
Juphoon Protocol Framework

Zero Operating System

Date of Issue: February 26, 2005

Please visit our website at <http://www.juphoon.com>

Juphoon ZOS Service Guide

Juphoon System Software

For information, visit: <http://www.juphoon.com>

Tel: +86-574-87287820

Fax: +86-574-87304379

Document Number:

Copyright © 2008, Juphoon System Software Corporation.

All Rights Reserved

Contents

1. INTRODUCTION	5
1.1 PURPOSE	5
1.2 READERS	5
1.3 WHAT'S IN THIS DOCUMENT	5
1.4 ABBREVIATIONS	5
2. CONCEPTS	5
2.1 ZOS CONTEXT	5
2.2 OVERVIEW OF ZOS MODULES	6
2.3 BASIC DATA TYPES	7
2.4 PLATFORM TYPES	7
2.5 BASIC CONCEPTS	8
2.5.1 <i>Module ID</i>	8
2.5.2 <i>Instance ID</i>	8
2.5.3 <i>Task ID</i>	8
2.5.4 <i>Processor ID</i>	8
3. MODULE AND TASK	9
3.1 TASK INTRODUCTION	9
3.2 TASK START	10
3.3 TASK INITIALIZATION	11
3.4 TASK DESTROY	11
3.5 MESSAGE PROCESS	11
4. MESSAGE	12
4.1 MESSAGE INTRODUCTION	12
4.2 MODULE SERVICE POST	12
4.3 SENDING MESSAGE	13
4.4 PROCESSING MESSAGE	14
5. TIMER MANAGEMENT	15
5.1 TIMER INTRODUCTION	15
5.2 ZOT TIMER	15
5.3 TIMER INTERFACE	16
6. MEMORY MANAGEMENT	17
6.1 MEMORY MANAGEMENT INTRODUCTION	17
6.2 CREATE AND DELETE	18
6.3 OTHER INTERFACES	19
6.4 EXMAPLE	20
7. DATA BUFFER MANAGEMENT	22
7.1 DATA BUFFER INTRODUCTION	22

7.2	DATA BUFFER	22
7.3	DATA BUFFER INTERFACES.....	24
7.4	DATA BUFFER EXAMPLE.....	24
8.	ZOS CONFIGURATION	25
8.1	CONFIGURATION	25
8.1.1	General Config.....	26
8.1.2	Memory Pool Config.....	27
8.1.3	Message Pool Config	27
8.1.4	Data Buffer Pool Config	28
8.1.5	Log Config.....	28
8.1.6	Module Config.....	30
8.1.7	Task Config	30
8.1.8	Timer Config	30
9.	CREATE APPLICATION.....	31
9.1	MODULE ID AND TASK ID	31
9.2	HEADER FILES.....	33
9.3	ZOS INITIALIZATION.....	33
9.4	DESIGN USER TASK	33
9.4.1	Using Module Functions	33
9.4.2	Using Zos_TaskSpawn	35
9.5	COMPILE	36
10.	APPENDIX	37
10.1	ZOS COMPILE OPTIONS	37
10.2	ZOS MACRO DEFINITION.....	38
10.3	ZOS MACRO FUNCTION.....	40
10.4	ZOS FUNCTION PROTOTYPE	43
10.5	ZOS ERROR NO	45
10.6	ZOS LOG LEVEL FUNCTION	50
10.7	ZOS LIST MACRO FUNCTION.....	50
10.8	ZOS DOUBLE LINKED LIST MACRO FUNCTION	54

List of Tables

Table 1-1	Abbreviations.....	5
Table 2-1	Basic data types.....	7
Table 8-1	General Config.....	27
Table 8-2	Memory Pool Config	27
Table 8-3	Message Pool Config	28
Table 8-4	Data Buffer Pool Config	28
Table 8-5	Log Config	29
Table 8-6	Module Config.....	30
Table 8-7	Task Config.....	30

Table 8-8 Timer Config..... 31

List of Figures

Figure 2-1 ZOS context..... 6
Figure 2-2 ZOS modules 6
Figure 3-1 Multiple ZOS Task depend on one OS Task 9
Figure 3-2 Each ZOS Task depend on one OS Task..... 9
Figure 4-1 ZOS Message between Tasks 12
Figure 5-1 ZOS Timer Task Driven 16
Figure 6-1 ZOS Memory Management 18
Figure 7-1 Data Buffer Structure..... 22

1. Introduction

ZOS (Zero Operating System) is service platform over operating system provided by Juphoon. It implement uniform interfaces supporting many operating system, which made it possible to build product independent with CPU, compiler and operating system. ZOS also implement enhanced function beyond system service, like task management, message queue, timer, memory management, data buffer management, log, and many protocol oriented interfaces, as ABNF, ASN.1 codec library.

1.1 Purpose

This document is for the developer using ZOS to build their product.

1.2 Readers

Readers of this document are assumed to have knowledge of C language and operating systems.

1.3 What's in this document

This document covers ZOS their usage, detail on how to create program based on ZOS.

1.4 Abbreviations

Table 1-1 includes all the abbreviations used in this document.

Abbreviation	Description
ANSI	American National Standards Institute
ZOS	Zero Operation System
ABNF	Augmented BNF
DBUF	DBUF
RTIMER	Ring Timer

Table 1-1 Abbreviations

2. Concepts

2.1 ZOS context

ZOS(Zero Operating System) is Juphoon's service platform for operating systems. It provides united abstract interfaces which support a variety of operating systems, making software products independent of processors, compilers and operating systems. In addition, ZOS also provides functions as Task Management, Message Queue, Timer Management, Memory Management, Data Buffer, Log Management and etc. It abstracts many procolot-related service interfaces, including ABNF Codec and ASN.1 Codec.

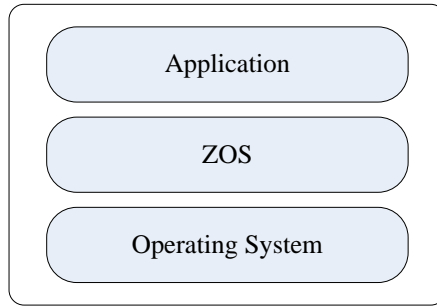


Figure 2-1 ZOS context

An operating system may be Windows, Linux, Solaris, VxWorks, pSOS or threadX.

2.2 Overview of ZOS modules

Figure 2-2 shows all ZOS modules.

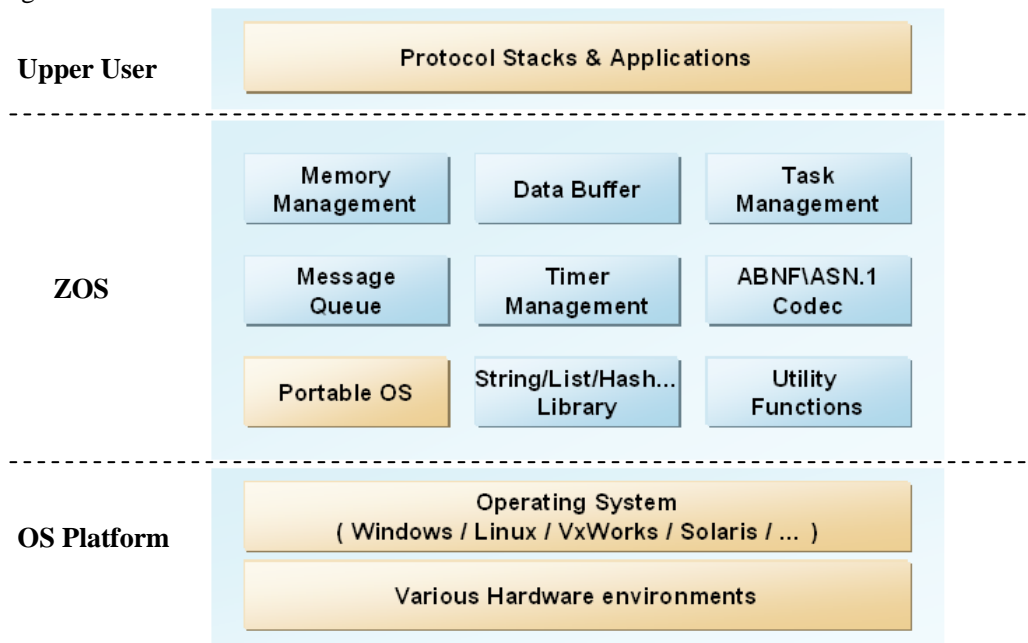


Figure 2-2 ZOS modules

Table 2-1 describes the functions of ZOS modules.

Module	Function
Memory Management	Responsible for memory management, eluding frequent memory applications and memory releases when upper applications are running, and providing object-oriented management which allows users to define several memory management instances.
Data Buffer	Aiming to reduce repeated memory copyings and manage memory with a large amount of flexibility, providing rules on memory usage in a bid to avoid memory leaks and invalid memory pointers.

Task Management	Providing a message-triggered task managing mechanism.
Message Queue	Providing a message queue mechanism.
Timer Management	Being able to meet different demands for timer mechanisms with different precisions and scales and providing practical timer mechanisms (including the timer precision, occupying indexes of system resources, etc.) to meet the requirements for hundreds of thousands of concurrent timers.
ABNF\ASN.1 Codec	A relatively independent module for processing protocol packages, which needs to provide a efficient coding and decoding mechanism. At present only ABNF Codec has been completed and ASN.1 is not available yet.
Basic Algorithms Module	Including basic algorithms as String, List, Hash, providing streamlineand efficient algorithms in C language environment.
Utility Functions Module	Providing tools as dump, log, gab, Fsm Map and etc.
Portable OS Module	Seal the OS-dependent modules including Mutex, Socket, Interrupt, Task Establish and Time.

Table 2-1 ZOS module functions

2.3 Basic data types

ZOS platform provides a group of basic data types listed in Table 2-1.

Data type	Prototype
ZDOUBLE	double
ZFLOAT	float
ZLONG	long
ZINT	int
ZSHORT	short
ZCHAR	char
ZULONG	unsigned long
ZUINT	unsigned int
ZSIZE_T	unsigned int
ZUSHORT	unsigned short
ZUCHAR	unsigned char
ZBOOL	boolean
ZVOID	void

Table 2-1 Basic data types

2.4 Platform types

Table 2-2 lists a group of basic platform types.

Platform type	Type
ZMUTEX	Mutex
ZSEM	Semaphore
ZTIME_T	Time
ZFUNCPTR	Function Pointer
ZVOIDFUNCPTR	Void Function Pointer
ZLOGID	Log ID
ZMODID	Module ID
ZINSTID	Instance ID
ZTASKID	Task ID
ZTIMERID	Timer ID
ZEVENTID	Event ID
ZPOOLID	Pool ID

Table 2-2 Platform types

2.5 Basic concepts

This section introduces four common ID structures used in ZOS and these structures will be referred in specific interfaces.

2.5.1 Module ID

Module ID is a 16-bit unsigned integer. There are two types of module, ZOS basic modules and modules defined by users.

2.5.2 Instance ID

A module may be shared by multiple instances with each instance identified by an ID. Like Module ID, Instance ID is a 16-bit unsigned integer, too.

2.5.3 Task ID

Task ID, a 32-bit unsigned integer, is used to distinguish between different tasks. Its 16 leftmost bits represent a Module ID, the rest bits (the rightmost 16 bits) an Instance ID.

2.5.4 Processor ID

An operating system may be running on more than one processor. So it is necessary that each processor should have an ID for the operating system to distinguish it from the other processors when they are communicating with each other. Distributed operating systems have not been supported.

3. Module and Task

3.1 Task Introduction

In ZOS, Task is a module instance. It is different from the task of operating system. But ZOS Task is dependent on OS Task. When implement, one OS Task can drive one ZOS Task, and one OS Task can drive multiple ZOS Tasks.

