Common Language Extension Interface for c++

Contents

1 Macro definitions and constants

Definition is in header file: “vsopencommtype.h”.

```
#define SRPAPI __cdecl
#define SRPCALLBACK __cdecl          Callback function, equivalent to SRPAPI
```

1.1 Constant and macro definitions

```
#define VSSYNCSTATUS_NOTSYNC      //  Not synchronized
#define VSSYNCSTATUS_SYNC         //  Synchronization status
#define VSSYNCSTATUS_INSYNC       //  In Synchronization process
```

1.1.1 Remote Procedure Call Results

```
#define VSRCALL_OK          0   //--- Success
#define VSRCALL_COMMERROR     -1   //--- Communication error
#define VSRCALL_OBJNOTEXIST   -2   //--- Object does not exist
#define VSRCALL_FUNCNOTEXIST   -3   //--- Function does not exist
#define VSRCALL_PARAERROR     -4   //--- Parameter error
#define VSRCALL_SYSERROR      -5   //--- System error
#define VSRCALL_INVALIDUSR    -6   //--- Users are illegal
#define VSRCALL_OVERTIME      -7   //--- Timeout
#define VSRCALL_UNKNOWN       -8   //--- Other errors
```

1.1.2 Remote Procedure Call source tag

```
#define VSRCALLSRC_C          0   //--- C-generated calls
#define VSRCALLSRC_SCRIPT     1   //--- Calls generated by kinds of scripting languages
#define VSRCALLSRC_WEBSERVICE  2   //--- WebServie call
```

1.1.3 Lua variable type

```
#define VSLUATYPE_NIL        0
#define VSLUATYPE_NUMBER      1
#define VSLUATYPE_BOOL        2
#define VSLUATYPE_STRING      3
#define VSLUATYPE_FUNCTION    4
#define VSLUATYPE_TABLE       5
#define VSLUATYPE_OBJECT      6
#define VSLUATYPE_PARAPKG     7
#define VSLUATYPE_QUERYRECORD  8
#define VSLUATYPE_TIME        9
#define VSLUATYPE_FONT        10
#define VSLUATYPE_RECT        11
#define VSLUATYPE_BINBUF      12
#define VSLUATYPE_SXML        13
#define VSLUATYPE_FUNCTIONPARA 14
#define VSLUATYPE_COMMINTERFACE 15
#define VSLUATYPE_INT         16  // is integer, always be VSLUATYPE_NUMBER
#define VSLUATYPE_USERDATA     17
#define VSLUATYPE_LIGHTUSERDATA 18
#define VSLUATYPE_UNKNOWN     255
```

1.1.4 System Event Processing flag

#define VSSYSEVENT_PROCESS_TICKET 0x0001 //--- Handle 10ms ticket event
#define VSSYSEVENT_PROCESS_FRAMETICKET 0x0002 //--- Handle frame pulse events from the server
#define VSSYSEVENT_PROCESS_IDLE 0x0004 //--- Handle idle event
#define VSSYSEVENT_PROCESS_APPACTIVE 0x0008
#define VSSYSEVENT_PROCESS_APPDEACTIVE 0x0010
#define VSSYSEVENT_PROCESS_SERVICEACTIVE 0x0020
#define VSSYSEVENT_PROCESS_SERVICEDEACTIVE 0x0040
#define VSSYSEVENT_PROCESS_SELFEVENT 0x0080 //--Handle object's own event
#define VSSYSEVENT_PROCESS_ACTIVESET 0x0100 //---Active set change event

1.1.5 Sub-object event handling flags

#define VSSYSEVENT_PROCESS_CREATE **[only generated for direct child objects]**
#define VSSYSEVENT_PROCESS_DESTROY
#define VSSYSEVENT_PROCESS_ACTIVATE
#define VSSYSEVENT_PROCESS_DEACTIVATE
#define VSSYSEVENT_PROCESS_SYNCGROUPCHANGE

#define VSSYSEVENT_PROCESS_ANYCREATE **[generated for any child objects, including child objects of the child]**
#define VSSYSEVENT_PROCESS_ANYDESTROY
#define VSSYSEVENT_PROCESS_ANYACTIVATE
#define VSSYSEVENT_PROCESS_ANYDEACTIVATE
#define VSSYSEVENT_PROCESS_ANYSYNCGROUPCHANGE

1.1.6 Text display format category

#define TEXTDISPLAY_CLASSID_NORMALTEXT normal text
#define TEXTDISPLAY_CLASSID_EXPLANE note
#define TEXTDISPLAY_CLASSID_OBJECTNAME object name
#define TEXTDISPLAY_CLASSID_ATTRIBUTETYPE attribute type
#define TEXTDISPLAY_CLASSID_NUMBER number
#define TEXTDISPLAY_CLASSID_ERRORORWARN error or warning

