECT_full_create("KVMAxLOG:zmap_grid_table", &var_28);
DDXAxOBJECT_full_create("KVMAxLOG:pdgc_grid_table", &var_2C);
DDXAxOBJECT_full_create("KVMAxLOG:valid_spots", &var_30);
```

- **格式化输出**

%d :对应一个int或enum或bool

%p : 对应一个指针

%f %lf : 对应一个double或float

%D : 对应三个量：字符串指针，变量地址

## • 一个idb文件可解析成哪些.c文件？



ENHO.idb

```
:
! Source File : './bld/ENHO'
! Source File : 'values-Xa.c'
! Source File : 'crtstuff.c'
! Source File : 'ENHOxAP_startupshutdown.c'
! Source File : 'ENHOxAP_functions.c'
! Source File : 'ENHOxAP_define_scans.c'
! Source File : 'ENHOxAP_driftcorrection.c'
! Source File : 'ENHOxAP_scenarios.c'
! Source File : 'ENHOxAP_measurements.c'
! Source File : 'ENHOxAP_moves.c'
! Source File : 'ENHOxAP_platemap.c'
! Source File : 'ENHOxAP_files.c'
! Source File : 'ENHOxAP_graphics.c'
! Source File : 'ENHOxAP_model.c'
! Source File : 'id.c'
! Source File : 'libgcc2.c'
! Source File : 'unwind-dw2.c'
! Source File : 'unwind-dw2-fde.c'
! Source File : 'crtstuff.c'

! Processor      : sparcb
! Target assembler: GNU assembler
```

在IDA中开头 “Source File” 部分以  
模块名开头的.c。

- 一个函数该放在哪个.c中？

通过ERXA的接口参数判断。

```
set    0x280, %g1
ld     [%17+%g1], %o0    ! //g=0x280, %17+%g1=5E7E8, %o0=3C8C8, _LLC2_0
                        ! %o0 = "Verification by %s failed: '%s' not TRUE";

set    0x268, %g1
ld     [%17+%g1], %o1    ! //g=0x268, %17+%g1=5E7D0, %o1=3E6D8, __FUNCTION__6966
                        ! %o1 = "ENH0xAP_functions_get_callbacks_cb";

set    0x284, %g1
call   _ERXAmakeContext
ld     [%17+%g1], %o2    ! //g=0x284, %17+%g1=5E7EC, %o2=3C8F8, _LLC3_0
                        ! %o2 = "(cb_ptr != NULL)";

st     %o0, [%sp+0x5C]
set    0x454E4D00, %o0
mov     0, %o1
add     %fp, var_4, %o2
set    0x288, %g1
ld     [%17+%g1], %o3    ! //g=0x288, %17+%g1=5E7F0, %o3=3C910, _LLC4_0
                        ! %o3 = "ENH0xAP_functions.c";