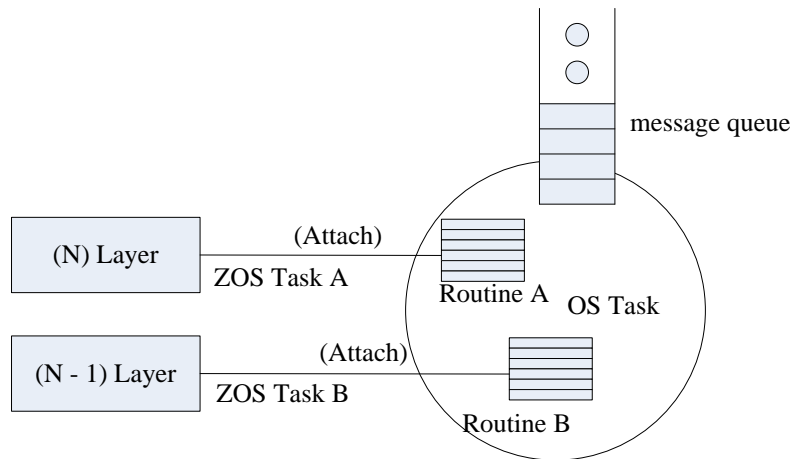


Figure 3-1 Multiple ZOS Task depend on one OS Task

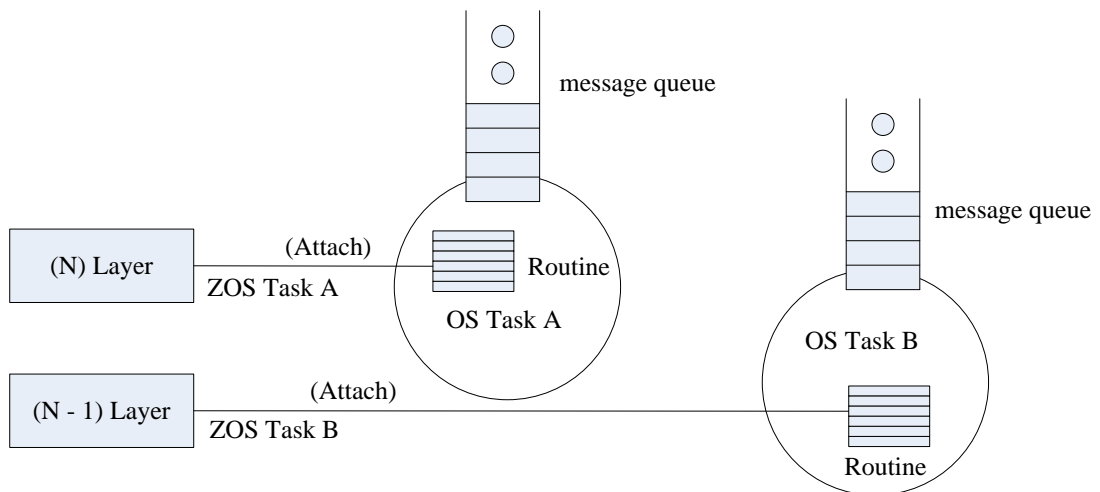


Figure 3-2 Each ZOS Task depend on one OS Task

ZOS Task basic functions include::

- Initialization
Initialize module instance.
- Message Process
Process message from other module instance.
- Timer Process
Process the time out event of the timer.

3.2 Task Start

There are 2 ways to start a ZOS Task:

- Call **Zos_TaskSpawn**

In this case, it will call task creation function of OS directly (for example, it calls `pthread_create` in Linux). And this ZOS task became a scheduling unit of OS.

```
/* task entry function */
typedef ZULONG (*PFN_ZTASKENTRY)(ZVOID *);

/* zos spawn task */
ZINT Zos_TaskSpawn(ZTASKID zTaskId, ZCHAR *pcName, ZINT iPriority,
                  ZULONG dwStackSize, ZULONG dwQueueSize, ZULONG dwTimerNum,
                  PFN_ZTASKENTRY pfnEntry, ZVOID *pParm);
```

- Using module function in ZOS

The ZOS module wraps the **Zos_TaskSpawn**, and provide some functions to made it more easy to use.

When task initialization, it will callback **PFN_ZINSTINIT**.

When task destroy, it will callback **PFN_ZINSTDESTROY**.

When task process message, it will callback **PFN_ZINSTMSGPROC**.

These callbacks will discuss on following sections. Here is an example on how to start a ZOS task using module function:

```

/* zos instance initialize function */
typedef ZINT (*PFN_ZINSTINIT)(ZINSTID zInstId);

/* zos instance destroy function */
typedef ZVOID (*PFN_ZINSTDESTROY)(ZINSTID zInstId);

/* zos instance message process function */
typedef ZINT (*PFN_ZINSTMSGPROC)(ZVOID *pMsg);

/* zos register one module */
#define ZOS_MOD_REG(_taskid, _name, _queuesize, _timernum, \
                    _pfninit, _pfnDestroy) \
    Zos_ModReg(_taskid, _name, _queuesize, _timernum, \
               (PFN_ZINSTINIT)_pfninit, (PFN_ZINSTDESTROY)_pfnDestroy)

/* zos deregister one module */
#define ZOS_MOD_DEREG(_taskid) \
    Zos_ModDereg(_taskid)

/* zos start one module */
#define ZOS_MOD_START(_taskid, _priority, _stacksize, _pfnmsgproc) \
    Zos_ModStart(_taskid, _priority, _stacksize, (PFN_ZINSTMSGPROC)_pfnmsgproc)

```

3.3 Task Initialization

PFN_ZINSTINIT is the prototype of task initialization function. When ZOS module manager start the task, it will call this function with Instance ID. It should initialize the configuration, allocate memory and other resources used in this task.

For example, SIP protocol task initialization function is:

```

/* sip task init */
ZINT Sip_TaskInit(ZINSTID zInstId)

```

3.4 Task Destroy

PFN_ZINSTDESTROY is the prototype of task destroy function. When ZOS module manager is about to stop the task, it will call this function. It should destroy all the resources used in this task.

3.5 Message Process

PFN_ZINSTMSGPROC is the prototype of task message process function. The message processed in this function is not the OS Message, but the ZOS Message, which may sent by other ZOS Task or from itself.

For example, SIP task message process function is:

```

/* sip task message process */
ZINT Sip_TaskMsgProc(ST_ZOS_MSG *pstMsg)

```

4. Message

4.1 Message Introduction

The communication between ZOS task was driven by ZOS Message. The ZOS message has no relation with OS message, like Windows Message, and is independent on OS message. Message driven mechanism is suitable for communication between layered modules, and it serializes the event processing. Program focus on the message it received, include:

- System Message, like time out message
- Upper Layer Message
- Lower Layer Message

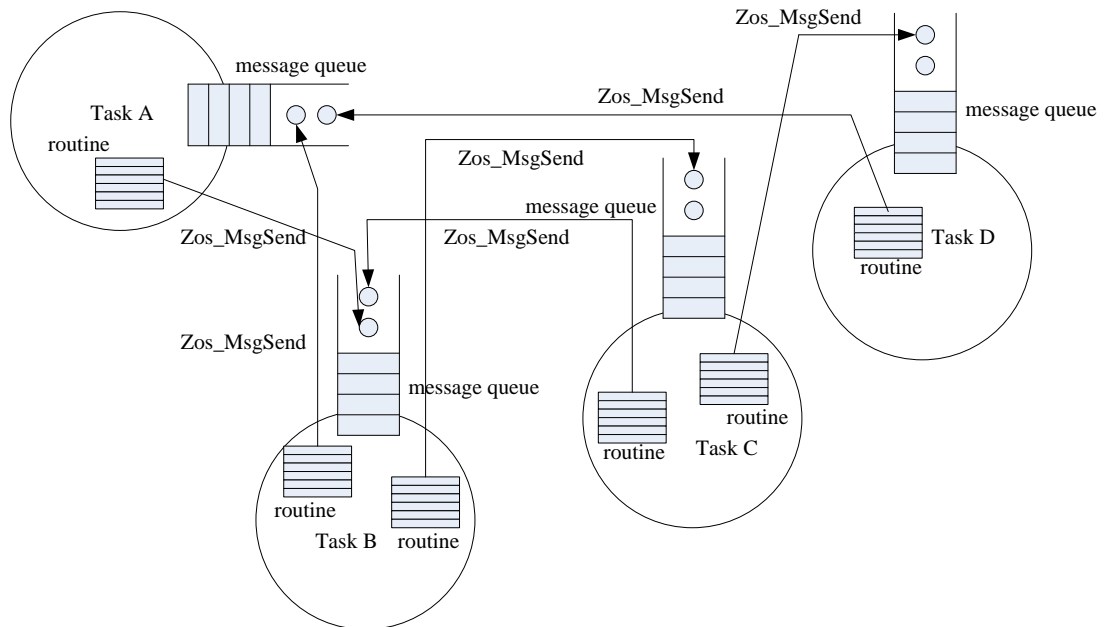


Figure 4-1 ZOS Message between Tasks

4.2 Module Service Post

When initialization, each task will assigned with a message queue to store the message received. To send ZOS message, it should compose message header structure as below:

```
/* zos module service post */
typedef struct tagZOS_MSP
{
    ZCPUID zSendCpuId;          /* sender cpu id */
    ZTASKID zSendTaskId;       /* sender task id */
    ZCPUID zRecvCpuId;         /* receiver cpu id */
    ZTASKID zRecvTaskId;       /* receiver task id */
    ZEVNTID zEvtId;            /* event id */
    ZPOOLID zPoolId;           /* memory pool id */
} ST_ZOS_MSP;

/* zos message */
typedef struct tagZOS_MSG
{
    ST_ZOS_MSP stMsp;          /* module service post */
    ZULONG dwMsgLen;           /* message length */
    ZULONG dwDataLen;          /* data length beside message header */
} ST_ZOS_MSG;
```

These data was for ZOS message manager to identify the sender and receiver. And it can also be used by receiver.

The message body should copy to the memory right after **ST_ZOS_MSG** structure which is allocate by **Zos_MsgAlloc**. And the receiver can get it by macro **ZOS_MSG_GET_DATA**.

4.3 Sending Message

For example, SIP transport layer sending message to upper layer when receiving SIP data:

```
/* sip transport data indication */
ZINT Sip_TptDataInd(ST_SIP_MSG_EVT *pstMsgEvt)
{
    ST_ZOS_MSG *pstSysMsg;
    ST_SIP_MSG_EVT *pstEvt;
    ST_ZOS_MSP stMsp;

    /* set upper layer task info */
    stMsp.zSendCpuId = ZCPUID_LOCAL;
    stMsp.zSendTaskId = ZTASKID_SIP_TPT;
    stMsp.zRecvCpuId = ZCPUID_LOCAL;
    stMsp.zRecvTaskId = ZTASKID_SIP;
    stMsp.zEvtId = pstMsgEvt->ucEvtType;
    stMsp.zPoolId = SIP_POOLID;

    /* alloc system message */
    pstSysMsg = Zos_MsgAlloc(&stMsp, sizeof(ST_SIP_MSG_EVT));
    if (pstSysMsg == ZNULL)
        return ZFAILED;

    /* set message event */
    pstEvt = (ST_SIP_MSG_EVT *)ZOS_MSG_GET_DATA(pstSysMsg);
    Zos_MemCpy(pstEvt, pstMsgEvt, sizeof(ST_SIP_MSG_EVT));

    /* print buffer for debug */
    if (pstEvt->pstMsgBuf)
        Sip_LogDbuf(pstEvt->pstMsgBuf);

    /* send message */
    if (Zos_MsgSend(pstSysMsg) != ZOK)
    {
        Zos_MsgFree(pstSysMsg);
        return ZFAILED;
    }

    return ZOK;
}
```

4.4 Processing Message

For example, SIP protocol task processing message from other task and timer:

```

/* sip task message process */
ZINT Sip_TaskMsgProc(ST_ZOS_MSG *pstMsg)
{
...
    /* process message from sender */
    switch (pstSysMsg->stMsp.zSendTaskId)
    {
    case ZTASKID_TIMER:
        /* process zos timer message */
        Sip_TmrMsgProc(...);
        break;
    case ZTASKID_SIP_TPT:
        /* process transport message */
        Sip_TptMsgProc(...);
        break;
    case ZTASKID_SUA:
        /* process sip user agent session message */
        Sip_CoreMsgProc(pstSessEvnt);
        break;
    default:
        /* received unknown event from unknown task */
        break;
    }

    return ZOK;
}

```

5. Timer Management

5.1 Timer Introduction

In communication system, timer can be used in following situations:

- Periodic functions in protocol task or system task. Like MG (Media Gateway) send MGCP register message to MGC (Media Gateway Controller) periodically.
- To start corresponding process works after a certain time. For example, one task request some data and it should start error handling after a certain time without receiving responds.