#define TEXTDISPLAY_FORMAT "\\Fmt" //-- Separated by a space,for example, \\Fmt1 ljsdflksdf

1.1.7 Service group ID

#define VS_DEFAULT_SERVICEGROUPID 0 //--Default service group ID
#define VS_INVALID_SERVICEGROUPID 0xFFFFFFFF

1.1.8 Object name length

#define DEFAULT_NAMELENGTH 40 //--- Object name, title, function name, title, properties, macro definition, structure name, module name, service name, the output event name, event handler function name. Maximum length is 39 characters
#define SCRIPTINTERFACE_LENGTH 16 //-- Scripting interface name length, which is actually 15 bytes.

1.1.9 Define the type

```
//=========ProgramTypeID
typedef VS_UINT16 VS_PROGRAMTYPE;
//--basic type
#define VS_SERVER      ((VS_UINT16)0x0000)
#define VS_CLIENT      ((VS_UINT16)0x0001)
#define VS_DEBUG       ((VS_UINT16)0x0004)
#define VS_TOOLS       ((VS_UINT16)0x0008)
//--extend type
#define VS_SERVER_SERVER     (((VS_UINT16)0x0000) | VS_SERVER)   //--Server side
#define VS_SERVER_USER       (((VS_UINT16)0x0100) | VS_SERVER)   //-- Standalone server side
#define VS_CLIENT_USER       (((VS_UINT16)0x0000) | VS_CLIENT)   //--Standalone client side
#define VS_CLIENT_COOPERATOR (((VS_UINT16)0x0100) | VS_CLIENT)   //-- Cooperator Client side
#define VS_CLIENT_CALLER     (((VS_UINT16)0x0200) | VS_CLIENT)   //-- The caller Client side
#define VS_BASIC_PROGRAMTYPE(X)  (X & 0x00FF)

//=========module type
#define VSMODULE_ALL              ((VS_UINT16)0)   //--valid at client,debug,server
#define VSMODULE_SERVER_SERVER    ((VS_UINT16)1)   //--valid at server
#define VSMODULE_SERVER_USER      ((VS_UINT16)2)   //--valid at server user
#define VSMODULE_CLIENT_USER      ((VS_UINT16)4)   //--valid at client user
#define VSMODULE_CLIENT_COOPERATOR ((VS_UINT16)8)   //--valid at client cooperator
#define VSMODULE_CLIENT_CALLER    ((VS_UINT16)16)   //--valid at server client end
#define VSMODULE_DEBUG            ((VS_UINT16)32)   //--valid at debug
#define VS_ISMODULEEXIST(Type,X)
#define VS_ISMODULENOTEXIST(Type,X)
```

1.1.10 Service running information

```
#define VSOS_WIN32     ((VS_UINT32)0x01)
#define VSOS_LINUX     ((VS_UINT32)0x02)
#define VSOS_ANDROID   ((VS_UINT32)0x04)
#define VSOS_ANDROIDV7A  ((VS_UINT32)0x08)
#define VSOS_IOS  ((VS_UINT32)0x10)
#define VSOS_WP  ((VS_UINT32)0x20)

typedef struct{
   VS_UINT32 OsType;
   VS_UINT16 ProgramRunType;
   VS_UINT16 Reserved;
}VS_SERVICEOSRUNINFO;
```

1.1.11 Global object permissions for client

```
#define VSCLIENTOP_CREATE       ((VS_ULONG)0x00000001)
#define VSCLIENTOP_DELETE       ((VS_ULONG)0x00000002)    //can delete object which does not
belong to this client
#define VSCLIENTOP_CHANGE       ((VS_ULONG)0x00000004)    //can change object which does not
belong to this client
```

1.1.12 System alarm level

```
#define VSFAULT_INDICATION      0x00  // indication
#define VSFAULT_WARNING         0x01  // warn
#define VSFAULT_NORMALERROR     0x02  // General error, you can proceed
#define VSFAULT_CRITICALERROR   0x03  // Serious error, need to exit the current service
#define VSFAULT_SYSTEMERROR     0x04  // Serious error, need to exit the program
#define VSFAULT_DISP            0x06  // Displays a single line of information
```

#define VSFAULT_OPENSHOW 0x07 // External print information, which call interface pSRP-> Print to print the information