mov     0xBF, %o4
call   _ERXAllogExceptionSingleLink_id
mov     0, %o5
```

若函数内无ERXA接口调用，则把它放在IDA中上一个或下一个函数放在同一个.c中。

若实在推测不出合适的.c，则建立一个与模块名同名的.c，放在里面。

## • 一个.idb文件需要定义哪些.h文件？

一般需定义一个CCXX.h与CCXX\_extern.h。（CCXX为模块名）

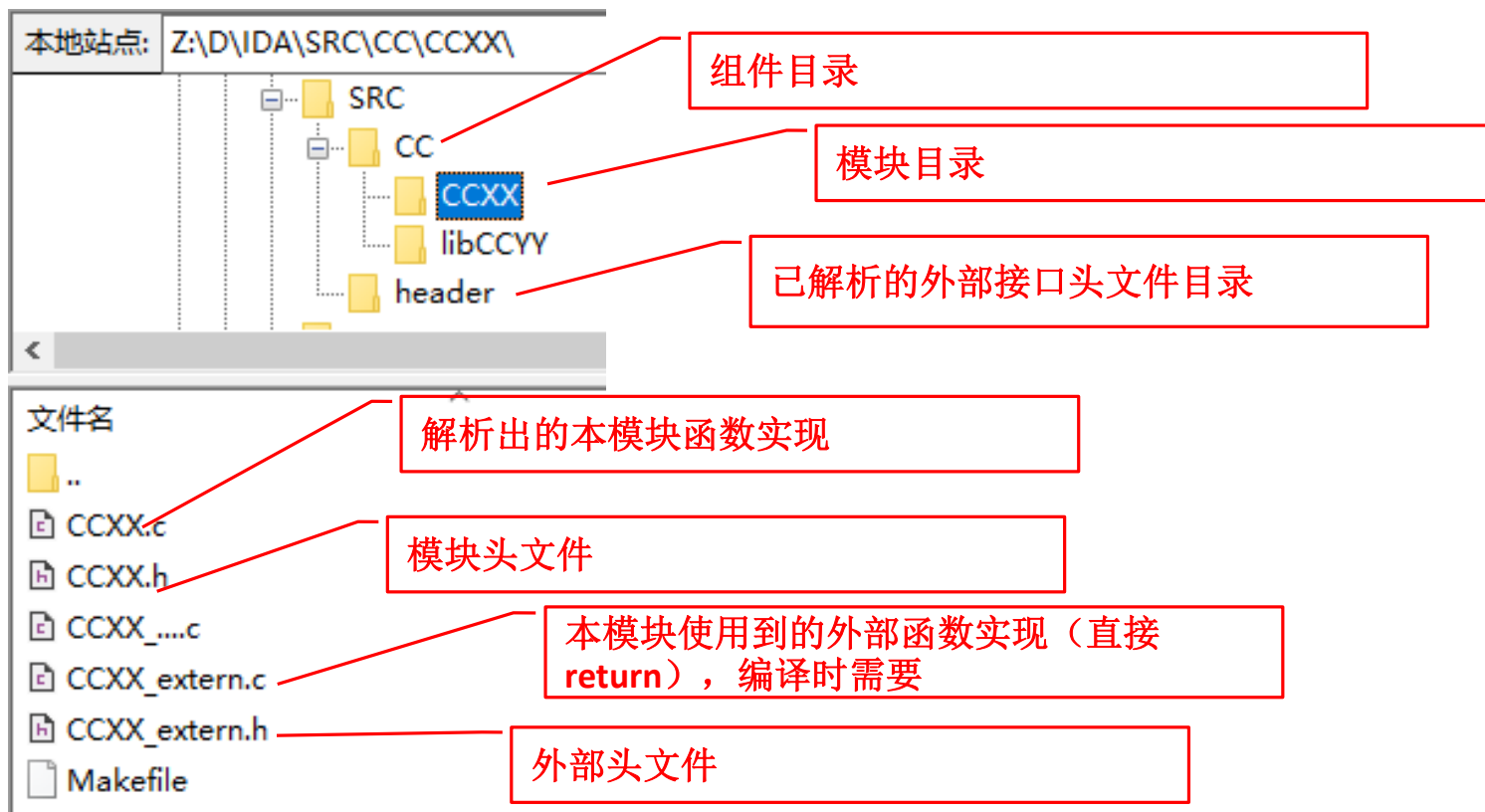
CCXX.h：定义与本模块内的数据结构、函数声明（名字一般以本模块名开头）

CCXX\_extern.h：定义与本模块会使用到的且未在header文件夹找到定义的外部数据结构、函数声明（名字以其他模块名开头）

代码组织：




SRC.zip



# • Cygwin下编译

## 1.修改makefile文件



The screenshot shows a text editor window titled 'Makefile'. The content of the file is as follows:

```
1 SOURCE = CCYY.c\  
2   CCYY_extern.c  
3  
4  
5 HEADER = CCYY.h\  
6   CCYY_extern.h  
7  
8  
9 INCPATH = -I../../header -I.  
10  
11  
12 LIBRARY = -L../../IT/lib  
13  
14  
15 all:CCYY  
16 CCYY:$(SOURCE) $(HEADER)  
17     gcc -shared -fPIC -w $(INCPATH) $(LIBRARY) -o libCCYY.so $(SOURCE)  
18
```

Red boxes and arrows highlight specific parts of the file:

- A red box around lines 1-2 is labeled '待编译的源文件' (Source files to be compiled).
- A red box around lines 5-6 is labeled '待编译的头文件' (Header files to be compiled).
- A red box around line 9 is labeled '头文件搜索路径' (Header file search path).
- A red box around line 15 is labeled '编译输出' (Compilation output).
- A red box around the output file 'libCCYY.so' in line 17 is also labeled '编译输出'.

注：这里只是简单的makefile写法，如果你精通makefile写法，也可根据需要自行撰写。

# • Cygwin下编译

## 2.Cygwin编译

```
jiangcl@DESKTOP-S10101 /cygdrive/z/D/IDA/SRC/CC/libCCYY
$ cd SRC/CC/libCCYY

jiangcl@DESKTOP-S10101 /cygdrive/z/D/IDA/SRC/CC/libCCYY
$ ls
CCYY.c  CCYY.h  CCYY_....c  CCYY_extrn.c  CCYY_extrn.h  Makefile

jiangcl@DESKTOP-S10101 /cygdrive/z/D/IDA/SRC/CC/libCCYY
$ make
gcc -shared -fPIC -w -I../..header -I. -L../..../IT/lib -o libCCYY.so CCYY.c CCYY_extrn.c

jiangcl@DESKTOP-S10101 /cygdrive/z/D/IDA/SRC/CC/libCCYY
$ ls
CCYY.c  CCYY.h  CCYY_....c  CCYY_extrn.c  CCYY_extrn.h  Makefile  libCCYY.so

jiangcl@DESKTOP-S10101 /cygdrive/z/D/IDA/SRC/CC/libCCYY
$ _
```

进入待编译的代码目录

make执行编译

编译输出

Thank you !