5.2 ZOT Timer

Timer types:

- Cycle Timer: **ZTIMER_MODE_CYCLE**
- Noncycle Timer: **ZTIMER_MODE_NOCYCLE**
- 100ms Timer: **ZTIMER_MODE_100MS**

Cycle timer will restart automatically after time out. And it only stops when stop interface called or the timer was deleted.

Non-cycle timer will not restart after time out. Start interface must be called again, if user want to restart the timer.

100ms timer is high resolution timer. It usually used by ZOS module. This type of timer is non-cycle.

In ZOS module, there is individual task for timer. It responds for timer management. The default clock ticktock is 100ms. This interval can change by `g_stZosSysCfg.stTimer.dwTaskDelayVal`. If the task was created with a parameter of timer count, the task itself will create a timer queue to manange these timers. And the task will register a 500ms cycle timer in ZOS module to drive the task's own timers.

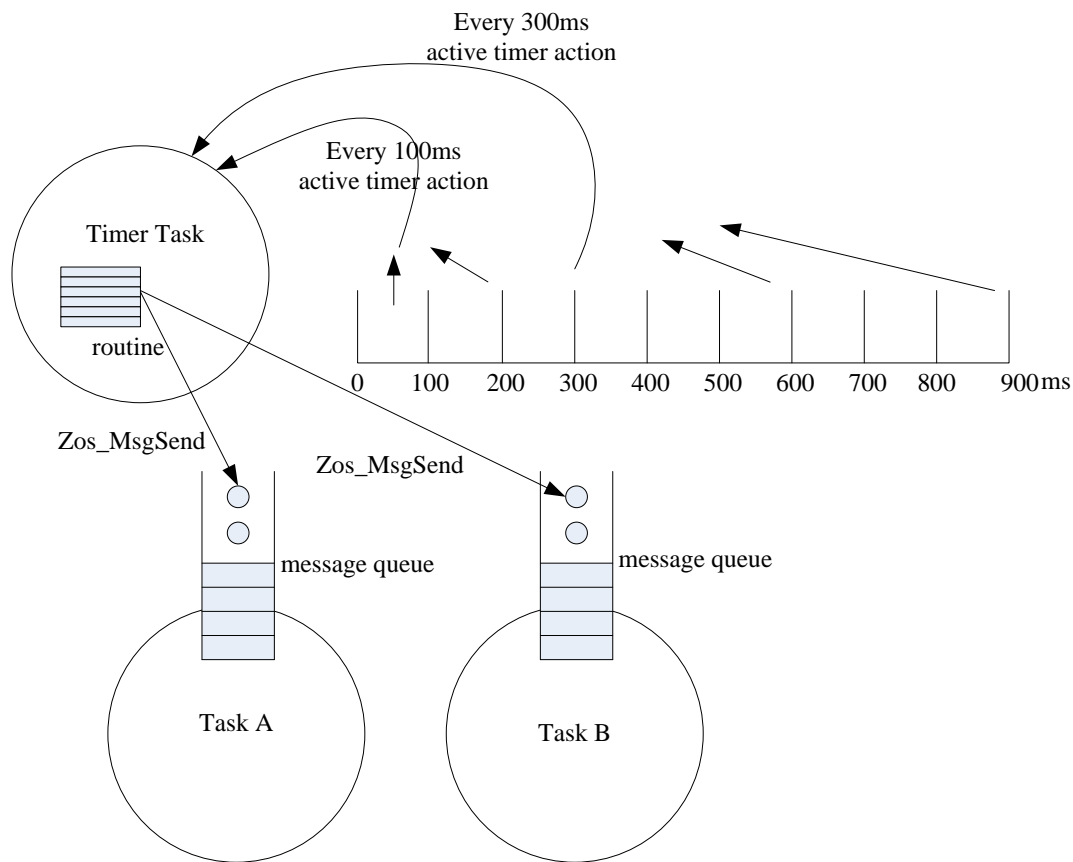


Figure 5-1 ZOS Timer Task Driven

5.3 Timer Interface

- Timer user should first implement the time out action fuction, which prototype is **PFN_ZTIMERACTIVE**:

```

/* zos timer active function for callback */
typedef ZVOID (*PFN_ZTIMERACTIVE)(ZTIMERID zTimerId, ZULONG dwTimerType,
                                   ZULONG dwParm);

```

To create a timer, call **Zos_TimerCreate**:

```

/* zos create a timer */
ZINT Zos_TimerCreate(ZTASKID zTaskId, ZUCHAR ucMode, ZTIMERID *pzTimerId);

```

To start the timer created, call **Zos_TimerStart**:

```

/* zos start a timer */
ZINT Zos_TimerStart(ZTASKID zTaskId, ZTIMERID zTimerId, ZULONG dwTimerType,
                   ZULONG dwTimeLen, ZULONG dwParm, PFN_ZTIMERACTIVE pfnActive);

```

To stop the timer already started, call **Zos_TimerStop**:

```

/* zos stop a timer */
ZINT Zos_TimerStop(ZTASKID zTaskId, ZTIMERID zTimerId);

```

To delete the timer after used, call **Zos_TimerDelete**:

```

/* delete one timer, if the timer is active, it will stop the timer */
ZINT Zos_TimerDelete(ZTASKID zTaskId, ZTIMERID zTimerId);

```

To check if the timer is running, call **Zos_TimerIsRun**:

```

/* is the timer in running state */
ZBOOL Zos_TimerIsRun(ZTASKID zTaskId, ZTIMERID zTimerId);

```

6. Memory Management

6.1 Memory Management Introduction

Although memory management is a basic function in every operating system, ZOS provides a set of memory management to conceal operating system difference.

The main object in memory management is Bucket Pool which contains many Bucket Group of different size, like 32 bytes, 64 bytes, etc. When user applies for memory, ZOS will search the most suitable bucket group and allocate memory from a free bucket.

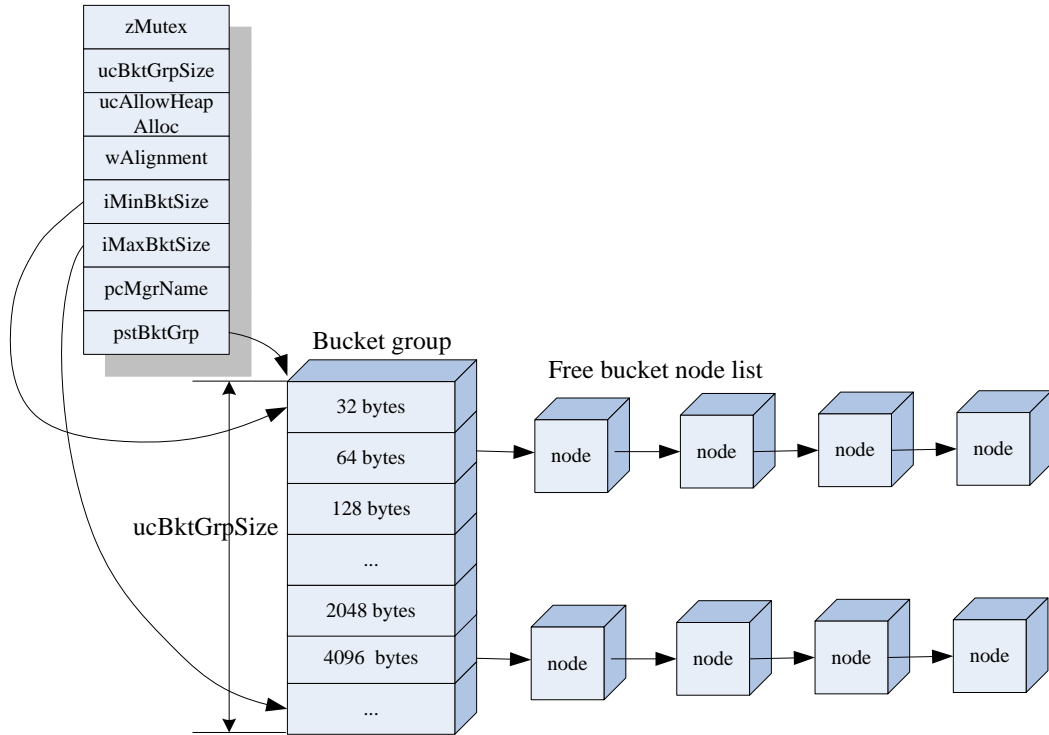


Figure 6-1 ZOS Memory Management

6.2 Create and Delete

To create and delete pool, call **Zos_PoolCreate** and **Zos_PoolDelete**:

```

/* zos create memory pool */
ZPOOLID Zos_PoolCreate(ST_ZOS_POOL_CFG *pstPoolCfg);

/* zos delete memory pool */
ZVOID Zos_PoolDelete(ZPOOLID zPoolId);

```

To a special propose, Bucket Pool can be customized by **ST_ZOS_POOL_CFG**. Following is the structure definition:

```

/* zos bucket config info */
typedef struct tagZOS_BKT_INFO
{
    ZULONG dwBktSize;           /* bucket size */
    ZULONG dwMaxCount;         /* bucket maixmum count */
    ZULONG dwIncCount;         /* bucket inc count per time */
} ST_ZOS_BKT_INFO;

/*****
zos pool config
the data structure should be init by user

+-----+
| *pcName      |
| ucIsNeedMutex | indicate the num of bkt info
| ucInfoGrpSize -----+
| aucSpare[2]  |          |
+-----+          V
| *pstInfoGrp -----> +-----+
|                | | bkt info 1 |
+-----+          | bkt info 2 |
| pfnHeapAlloc   | | ...        |
| pfnHeapFree    | | bkt info n |
+-----+          +-----+

*****/
typedef struct tagZOS_POOL_CFG
{
    ZCHAR *pcName;             /* bucket manager name */
    ZUCHAR ucIsNeedMutex;     /* is need mutex */
    ZUCHAR ucInfoGrpSize;     /* info group size */
    ZUCHAR aucSpare[2];       /* for 32 bit alignment */
    ST_ZOS_BKT_INFO *pstInfoGrp; /* info group */
    PFN_ZHEAPMALLOC pfnHeapAlloc; /* heap memory alloc function */
    PFN_ZHEAPFREE pfnHeapFree; /* heap memory free function */
} ST_ZOS_POOL_CFG;

```

The actual memory was allocated by calling malloc of OS by default. To use interface other than default, user can set it in the **PFN_ZHEAPMALLOC** of **ST_ZOS_POOL_CFG**. So did the free interface.

6.3 Other Interfaces

The interfaces to allocate and free memory includes:

```
/* zos memory pool alloc memory block with specific size */
ZVOID * Zos_PoolAlloc(ZPOOLID zPoolId, ZUINT iSize);

/* zos memory pool alloc memory block with specific size and zero block */
ZVOID * Zos_PoolAllocClrd(ZPOOLID zPoolId, ZUINT iSize);

/* zos memory pool free memory block */
ZVOID Zos_PoolFree(ZPOOLID zPoolId, ZVOID *pMem);

/* zos memory pool get size by the memory address */
ZINT Zos_PoolGetSize(ZPOOLID zPoolId, ZVOID *pMem, ZUINT *piSize);

/* zos memory pool check valid address of memory block */
ZBOOL Zos_PoolIsValid(ZPOOLID zPoolId, ZVOID *pData);
```

6.4 Exmample

First, you should define the bucket group information **ST_ZOS_BKT_INFO**. Then call **Zos_PoolCreate** to create bucket manager, call **Zos_PoolAlloc** to allocate memory from pool, call **Zos_PoolFree** to free memory. Detail example as following:

```

/* zos default memory bucket config info group */
static ST_ZOS_BKT_INFO m_astZosCfgMemDftBktInfoGrp[] =
{
    /* size,          maximum count,      increment count */
    {32,             0,                   10},
    {64,             0,                   10},
    {128,            0,                   10},
    {256,            0,                   10},
    {512,            0,                   10},
    {1024,           0,                   10},
    {2048,           0,                   10},
    {4096,           0,                   10},
    {8192,           0,                   10}
};

/* zos memory pool id for memory management */
static ZPOOLID m_zZosMemPoolId = ZNULL;

/* zos memory initialization */
ZINT Zos_MemInit()
{
    /* create memory pool */
    m_zZosMemPoolId = Zos_PoolCreate(&g_stZosSysCfg.stMem);
    if (m_zZosMemPoolId == ZNULL)
    {
        return ZFAILED;
    }

    return ZOK;
}

/* zos memory malloc */
ZVOID * Zos_Malloc(ZSIZE_T zSize)
{
    if (zSize >= ZMAXINT || zSize == 0)
        return ZNULL;

    return Zos_PoolAlloc(m_zZosMemPoolId, zSize);
}

/* zos memory free */
ZVOID Zos_Free(ZVOID *pMem)
{
    /* free data into the memory pool */

```

```

Zos_PoolFree(m_zZosMemPoolId, pMem);

return;
}

```

7. Data Buffer Management

7.1 Data Buffer Introduction

Buffer is used for data exchange between modules. The data could be protocol digest, task control message, etc. The main purpose to use buffer is reducing data copy times. And also use it as an effective and stable memory control.

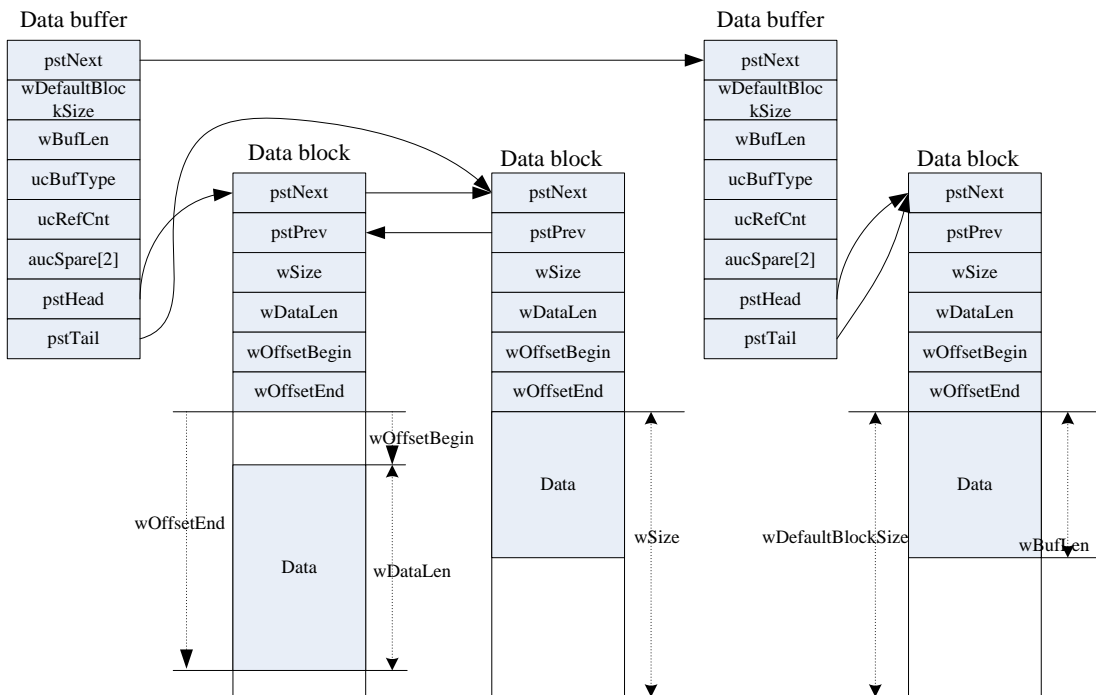


Figure 7-1 Data Buffer Structure

7.2 Data Buffer

There are 2 types of Data Buffer: byte aligned and structure aligned. Each buffer address get from structure aligned data buffer was aligned by 4 bytes.

The data buffer has a reference count. This count will increase by 1 when data buffer is cloned, and decrease by 1 when data buffer is deleted. When count is 0, the memory will actually be freed.

Here is the data sturcture:

```

/* zos data buffer data block */
typedef struct tagZOS_DBUF_DATA
{
    struct tagZOS_DBUF_DATA *pstNext; /* next data block */
    struct tagZOS_DBUF_DATA *pstPrev; /* previous data block */
    ZULONG dwSize;                    /* block size */
    ZULONG dwDataLen;                 /* data length in block */
    ZULONG dwOffsetBegin;             /* data begin offset in block */
    ZULONG dwOffsetEnd;               /* data end offset in block */
    /*lint -save -e* */
    ZCHAR acData[0];                  /* data memory */
    /*lint -restore */
} ST_ZOS_DBUF_DATA;

/* zos data buffer block */
typedef struct tagZOS_DBUF
{
    struct tagZOS_DBUF *pstNext;      /* next data buffer */
    ZPOOLID zPoolId;                  /* memory pool id for dbuf alloc */
    ZULONG dwBufLen;                  /* buffer used length */
    ZULONG dwDftBlkSize;              /* default data block size in buffer */
    ZUCHAR ucBufType;                 /* buffer mode ZDBUF_TYPE_BYTE... */
    ZUCHAR ucRefCnt;                  /* buffer reference count */
#ifdef ZOS_SUPT_DUMP
    ZDUMPID zDumpId;                  /* stack dump */
#endif
    ST_ZOS_DBUF_DATA *pstHead;        /* the first data block in buffer */
    ST_ZOS_DBUF_DATA *pstTail;       /* the last data block in buffer */
} ST_ZOS_DBUF;

```

7.3 Data Buffer Interfaces

```

/* create a data buffer */
ST_ZOS_DBUF * Zos_DbufCreate(ZPOOLID zPoolId, ZUCHAR ucType,
                             ZULONG dwDftBlkSize);

/* delete dbuf and all data block */
ZVOID Zos_DbufDelete(ST_ZOS_DBUF *pstBuf);

/* zos only free all data block */
ZVOID Zos_DbufFree(ST_ZOS_DBUF *pstBuf);

/* alloc data memory from dbuf */
ZVOID * Zos_DbufAlloc(ST_ZOS_DBUF *pstBuf, ZULONG dwSize);

/* alloc data memory from dbuf and clear data to 0 */
ZVOID * Zos_DbufAllocClr(ST_ZOS_DBUF *pstBuf, ZULONG dwSize);

/* length (used) of all data block in dbuf */
ZULONG Zos_DbufLen(ST_ZOS_DBUF *pstBuf);

/* size of all data block in dbuf */
ZULONG Zos_DbufSize(ST_ZOS_DBUF *pstBuf);

```

Please refer “ZOS Function Definition” for more information.

7.4 Data Buffer Example

```

/* sdp test decode message */
TSdp_DecodeMsg(ZCHAR *pcMemMsg)
{
    ST_ZOS_DBUF *pstMemBuf;
    ST_SDP_ANCMT *pstSdpMsg;

    /* create the dbuf */
    pstMemBuf = Zos_DbufCreate(ZNULL, ZDBUF_TYPE_STRUCT, 1024);
    if (pstMemBuf == ZNULL)
        return ZFAILED;
}

```

```
/* allocate memory for sdp message */
pstSdpMsg = Zos_DbufAlloc(pstMemBuf, sizeof(ST_SDP_ANCMT));
if (pstSdpMsg == ZNULL)
{
    /* delete memory buffer */
    Zos_DbufDelete(pstMemBuf);
    return ZFAILED;
}

/* initialize the error info */
Abnf_ErrInit(&stErrInfo);

/* decode message */
if (Sdp_DecodeMsg(&stDataStr, pstMemBuf, &stErrInfo, pstSdpMsg) != ZOK)
{
    /* delete memory buffer */
    Zos_DbufDelete(pstMemBuf);

    /* destroy error info */
    Abnf_ErrDestroy(&stErrInfo);
    return ZFAILED;
}

/* delete memory buffer */
Zos_DbufDelete(pstMemBuf);

/* destroy error info */
Abnf_ErrDestroy(&stErrInfo);

return ZOK;
}
```

8. ZOS Configuration

8.1 Configuration

Developer can change the configuration of ZOS, including memory configuration, modules, log setting, etc. All settings was store in the global structure of *g_stZosSysCfg* which structure is:

```

/* zos system config */
typedef struct tagZOS_SYS_CFG
{
    ST_ZOS_GEN_CFG stGen;           /* generic config */
    ST_ZOS_POOL_CFG stMem;         /* memory config */
    ST_ZOS_POOL_CFG stMsg;        /* message config */
    ST_ZOS_POOL_CFG stDbuf;       /* dbuf config */
    ST_ZOS_LOG_CFG stLog;         /* log config */
    ST_ZOS_MOD_CFG stMod;        /* module config */
    ST_ZOS_TASK_CFG stTask;       /* task config */
    ST_ZOS_TIMER_CFG stTimer;     /* timer config */
} ST_ZOS_SYS_CFG;

```

ZOS configuration includes:

- General Config
- Memory Pool Config
- Message Pool Config
- Data Buffer Pool Config
- Log Config
- Module Config
- Task Config
- Timer Config

8.1.1 General Config

The structure was defined as following:

```

/* zos general config parameter */
typedef struct tagZOS_GEN_CFG
{
    ZUCHAR ucIsSuptAssert;        /* is support ASSERT */
    ZUCHAR aucSpare[3];          /* for 32 bit alignment */
    PFN_ZPRINTDISP pfnPrintfDisp; /* printf display function */
    PFN_ZPRINTDISP pfnLogStrDisp; /* log string display function */
    PFN_ZHEAPMALLOC pfnHeapMalloc; /* heap memory alloc */
    PFN_ZHEAPFREE pfnHeapFree;    /* heap memory free */
} ST_ZOS_GEN_CFG;

```

Value	Default	Description
ucIsSuptAssert	ZFALSE	If ZTURE, ZOS_ASSERT will be replaced by Zos_Assert. Else, the macro will be null.
pfnPrintfDisp	ZNULL	Redirect Zos_Printf.

pfnLogStrDisp	ZNULL	Redirect log output.
pfnHeapMalloc	ZNULL	Memory allocate function pointer. If ZNULL, ZOS will call malloc() to get memory from OS.
pfnHeapFree	ZNULL	Memory free function pointer. If ZNULL, ZOS will call free() to free memory to OS.

Table 8-1 General Config

8.1.2 Memory Pool Config

The structure was defined as following:

```

/* zos pool config */
typedef struct tagZOS_POOL_CFG
{
    ZCHAR *pcName;           /* bucket manager name */
    ZUCHAR ucIsNeedMutex;   /* is need mutex */
    ZUCHAR ucInfoGrpSize;   /* info group size */
    ZUCHAR aucSpare[2];     /* for 32 bit alignment */
    ST_ZOS_BKT_INFO *pstInfoGrp; /* info group */
    PFN_ZHEAPALLOC pfnHeapAlloc; /* heap memory alloc function */
    PFN_ZHEAPFREE pfnHeapFree;  /* heap memory free function */
} ST_ZOS_POOL_CFG;

```

Value	Default	Description
pcName	"zos memory"	Memory bucket pool manager name.
ucIsNeedMutex	ZTRUE	If ZTRUE, ZOS will lock memory pool when access.
ucInfoGrpSize	ZOS_GET_TABLE_SIZE(m_astZosCfgMemDftBktInfoGrp)	The bucket information group size.
pstInfoGrp	m_astZosCfgMemDftBktInfoGrp	The bucket information group.
pfnHeapAlloc	ZNULL	The memory pool will get memory from OS by calling this function. If ZNULL, it will call stGen.pfnHeapMalloc .
pfnHeapFree	ZNULL	The memory pool will free memory to OS by calling this function. If ZNULL, it will call stGen.pfnHeapFree .

Table 8-2 Memory Pool Config

8.1.3 Message Pool Config

Value	Default	Description
pcName	"zos message"	Memory bucket pool manager name.
uclsNeedMutex	ZTRUE	If ZTRUE, ZOS will lock memory pool when access.
uclInfoGrpSize	ZOS_GET_TABLE_SIZE(m_astZosCfgMsgDftBktInfoGrp)	The bucket information group size.
pstInfoGrp	m_astZosCfgMsgDftBktInfoGrp	The bucket information group.
pfnHeapAlloc	Zos_Malloc	The memory pool will get memory from OS by calling this function. If ZNULL, it will call stGen.pfnHeapMalloc .
pfnHeapFree	Zos_Free	The memory pool will free memory to OS by calling this function. If ZNULL, it will call stGen.pfnHeapFree .

Table 8-3 Message Pool Config

8.1.4 Data Buffer Pool Config

Value	Default	Description
pcName	"zos dbuf"	Memory bucket pool manager name.
uclsNeedMutex	ZTRUE	If ZTRUE, ZOS will lock memory pool when access.
uclInfoGrpSize	ZOS_GET_TABLE_SIZE(m_astZosCfgDbufDftBktInfoGrp)	The bucket information group size.
pstInfoGrp	m_astZosCfgDbufDftBktInfoGrp	The bucket information group.
pfnHeapAlloc	Zos_Malloc	The memory pool will get memory from OS by calling this function. If ZNULL, it will call stGen.pfnHeapMalloc .
pfnHeapFree	Zos_Free	The memory pool will free memory to OS by calling this function. If ZNULL, it will call stGen.pfnHeapFree .

Table 8-4 Data Buffer Pool Config

8.1.5 Log Config

The structure was defined as following:

```

/* zos log config */
typedef struct tagZOS_LOG_CFG
{
    ZBOOL bTaskSupt;           /* log task support flag */
    ZULONG dwTaskStackSize;    /* log task stack size */
    ZINT iTaskPriority;        /* log task priority */
    ZULONG dwTaskDelayTime;    /* log task delay time(milliseconds) */
    ZULONG dwLineSize;        /* log line size */
    ZCHAR *pcZosLogFileName;   /* zos log file name */
    ZULONG dwZosLogLevel;     /* zos log level */
    ZULONG dwZosLogBufSize;    /* zos buffer size */
    ZULONG dwZosLogFileSize;   /* zos log file size */
    ZBOOL bZosLogPrint;       /* zos log print while start to run */
} ST_ZOS_LOG_CFG;

```

Value	Default	Description
bTaskSupt	ZFALSE	If ZTURE, it will start a task to process log.
dwTaskStackSize	40960	The log task stack size.
iTaskPriority	ZTASK_PRIORITY_MIN	The log task priority. ZTASK_PRIORITY_MAX ZTASK_PRIORITY_NORMAL ZTASK_PRIORITY_MIN ZTASK_PRIORITY_INCREMENT
dwTaskDelayTime	60000	The refresh time length in ms of log.
dwLineSize	1024	The line limit of log.
pcZosLogFileName	"zos.log"	The log file name.
dwZosLogLevel	ZLOG_LEVEL_FATAL ZLOG_LEVEL_ERROR ZLOG_LEVEL_INFO	The log level ZLOG_LEVEL_NULL, ZLOG_LEVEL_ALL, ZLOG_LEVEL_FATAL, ZLOG_LEVEL_ERROR, ZLOG_LEVEL_WARNING, ZLOG_LEVEL_INFO, ZLOG_LEVEL_DBG
dwZosLogBufSize	120 * 50	The buffer size of log.
dwZosLogFileSize	2000000	The file size of log.
bZosLogPrint	ZTRUE	If ZTRUE, it will output to stdout , while save to log file.

Table 8-5 Log Config

8.1.6 Module Config

The structure was defined as following:

```

/* zos module config */
typedef struct tagZOS_MOD_CFG
{
    ZUSHORT wInstNumPerMod;    /* instance number per module */
    ZUSHORT wModCount;        /* module count */
} ST_ZOS_MOD_CFG;
    
```

Value	Default	Description
wInstNumPerMod	5	The instance count per module.
wModCount	25	The module count.

Table 8-6 Module Config

8.1.7 Task Config

The structure was defined as following:

```

/* zos task config
 * time-slice in ticks or 0 to disable round-robin
 */
typedef struct tagZOS_TASK_CFG
{
    ZULONG dwDftTimeSlice;    /* default time slice */
    ZULONG dwDftStackSize;    /* default stack size */
    ZULONG dwDftQueueSize;    /* default queue size */
} ST_ZOS_TASK_CFG;
    
```

Value	Default	Description
dwDftTimeSlice	0	Only used in VxWorks.
dwDftStackSize	0x10000	The default stack size of ZOS Task.
dwDftQueueSize	100	The default message queue size of task.

Table 8-7 Task Config

8.1.8 Timer Config

The structure was defined as following:

```

/* zos timer config */
typedef struct tagZOS_TIMER_CFG
{
    ZULONG dwTimerNum;           /* timer number */
    ZULONG dwTimerInterval;     /* timer interval */
    ZULONG dwTaskDelayVal;      /* timer task delay value */
    ZULONG dwTaskStackSize;     /* timer task stack size */
    ZINT iTaskPriority;          /* timer task priority */
} ST_ZOS_TIMER_CFG;

```

Value	Default	Description
dwTimerNum	100	The timer count in timer task.
dwTimerInterval	500	The interval in ms of task timer time out. This value shouldn't less than 100.
dwTaskDelayVal	100	The delay in ms after process in timer task.
dwTaskStackSize	32768	The stack size of timer task.
iTaskPriority	ZTASK_PRIORITY_NORMAL	The priority level of timer task.

Table 8-8 Timer Config

9. Create Application

To create application base on ZOS, developer should check following:

- Define module ID and task ID.
- Include ZOS header files.
- Call ZOS initialization.
- Designe user task.
- Compile application with correct parameter.

9.1 Module ID and Task ID

Usually, one ZOS task is a thread of OS. And a ZOS module includes serval ZOS task which may have the similar function. Each task should be assigned with a Task ID, which made from Module ID. These ID are defined by developer. For example:

```

/* zos basic module configuration */
#define ZMODID_ROOT      0      /* root module id */
#define ZMODID_SYS      1      /* system module id */
#define ZMODID_UTAL     2      /* utal module id */
#define ZMODID_SIP      3      /* protocol sip module id */
#define ZMODID_RTP      4      /* protocol rtp module id */
#define ZMODID_H323     5      /* protocol h323 module id */
#define ZMODID_RTSP     6      /* protocol rtsp module id */
#define ZMODID_TEST     7      /* test module id */
#define ZMODID_SUA      8      /* sip user agent module id */
#define ZMODID_DNS      9      /* dns resolver module id */

/* zos basic module task identifier */
#define ZTASKID_ROOT      ZTASKID_MAKE(ZMODID_ROOT, 0)
#define ZTASKID_TIMER    ZTASKID_MAKE(ZMODID_SYS, 0)
#define ZTASKID_LOG      ZTASKID_MAKE(ZMODID_SYS, 1)
#define ZTASKID_UTAL     ZTASKID_MAKE(ZMODID_UTAL, 0)
#define ZTASKID_SIP      ZTASKID_MAKE(ZMODID_SIP, 0)
#define ZTASKID_SIP_TPT  ZTASKID_MAKE(ZMODID_SIP, 1)
#define ZTASKID_RTP      ZTASKID_MAKE(ZMODID_RTP, 0)
#define ZTASKID_RTP_TPT  ZTASKID_MAKE(ZMODID_RTP, 1)
#define ZTASKID_H323     ZTASKID_MAKE(ZMODID_H323, 0)
#define ZTASKID_H323_TPT ZTASKID_MAKE(ZMODID_H323, 1)
#define ZTASKID_RTSP     ZTASKID_MAKE(ZMODID_RTSP, 0)
#define ZTASKID_RTSP_UA  ZTASKID_MAKE(ZMODID_RTSP, 1)
#define ZTASKID_TEST     ZTASKID_MAKE(ZMODID_TEST, 0)
#define ZTASKID_TEST_0   ZTASKID_MAKE(ZMODID_TEST, 1)
#define ZTASKID_TEST_1   ZTASKID_MAKE(ZMODID_TEST, 2)
#define ZTASKID_TEST_2   ZTASKID_MAKE(ZMODID_TEST, 3)
#define ZTASKID_SUA      ZTASKID_MAKE(ZMODID_SUA, 0)
#define ZTASKID_DNS      ZTASKID_MAKE(ZMODID_DNS, 1)

```

If developer write a application named exm_zos, which contains 2 ZOS task. He may define as following:

```

/* exm module ID */
#define ZMODID_EXM      20

/* exm task 0 ID */
#define ZTASKID_EXM_0   ZTASKID_MAKE(ZMODID_EXM, 0)

/* exm task 1 ID */
#define ZTASKID_EXM_1   ZTASKID_MAKE(ZMODID_EXM, 1)

```

9.2 Header Files

The ZOS interfaces and definitions are include in the file of zos.h. Developer should include it in the program.

```
#include "zos.h"                /* zos system environment */
```

Please refer “ZOS Function Definition” for detail.

9.3 ZOS Initialization

Usually, the main function may looks like:

```
ZINT main()
{
    /* generic config */
    Zos_SysCfgInit();

    /* system init */
    if (Zos_SysInit() != ZOK)
    {
        return -1;
    }

    /* configure module */
    Exam_CfgInit();

    /* example start */
    Exam_Start();

    /* system destroy */
    Zos_SysDestroy();

    return 0;
}
```

9.4 Design User Task

9.4.1 Using Module Functions

In this case, we first define *Exm_TaskInit()*, *Exm_TaskDestroy()* 和 *Exm_TaskMsgProc()*. Then call **ZOS_MOD_REG** 和 **ZOS_MOD_START** to register and start task.

```
/* example task 0 start */
ZINT Exm0_Start()
{
    /* reg EXM_0 */
    if (ZOS_MOD_REG(ZTASKID_EXM_0, "EXM_0", 50, 10, Exm_TaskInit,
        Exm_TaskDestroy) != ZOK)
        return ZFAILED;

    Zos_Printf("reg EXM_0 ok.\r\n");

    /* start EXM_0 */
    if (ZOS_MOD_START(ZTASKID_EXM_0, ZTASK_PRIORITY_NORMAL, 0,
        Exm_TaskMsgProc) != ZOK)
        return ZFAILED;

    return ZOK;
}
```

Exm_SendMsg was used to send ZOS message.

```

/* send message to EXM_0 */
ZINT Exm_SendMsg()
{
    ST_EXM_MSG *pstExmMsg;
    ST_ZOS_MSG *pstSysMsg;
    ST_ZOS_MSP stMsp;

    /* init upper layer msp */
    stMsp.zSendCpuId = ZCPUID_LOCAL;
    stMsp.zSendTaskId = 0; /* in fact from main task */
    stMsp.zRecvCpuId = ZCPUID_LOCAL;
    stMsp.zRecvTaskId = ZTASKID_EXM_0;
    stMsp.zPoolId = NULL;

    /* alloc system message */
    pstSysMsg = Zos_MsgAlloc(&stMsp, sizeof(ST_EXM_MSG));
    if (pstSysMsg == ZNULL)
    {
        Zos_Printf("alloc ZOS message failed!\r\n");
        return ZFAILED;
    }

    /* set information */
    pstExmMsg = (ST_EXM_MSG *)ZOS_MSG_GET_DATA(pstSysMsg);
    Zos_StrNCpy(pstExmMsg->acData, "Hello ZOS!", 15);

    /* send ZOS message */
    if (Zos_MsgSend(pstSysMsg) != ZOK)
    {
        /* free system message */
        Zos_MsgFree(pstSysMsg);
        Zos_Printf("send ZOS message failed\r\n");
        return ZFAILED;
    }

    return ZOK;
}

```

9.4.2 Using Zos_TaskSpawn

```

/* example task 1 start */
ZINT Exm1_Start()
{
    /* spawn EXM_1 */

```

```

if (Zos_TaskSpawn(ZTASKID_EXM_1, "EXM_1", ZTASK_PRIORITY_NORMAL,
                 0, 0, 100, Exm_Perm, 0) != ZOK)
    return ZFAILED;

return ZOK;
}

```

9.5 Compile

ZOS has been porting to Windows, VxWorks, Linux and Solaris. To compile under different environment, developer should add some parameter.

For example, in Linux and using gcc and make tools, we should check:

Flags	Description
Include Dir	Directories including ZOS header files must be included in compile phase. Get those headers from released product set and add words in compile environment like this: <code>/I "../include/zos"</code> in Win32 environment or <code>-I "../include/zos"</code>
Platform Macro	A macro named ZPLATFORM must be defined to indicate the platform on which SIP applications will be compiled and run. Available values for ZPLATFORM are: ZPLATFORM_VXWORKS ZPLATFORM_WIN32 ZPLATFORM_LINUX ZPLATFORM_SOLARIS The words may be like this: /D ZPLATFORM=ZPLATFORM_WIN32 or -DZPLATFORM=ZPLATFORM_LINUX
Link Dir	Library directories including all necessary Juphoon product libraries should be indicated in compile environment. The words may like be this: /libpath:"../lib" or -L "../lib"
Link Library	ZOS lib must be linked at the link phase. The words may me like this: zos.lib or -lzos Other system libraries may be also linked. For example, the winsock library <code>ws2_32.lib</code> in Win32 environment and pthread library in POSIX environment.

Table 9-1 Compile & Link Flags

10. Appendix

10.1 ZOS Compile Options

Option	Description
ZPLATFORM	ZPLATFORM_WIN32 ZPLATFORM_WINCE ZPLATFORM_VXWORKS ZPLATFORM_THREADX ZPLATFORM_LINUX ZPLATFORM_FREEBSD ZPLATFORM_SOLARIS Under Win32: -DZPLATFORM=ZPLATFORM_WIN32
ZOS_SUPT_64BIT	
ZCPU_SPARC	
ZCPU_68K	
ZOS_SUPT_MEM_DBG	
ZOS_SUPT_DBG	
ZOS_SUPT_DUMP	
ZOS_SUPT_DUMP_TASK	
ZOS_SUPT_POOL_DUMP	
ZOS_NOTUSE_STDLIB	
ZOS_NOTUSE_STDIO	

Table 10-1 ZOS Compile Option

10.2 ZOS Macro Definition

```

/* zos specific type maximum value */
#define ZMAXCPUID ZMAXULONG          /* maximum cpu id value */
#define ZMAXMODID ZMAXUSHORT         /* maximum module id value */
#define ZMAXINSTID ZMAXUSHORT        /* maximum instance id value */
#define ZMAXTASKID ZMAXULONG         /* maximum task id value */
#define ZMAXTIMERID ZMAXULONG        /* maximum timer id value */
#define ZMAXEVTID ZMAXULONG          /* maximum event id value */
#define ZMAXSAPID ZMAXULONG          /* maximum sap id value */
#define ZMAXENDPID ZMAXULONG         /* maximum endpoint id value */
#define ZMAXREASONID ZMAXULONG       /* maximum reason id value */

/* operating system type */
#define ZPLATFORM_WIN32 1 /* windows */
#define ZPLATFORM_WINCE 2 /* windows */
#define ZPLATFORM_VXWORKS 3 /* vxworks */
#define ZPLATFORM_THREADX 4 /* threadx */
#define ZPLATFORM_LINUX 5 /* linux */
#define ZPLATFORM_FREEBSD 6 /* freebsd */
#define ZPLATFORM_SOLARIS 7 /* solaris -- unix */

/* zos protocol type */
#define ZPROTOCOL_UNKNOWN 0 /* unknown protocol */
#define ZPROTOCOL_SDP 1 /* sdp protocol */
#define ZPROTOCOL_MGCP 2 /* mgcp text protocol */
#define ZPROTOCOL_MGCO_TXT 3 /* megaco text protocol */
#define ZPROTOCOL_MGCO_BIN 4 /* megaco binary protocol */
#define ZPROTOCOL_SIP 5 /* sip protocol */
#define ZPROTOCOL_RTSP 6 /* rtsp protocol */
#define ZPROTOCOL_H323 7 /* h.323 protocol */

/* zos data buffer type */
#define ZDBUF_TYPE_NULL 0 /* buffer type undefined */
#define ZDBUF_TYPE_BYTE 1 /* all data blocks are byte alignment */
#define ZDBUF_TYPE_STRUCT 2 /* all data blocks are 4 bytes alignment */

/* zos dlist maximum infinite size */
#define ZDLIST_INFINITE_SIZE ZMAXULONG

/* fsm next state needn't change, fsm keep state */
#define ZFSM_STA_NOCHANGE 0

/* fsm next state is invalid while input invalid event, fsm keep state */
#define ZFSM_STA_INVALID -1

```

```
/* fsm next state is error while input error event, fsm keep state */
#define ZFSM_STA_ERROR      -2

/* fsm decode function return failed */
#define ZFSM_LOCATE_FAIL    -1

#define ZFSM_OK             0 /* fsm run ok */
#define ZFSM_FAIL          -1 /* fsm run fail */
#define ZFSM_ERR_UNKNOWN_STA -2 /* fsm unknown state error */
#define ZFSM_ERR_UNKNOWN_EVNT -3 /* fsm unknown event error */
#define ZFSM_ERR_INVALID_EVNT -4 /* fsm invalid event error */
#define ZFSM_ERR_ERROR_EVNT -5 /* fsm error event error */
#define ZFSM_ERR_UNEXPECTED_EVNT -6 /* fsm unexpected event error */
#define ZFSM_ERR_DATA_ERROR -7 /* fsm object or event data error */

/* zos fsm dump stack size */
#define ZFSM_DUMP_STACK_SIZE 10

/* zos log option */
#define ZLOG_OPT_NULL      0x00000000 /* no option */
#define ZLOG_OPT_MUTEX    0x00000001 /* asynchronism option */
#define ZLOG_OPT_PRINT    0x00000002 /* print option */

/* zos log level */
#define ZLOG_LEVEL_NULL    0x00000000 /* null to be logged */
#define ZLOG_LEVEL_FATAL  0x00010000 /* fatal message to be logged */
#define ZLOG_LEVEL_ERROR  0x00020000 /* error message to be logged */
#define ZLOG_LEVEL_WARNING 0x00040000 /* warning message to be logged */
#define ZLOG_LEVEL_INFO   0x00080000 /* info message to be logged */
#define ZLOG_LEVEL_DBG    0x00100000 /* debug message to be logged */
#define ZLOG_LEVEL_ALL    0xFFFF0000 /* all message to be logged */
```

```

/* semaphore wait timeout macros */
#define ZSEMA_WAIT_FOREVER ZMAXULONG /* semaphore wait forever */
#define ZSEMA_NO_WAIT 0 /* semaphore no wait */

/* zos slist maximum infinite size */
#define ZSLIST_INFINITE_SIZE ZMAXULONG

/* zos string default max length of string */
#define ZSTRUL_MAXLEN 32
#define ZSTRUS_MAXLEN 16
#define ZSTRUC_MAXLEN 8

/* task options */
#define ZTASK_PREEMPT 0x00 /* enable task rescheduling */
#define ZTASK_NO_PREEMPT 0x01 /* disable task rescheduling */
#define ZTASK_TIMESLICE 0x02 /* enable round-robin selection */
#define ZTASK_NO_TIMESLICE 0x04 /* disable task rescheduling */

/* local cpu id */
#define ZCPUID_LOCAL 0

/* timer mode */
#define ZTIMER_MODE_CYCLE 0x01 /* cycle timer */
#define ZTIMER_MODE_NOCYCLE 0x02 /* not cycle timer */
#define ZTIMER_MODE_100MS 0x04 /* 100 ms timer */

```

10.3 ZOS Macro Function

```

/* get low/high byte of a word */
ZOS_GET_LOW_BYTE
ZOS_GET_HIGH_BYTE
/* get low/high word of a unsigned long */
ZOS_GET_LOW_WORD
ZOS_GET_HIGH_WORD
/* set low/high byte of a word */
ZOS_PUT_LOW_BYTE
ZOS_PUT_HIGH_BYTE
/* set low/high word of a word */
ZOS_PUT_LOW_WORD
ZOS_PUT_HIGH_WORD
ZOS_MAKE_WORD
ZOS_MAKE_LONG
/* check specific character property */

```

```
ZOS_ISALPHA
ZOS_ISUPPER
ZOS_ISLOWER
ZOS_ISDIGIT
ZOS_ISXDIGI
ZOS_ISSPACE
ZOS_ISPUNCT
ZOS_ISALNUM
ZOS_ISPRINT
ZOS_ISGRAPH
ZOS_ISCNTRL
ZOS_ISASCII
/* space or tab character test */
ZOS_ISBLANK
/* is whitespace */
ZOS_ISWS
ZOS_ISLWS
/* to upper/lower */
ZOS_TOUPPER
ZOS_TOLOWER
/* to ascii */
ZOS_TOASCII
/* is odd/even */
ZOS_ISODD
ZOS_ISEVEN
/* to byte type */
ZOS_TOZCHAR
ZOS_TOBYTE
/* to absolute value */
ZOS_TOABS
/* get member offset from structure */
ZOS_OFFSETOF

/* ZOS_ASSERT macro */
ZOS_ASSERT

/* zos calculate the hash key from case sensitive */
ZOS_HASH_GET_KEY_FROM_STR

/* zos calculate the hash key from non-case sensitive */
ZOS_HASH_GET_KEY_FROM_STR_NOCASE

/* zos register one module */
ZOS_MOD_REG
```

```
/* zos deregister one module */
ZOS_MOD_DEREG
/* zos start one module */
ZOS_MOD_START

/* zos message macors */
ZOS_MSG_GET_EVNT_TYPE
ZOS_MSG_GET_LEN
ZOS_MSG_GET_DATA
ZOS_MSG_GET_DATA_LEN
ZOS_MSG_GET_USED_LEN

/* zos size alignment */
ZOS_ALIGN

/* zos time to high resolution time */
ZOS_TIME_TO_HRTIME

/* zos string buffer print macros */
ZOS_PRINT_OUT_START
ZOS_PRINT_OUT_END
ZOS_PRINT_OUT_CHECK
ZOS_PRINT_PUT_STR
ZOS_PRINT_PUT_NSTR
ZOS_PRINT_PUT_SSTR
ZOS_PRINT_PUT_FMT1
ZOS_PRINT_PUT_FMT2
ZOS_PRINT_PUT_FMT3
ZOS_PRINT_PUT_FMT4

/* get maximum, minimum va
ZOS_MAX
ZOS_MIN

/* set operation */
ZOS_MASK
ZOS_SET
ZOS_CLR
ZOS_ISSET
ZOS_ZERO

/* counting and rounding *
ZOS_ROUNDDOWN
ZOS_ROUNDUP
```

```
ZOS_ROUNDUP2
```

```
ZOS_ISPOWEROF2
```

```
/* get table size */
```

```
ZOS_GET_TABLE_SIZE
```

10.4 ZOS Function Prototype

```
/* zos brick match function type for find_if */
typedef ZINT (*PFN_ZBKMATCH)(ZBKDATA zBkData, ZVOID *pCond);

/* zos brick action function type for for_each */
typedef ZVOID (*PFN_ZBKACTION)(ZBKDATA zBkData, ZVOID *pParm);

/* zos fsm state action about event */
typedef ZINT (*PFN_ZFSMACTION)(ZVOID *, ZVOID *);

/* zos fsm state table locate index from event */
typedef ZINT (*PFN_ZFSMLOCATE)(ZVOID *pFsmObj, ZINT iMajorEvt, ZINT iMinorEvt);

/* zos hash key and compare function pointer macros */
typedef ZINT (*PFN_ZHASHKEY)(ZULONG dwType, ZULONG dwParm1, \
                             ZULONG dwParm2, ZULONG *pdwHashKey);

typedef ZINT (*PFN_ZHASHCMP)(ZULONG dwEntry, ZULONG dwType, \
                             ZULONG dwParm1, ZULONG dwParm2);

/* zos instance initialize function */
typedef ZINT (*PFN_ZINSTINIT)(ZINSTID zInstId);

/* zos instance destroy function */
typedef ZVOID (*PFN_ZINSTDESTROY)(ZINSTID zInstId);

/* zos instance message process function */
typedef ZINT (*PFN_ZINSTMSGPROC)(ZVOID *pMsg);

/* task entry function */
typedef ZULONG (*PFN_ZTASKENTRY)(ZVOID *);

/* zos ring timer active function for callback */
typedef ZVOID (*PFN_ZRTIMERACTION)(ZTIMERID zTimerId,
                                   ZULONG dwTimerType, ZULONG dwParm);

/* zos print display functions. */
typedef ZINT (*PFN_ZPRINTDISP)(const ZCHAR *pcFormat, ...);

/* zos heap memory allocate function from specific os */
typedef ZVOID * (*PFN_ZHEAPMALLOC)(ZSIZE_T zSize);

/* zos heap memory free function from specific os */
typedef ZVOID (*PFN_ZHEAPFREE)(ZVOID *pMem);
```

```
/* zos timer active function for callback */
typedef ZVOID (*PFN_ZTIMERACTIVE)(ZTIMERID zTimerId, ZULONG dwTimerType,
                                   ZULONG dwParm);
```

10.5 ZOS Error No

```
/* no error */
ZERR_NO

/* error code of task module */
ZERR_TASK_INIT
ZERR_TASK_NULLP
ZERR_TASK_INV
ZERR_TASK_TIMERCREATE
ZERR_TASK_TIMERDELETE
ZERR_TASK_TIMERSTART
ZERR_TASK_GETTASK
ZERR_TASK_GETSELFID
ZERR_TASK_STATE
ZERR_TASK_QUEUECREATE
ZERR_TASK_QUEUEGET
ZERR_TASK_QUEUEPOP
ZERR_TASK_QUEUEPUSH
ZERR_TASK_QTIMERINIT
ZERR_TASK_QTIMERCREATE
ZERR_TASK_QTIMERSTART
ZERR_TASK_SPAWN
ZERR_TASK_DELETE
ZERR_TASK_SUSPEND
ZERR_TASK_RESUME
ZERR_TASK_RESTART
ZERR_TASK_HASHINSERT
ZERR_TASK_MSGALLOC
ZERR_TASK_MSGSEND
ZERR_TASK_MSGLEN
ZERR_TASK_PRIORITYSET
ZERR_TASK_NULLENTRY
ZERR_TASK_CREATEMUTEX
ZERR_TASK_CREATTHREAD
```

```
/* error code of msg module */
ZERR_MSG_NULLP
ZERR_MSG_POOLCREATE
ZERR_MSG_POOLALLOC
ZERR_MSG_INVLEN
ZERR_MSG_INVSTENTRY

/* error code of time module */
ZERR_TIME_INIT
ZERR_TIME_NULLP
ZERR_TIME_GETEPOCH
ZERR_TIME_SETEPOCH
ZERR_TIME_MKTIME
ZERR_TIME_LOCALTIME
ZERR_TIME_GETDAYOFWK
ZERR_TIME_CLOCKTICK
ZERR_TIME_OPENCPUINFO
ZERR_TIME_CLOCKGETTIME
ZERR_TIME_CLOCKSETTIME
ZERR_TIME_CLOCKGETRES
ZERR_TIME_CONVERTSYSTIME
ZERR_TIME_SETSYSTIME
ZERR_TIME_SETTIMEOFDAY

/* error code of timer module */
ZERR_TIMER_INIT
ZERR_TIMER_NULLP
ZERR_TIMER_TASKPROCESS
ZERR_TIMER_CREATE
ZERR_TIMER_DELETE
ZERR_TIMER_START
ZERR_TIMER_STOP
ZERR_TIMER_TIMELEN

/* error code of string module */
ZERR_STRING_NULLP
ZERR_STRING_INV

/* error code of socket module */
ZERR_SOCKET_NULLP
ZERR_SOCKET_INV

/* error code of inet module */
ZERR_INET_NULLP
```

```
ZERR_INET_ADDR
ZERR_INET_WSSTARTUP
ZERR_INET_WSCLEANUP

/* error code of list module */
ZERR_LIST_NULLP
ZERR_LIST_FULL
ZERR_LIST_EMPTY
ZERR_LIST_NOTIN

/* error code of queue module */
ZERR_QUEUE_NULLP
ZERR_QUEUE_INV
ZERR_QUEUE_MUTEXCREATE
ZERR_QUEUE_SEMACREATE
ZERR_QUEUE_MALLOC
ZERR_QUEUE_FULL

/* error code of qtimer module */
ZERR_QTIMER_NULLP
ZERR_QTIMER_INVNUM
ZERR_QTIMER_INVTMRID
ZERR_QTIMER_MALLOC
ZERR_QTIMER_MUTEXCREATE
ZERR_QTIMER_EMPTY
ZERR_QTIMER_STATE
ZERR_QTIMER_INSERT
ZERR_QTIMER_MSGALLOC
ZERR_QTIMER_MSGSEND

/* error code of rtimer module */
ZERR_RTIMER_NULLP
ZERR_RTIMER_INVNUM
ZERR_RTIMER_INVTMRID
ZERR_RTIMER_MALLOC
ZERR_RTIMER_MUTEXCREATE
ZERR_RTIMER_EMPTY
ZERR_RTIMER_STATE
ZERR_RTIMER_INSERT
ZERR_RTIMER_MSGALLOC
ZERR_RTIMER_MSGSEND
ZERR_RTIMER_BASSETIMER

/* error code of pool module */
```

```
ZERR_POOL_NULLP
ZERR_POOL_INV
ZERR_POOL_SIZE2OBIG
ZERR_POOL_HEAPALLOC
ZERR_POOL_INVBKTID
ZERR_POOL_INVMAGICID
ZERR_POOL_INVREDZONE
ZERR_POOL_INVSOMEID
ZERR_POOL_INVBKTSIZE
ZERR_POOL_FORBIDINC
ZERR_POOL_BKTCREATE
ZERR_POOL_BKTGRPCREATE
ZERR_POOL_MUTEXCREATE
ZERR_POOL_EMPTY
ZERR_POOL_INVCOUNT

/* error code of mutex module */
ZERR_MUTEX_NULLP

/* error code of module module */
ZERR_MODULE_NULLP
ZERR_MODULE_INVTASKID
ZERR_MODULE_NOTINIT
ZERR_MODULE_NOTREG
ZERR_MODULE_REGED
ZERR_MODULE_TASKSPAWN
ZERR_MODULE_MALLOC
ZERR_MODULE_NOQUEUE

/* error code of memory module */
ZERR_MEM_NULLP
ZERR_MEM_POOLCREATE
ZERR_MEM_POOLALLOC
ZERR_MEM_POOLGETSIZE
ZERR_MEM_INVSIZE
ZERR_MEM_MEMCHK

/* error code of hash module */
ZERR_HASH_NULLP
ZERR_HASH_MALLOC
ZERR_HASH_ITEMEXIST
ZERR_HASH_ITEMNOTEXIST
ZERR_HASH_FULL
```

```
/* error code of fsm module */
```

```
ZERR_FSM_NULLP  
ZERR_FSM_UNKSTATE  
ZERR_FSM_UNKEVNT  
ZERR_FSM_INVEVNT  
ZERR_FSM_ERREVNT
```

```
/* error code of dbuf module */
```

```
ZERR_DBUF_NULLP  
ZERR_DBUF_INV  
ZERR_DBUF_ALLOCDATA  
ZERR_DBUF_POOLCREATE  
ZERR_DBUF_INVTYPE  
ZERR_DBUF_INVBLKSIZE  
ZERR_DBUF_ALLOCRES  
ZERR_DBUF_BUFREUSE  
ZERR_DBUF_INVLEN  
ZERR_DBUF_DBUFCREATE  
ZERR_DBUF_INVOFFSET  
ZERR_DBUF_INVSRCBUF  
ZERR_DBUF_NOROOM  
ZERR_DBUF_ADDDATA  
ZERR_DBUF_NODATA  
ZERR_DBUF_NOTINBUF
```

```
/* error code of abnf module */
```

```
ZERR_ABNF_NULLP  
ZERR_ABNF_INV  
ZERR_ABNF_MALLOC  
ZERR_ABNF_INVBITARRAYSIZE  
ZERR_ABNF_INVMAGICID  
ZERR_ABNF_HASHCREATE  
ZERR_ABNF_HASHINSERT  
ZERR_ABNF_TKNFULL  
ZERR_ABNF_PREADDDATA  
ZERR_ABNF_PSTADDDATA  
ZERR_ABNF_PREDELDATA  
ZERR_ABNF_PSTDELDATA  
ZERR_ABNF_DBUFCOPY  
ZERR_ABNF_DBUFCAT
```

```
/* error code of idle module */
```

```
ZERR_IDLE_NULLP  
ZERR_IDLE_INV
```

ZERR_IDLE_TASKSPAWN

10.6 ZOS Log Level Function

```
/* zos log macros */  
#define ZOS_LOG_FATAL(_args) ZXX_LOG_FATAL(ZOS_LOGID, _args)  
#define ZOS_LOG_ERROR(_args) ZXX_LOG_ERROR(ZOS_LOGID, _args)  
#define ZOS_LOG_WARNING(_args) ZXX_LOG_WARNING(ZOS_LOGID, _args)  
#define ZOS_LOG_INFO(_args) ZXX_LOG_INFO(ZOS_LOGID, _args)  
#define ZOS_LOG_DBG(_args) ZXX_LOG_DBG(ZOS_LOGID, _args)
```

10.7 ZOS List Macro Function

```

#define ZOS_SLIST_SIZE(_slist) \
    ((_slist)->dwCount)

#define ZOS_SLIST_ISFULL(_slist) \
    ((_slist)->dwCount >= (_slist)->dwMaxNum)

#define ZOS_SLIST_ISEMPY(_slist) \
    ((_slist)->pstHead == ZNULL)

#define ZOS_SLIST_HEAD_NODE(_slist) (_slist)->pstHead
#define ZOS_SLIST_TAIL_NODE(_slist) (_slist)->pstTail

#define ZOS_SLIST_NODE_NEXT(_node) (_node)->pstNext
#define ZOS_SLIST_NODE_DATA(_node) (_node)->pData

#define FOR_ALL_NODE_IN_SLIST(_slist, _node) \
    for (_node = ZOS_SLIST_HEAD_NODE(_slist); \
        _node != ZNULL; \
        _node = ZOS_SLIST_NODE_NEXT(_node))

/* zos slist create */
#define ZOS_SLIST_CREATE(_slist, _size) \
    Zos_SlistCreate(_slist, _size)

/* zos slist delete */
#define ZOS_SLIST_DELETE(_slist) \
    Zos_SlistDelete(_slist)

/* zos slist node init */
#define ZOS_SLIST_NODE_INIT(_node, _data) do { \
    (_node)->pstNext = ZNULL; \
    (_node)->pData = (ZCHAR *)(_data); \
} while (0)

/* zos insert node before head in slist, seemed as a function */
#define Zos_SlistAdd2Head(_slist, _node) \
    Zos_SlistInsert(_slist, ZNULL, _node)

/* zos insert node after tail in slist, seemed as a function */
#define Zos_SlistAdd2Tail(_slist, _node) \
    Zos_SlistInsert(_slist, (_slist)->pstTail, _node)

```

```

/* zos insert node before head in slist */
#define ZOS_SLIST_ADD2HEAD(_slist, _node) \
    Zos_SlistInsert(_slist, ZNULL, _node)

/* zos insert node after tail in slist */
#define ZOS_SLIST_ADD2TAIL(_slist, _node) \
    Zos_SlistInsert(_slist, (_slist)->pstTail, _node)

/* zos insert node after previou node in slist */
#define ZOS_SLIST_INSERT(_slist, _prevnode, _node) \
    Zos_SlistInsert(_slist, _prevnode, _node)

/* zos dequeue node from the first node in slist */
#define ZOS_SLIST_DEQUEUE(_slist, _node) \
    _node = Zos_SlistDequeue(_slist)

/* zos remove one node */
#define ZOS_SLIST_REMOVE(_slist, _node) \
    Zos_SlistRemove(_slist, _node)

/* zos find node by index */
#define ZOS_SLIST_FIND_BY_INDEX(_slist, _index, _node) \
    _node = Zos_SlistFindByIndex(_slist, _index)

/* zos typedef slist with specific name */
#define ZOS_TYPEDEF_SLIST(_name) \
    /* single list node */ \
    typedef struct tag##_name##_LST_NODE \
    { \
        struct tag##_name##_LST_NODE *pstNext; /* next slist node */ \
        ST_##_name *pData; /* slist node data */ \
    } ST_##_name##_LST_NODE; \
    /* single list */ \
    typedef struct tag##_name##_LST \
    { \
        ZULONG dwMaxNum; /* maximum number of slist nodes */ \
        ZULONG dwCount; /* actual count of slist nodes */ \
        ST_##_name##_LST_NODE *pstHead; /* slist node head */ \
        ST_##_name##_LST_NODE *pstTail; /* slist node tail */ \
    } ST_##_name##_LST

```

```
/* typedef slist macros */
#define ZOS_TYPEDEF_SLIST_SIZE(_list) \
    ZOS_SLIST_SIZE((ST_ZOS_SLIST *)(_list))

#define ZOS_TYPEDEF_SLIST_ISFULL(_list) \
    ZOS_SLIST_ISFULL((ST_ZOS_SLIST *)(_list))

#define ZOS_TYPEDEF_SLIST_ISEMPY(_list) \
    ZOS_SLIST_ISEMPY((ST_ZOS_SLIST *)(_list))

#define ZOS_TYPEDEF_SLIST_HEAD_NODE(_list) \
    ZOS_SLIST_HEAD_NODE((ST_ZOS_SLIST *)(_list))

#define ZOS_TYPEDEF_SLIST_TAIL_NODE(_list) \
    ZOS_SLIST_TAIL_NODE((ST_ZOS_SLIST *)(_list))

#define ZOS_TYPEDEF_SLIST_NODE_NEXT(_node) \
    ZOS_SLIST_NODE_NEXT((ST_ZOS_SLIST_NODE *)(_node))

#define ZOS_TYPEDEF_SLIST_NODE_DATA(_node) \
    ZOS_SLIST_NODE_DATA((ST_ZOS_SLIST_NODE *)(_node))

#define FOR_ALL_NODE_IN_TYPEDEF_SLIST(_list, _node) \
    FOR_ALL_NODE_IN_SLIST((ST_ZOS_SLIST *)(_list), _node)

/* zos typedef slist create */
#define ZOS_TYPEDEF_SLIST_CREATE(_list, _size) \
    ZOS_SLIST_CREATE((ST_ZOS_SLIST *)(_list), _size)

/* zos zos typedef slist delete */
#define ZOS_TYPEDEF_SLIST_DELETE(_list) \
    ZOS_SLIST_DELETE((ST_ZOS_SLIST *)(_list))

/* zos typedef slist node init */
#define ZOS_TYPEDEF_SLIST_NODE_INIT(_node, _data) \
    ZOS_SLIST_NODE_INIT((ST_ZOS_SLIST_NODE *)(_node), _data)

/* zos insert node before head in typedef slist */
#define ZOS_TYPEDEF_SLIST_ADD2HEAD(_list, _node) \
    ZOS_SLIST_ADD2HEAD((ST_ZOS_SLIST *)(_list), _node)

/* zos insert node after tail in typedef slist */
#define ZOS_TYPEDEF_SLIST_ADD2TAIL(_list, _node) \
    ZOS_SLIST_ADD2TAIL((ST_ZOS_SLIST *)(_list), _node)
```

```

/* zos insert node after previou node in typedef slist */
#define ZOS_TYPEDEF_SLIST_INSERT(_list, _prevnode, _node) \
    ZOS_SLIST_INSERT((ST_ZOS_SLIST *)(_list), _prevnode, _node)

/* zos dequeue node from the first node in typedef slist */
#define ZOS_TYPEDEF_SLIST_DEQUEUE(_list, _node) \
    ZOS_SLIST_DEQUEUE((ST_ZOS_SLIST *)(_list), _node)

/* zos remove one node */
#define ZOS_TYPEDEF_SLIST_REMOVE(_list, _node) \
    Zos_SlistRemove((ST_ZOS_SLIST *)(_list), (_node))

/* zos find node by index */
#define ZOS_TYPEDEF_SLIST_FIND_BY_INDEX(_list, _index, _node) \
    ZOS_SLIST_FIND_BY_INDEX((ST_ZOS_SLIST *)_list, _index, _node)

```

10.8 ZOS Double Linked List Macro Function

```

#define ZOS_DLIST_SIZE(_dlist) \
    ((_dlist)->dwCount)

#define ZOS_DLIST_ISFULL(_dlist) \
    ((_dlist)->dwCount >= (_dlist)->dwMaxNum)

#define ZOS_DLIST_ISEMPY(_dlist) \
    ((_dlist)->pstHead == ZNULL)

#define ZOS_DLIST_HEAD_NODE(_dlist) (_dlist)->pstHead
#define ZOS_DLIST_TAIL_NODE(_dlist) (_dlist)->pstTail

#define ZOS_DLIST_NODE_NEXT(_node) (_node)->pstNext
#define ZOS_DLIST_NODE_DATA(_node) (_node)->pData

#define FOR_ALL_NODE_IN_DLIST(_dlist, _node) \
    for (_node = ZOS_DLIST_HEAD_NODE(_dlist); \
        _node != ZNULL; \
        _node = ZOS_DLIST_NODE_NEXT(_node))

/* zos dlist create */
#define ZOS_DLIST_CREATE(_dlist, _size) \
    Zos_DlistCreate(_dlist, _size)

/* zos dlist delete */

```

```
#define ZOS_DLIST_DELETE(_dlist) \  
    Zos_DlistDelete(_dlist)  
  
/* zos dlist node init */  
#define ZOS_DLIST_NODE_INIT(_node, _data) do { \  
    (_node)->pstPrev = ZNULL; \  
    (_node)->pstNext = ZNULL; \  
    (_node)->pData = (ZCHAR *)_data; \  
} while (0)  
  
/* zos insert node before head in dlist, seemed as a function */  
#define Zos_DlistAdd2Head(_dlist, _node) \  
    Zos_DlistInsert(_dlist, ZNULL, _node)  
  
/* zos insert node after tail in dlist, seemed as a function */  
#define Zos_DlistAdd2Tail(_dlist, _node) \  
    Zos_DlistInsert(_dlist, (_dlist)->pstTail, _node)  
  
/* zos insert node before head in dlist */  
#define ZOS_DLIST_ADD2HEAD(_dlist, _node) \  
    Zos_DlistInsert(_dlist, ZNULL, _node)  
  
/* zos insert node after tail in dlist */  
#define ZOS_DLIST_ADD2TAIL(_dlist, _node) \  
    Zos_DlistInsert(_dlist, (_dlist)->pstTail, _node)  
  
/* zos insert node after the after previou node in dlist */  
#define ZOS_DLIST_INSERT(_dlist, _prevnode, _node) \  
    Zos_DlistInsert(_dlist, _prevnode, _node)
```

```

/*lint -save -e* */

/* zos dequeue node from the first node in dlist */
#define ZOS_DLIST_DEQUEUE(_dlist, _node) \
    _node = Zos_DlistDequeue(_dlist)

/* zos remove one node */
#define ZOS_DLIST_REMOVE(_dlist, _node) \
    Zos_DlistRemove(_dlist, _node)

/* zos find node by index */
#define ZOS_DLIST_FIND_BY_INDEX(_dlist, _index, _node) \
    _node = Zos_DlistFindByIndex(_dlist, _index)

/*lint -restore */

/* zos typedef dlist with specific name */
#define ZOS_TYPEDEF_DLIST(_name) \
    /* double list node */ \
    typedef struct tag##_name##_LST_NODE \
    { \
        struct tag##_name##_LST_NODE *pstNext; /* next dlist node */ \
        struct tag##_name##_LST_NODE *pstPrev; /* previous dlist node */ \
        ST_##_name *pData; /* dlist node data */ \
    } ST_##_name##_LST_NODE; \
    /* double list */ \
    typedef struct tag##_name##_LST \
    { \
        ZULONG dwMaxNum; /* maximum number of dlist nodes */ \
        ZULONG dwCount; /* actual count of dlist nodes */ \
        ST_##_name##_LST_NODE *pstHead; /* dlist node head */ \
        ST_##_name##_LST_NODE *pstTail; /* dlist node tail */ \
    } ST_##_name##_LST

/* typedef dlist macros */
#define ZOS_TYPEDEF_DLIST_SIZE(_list) \
    ZOS_DLIST_SIZE((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_ISFULL(_list) \
    ZOS_DLIST_ISFULL((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_ISEMPY(_list) \
    ZOS_DLIST_ISEMPY((ST_ZOS_DLIST *)(_list))

```

```
#define ZOS_TYPEDEF_DLIST_HEAD_NODE(_list) \  
    ZOS_DLIST_HEAD_NODE((ST_ZOS_DLIST *)(_list))  
  
#define ZOS_TYPEDEF_DLIST_TAIL_NODE(_list) \  
    ZOS_DLIST_TAIL_NODE((ST_ZOS_DLIST *)(_list))  
  
#define ZOS_TYPEDEF_DLIST_NODE_NEXT(_node) \  
    ZOS_DLIST_NODE_NEXT((ST_ZOS_DLIST_NODE *)(_node))  
  
#define ZOS_TYPEDEF_DLIST_NODE_DATA(_node) \  
    ((ST_ZOS_DLIST_NODE *)(_node))->pData  
  
#define FOR_ALL_NODE_IN_TYPEDEF_DLIST(_list, _node) \  
    FOR_ALL_NODE_IN_DLIST((ST_ZOS_DLIST *)(_list), _node)  
  
/* zos typedef dlist create */  
#define ZOS_TYPEDEF_DLIST_CREATE(_list, _size) \  
    ZOS_DLIST_CREATE((ST_ZOS_DLIST *)(_list), _size)  
  
/* zos typedef dlist delete */  
#define ZOS_TYPEDEF_DLIST_DELETE(_list) \  
    ZOS_DLIST_DELETE((ST_ZOS_DLIST *)(_list))  
  
/* zos typedef node init */  
#define ZOS_TYPEDEF_DLIST_NODE_INIT(_node, _data) \  
    ZOS_DLIST_NODE_INIT((ST_ZOS_DLIST_NODE *)_node, _data)  
  
/* zos insert node before head in typedef dlist */  
#define ZOS_TYPEDEF_DLIST_ADD2HEAD(_list, _node) \  
    ZOS_DLIST_ADD2HEAD((ST_ZOS_DLIST *)(_list), _node)  
  
/* zos insert node after tail in typedef dlist */  
#define ZOS_TYPEDEF_DLIST_ADD2TAIL(_list, _node) \  
    ZOS_DLIST_ADD2TAIL((ST_ZOS_DLIST *)(_list), _node)  
  
/* zos insert node after the after previou node in typedef dlist */  
#define ZOS_TYPEDEF_DLIST_INSERT(_dlist, _prevnode, _node) \  
    ZOS_DLIST_INSERT((ST_ZOS_DLIST *)(_dlist), _prevnode, _node)  
  
/* zos dequeue node from the first node in typedef dlist */  
#define ZOS_TYPEDEF_DLIST_DEQUEUE(_list, _node) \  
    ZOS_DLIST_DEQUEUE((ST_ZOS_DLIST *)(_list), _node)
```

```
/* zos remove one node */
#define ZOS_TYPEDEF_DLIST_REMOVE(_list, _node) \
    ZOS_DLIST_REMOVE((ST_ZOS_DLIST *)(_list), _node)

/* zos typedef dlist find node data on index location */
#define ZOS_TYPEDEF_DLIST_FIND_BY_INDEX(_list, _index, _node) \
    ZOS_DLIST_FIND_BY_INDEX((ST_ZOS_DLIST *)_list, _index, _node)
```