1.1.13 Object alloc type

#define VSALLOCTYPE_STATIC 1
#define VSALLOCTYPE_GLOBAL 2
#define VSALLOCTYPE_CLIENT 3
#define VSALLOCTYPE_LOCAL 4

1.1.14 Object save type

//=========Define object save flag
#define VSSAVE_SAVE 0 //--- Based on object's type: global (static, dynamic, customer) or local
#define VSSAVE_LOCAL 1 //---save as local object
#define VSSAVE_GLOBAL 2 //---save as global object
#define VSSAVE_NONE 3 //---do not save

1.1.15 Object active command

#define VSACTIVE_ALONE 0 //---Activate or deactivate object by command
#define VSACTIVE_FOLLOW 1 //--- Activated with parent object. If the parent object is service item, it will be activated automatically.
//-- The following two commands are dynamically set, can not be stored. When saved, it will be converted to VSACTIVE_ALONE.
#define VSACTIVE_ACTIVE 2 //--- If the service is running, the object will be automatically activated
#define VSACTIVE_DEACTIVE 3 //--- If the service is running, the object will be automatically deactivated

1.1.16 Static data save flag

#define VSSTATIC_SAVE 0 //---save
#define VSSTATIC_CLIENTSAVE 1 //---save at client side
#define VSSTATIC_NONE 2 //---not save

1.1.17 Object attribute index

#define **INVALID_OBJECTATTRIBUTEINDEX** Invalid attribute index

typedef VS_UINT8 **OBJECTATTRIBUTEINDEX**; Attribute index scope is [0-127].Each object is up to 127 properties

//--- The following structure is used in change notification event of object properties
typedef struct{
VS_ULONG SysAttributeMap; //--- Property for internal use
VS_ULONG AppAttributeMap[4]; //--- Property for app use
}**VS_ATTRIBUTEINDEXMAP**;
#define VS_SETSYSATTRMAP(MapPtr,Attr)
#define VS_SETAPPATTRMAP(MapPtr,Attr)
#define VS_GETSYSATTRMAP(MapPtr,Attr)
#define VS_GETAPPATTRMAP(MapPtr,Attr)

#define **UUID_ISEQUAL**(X,Y)
#define **UUID_ISUNEQUAL**(X,Y)
#define **INIT_ UUID**(X)
#define **UUID_ISINVALID** (X)

#define **UUID_ISVALID** (X)

#define **DEFAULT_ACTIVESET_NUMBER** Maximum number of active set
typedef struct{
VS_INT32 ActiveSetNumber; //-- If equal to 0, then sync only for the group 0
VS_ULONG ActiveSet[DEFAULT_ACTIVESET_NUMBER]; // Group synchronization index == 0 is the default. The number of elements is ActiveSetNumber. If [0] value is equal to 0xFFFFFFFF, then all groups should be synchronized **[Note: synchronization settings should be done on the server side only]**
}**VS_ACTIVESETITEM**;

//=========Object sync group
typedef **VS_ULONG VS_SYNCGROUP**;

#define **VSMODULEDEPEND_OBJECT**
#define **VSMODULEDEPEND_FUNCRETURN**
#define **VSMODULEDEPEND_FUNCPARA**

typedef struct{
VS_INT32 AttributeType;
VS_INT32 AttributeOffset;
}**VS_DEPENDATTRIBUTE**;

1.1.18 Common struct define

struct **SrtuctOfClassSkeleton_PointerSequence**{ sequence pointer
VS_INT32 Number;
VS_INT8 *Sequence[1];
};

typedef RECT **VS_RECT**;

typedef struct{
VS_INT8 *Ptr;
VS_INT8 *SequencePtr;
}**VS_SEQUENCEPTR**;

//====Font parameter
#define VSFONT_BOLD 0x01
#define VSFONT_ITALIC 0x02
#define VSFONT_UNDERLINE 0x04
#define VSFONT_STRIKEOUT 0x08
#define VSFONT_NAMELENGTH 32

typedef struct{
VS_COLOR Color;
VS_INT32 Height,Size;
VS_UINT8 CharSet;
VS_UINT8 Style;
VS_UINT8 Position;
VS_INT8 Reserve;
VS_INT32 Pitch;
VS_CHAR Name[LOCALFONT_NAMELENGTH];
}**VS_FONT**;

typedef SYSTEMTIME **VS_TIME**;

typedef struct{ //---variable length string
VS_CHAR *Buf; //-- The actual buffer length is Length + 1 (include the last 0), if application directly set the pointer, it must use interface function pSRP -> Malloc to alloc buffer.
}**VS_VSTRING**;

typedef VS_UUID **VS_STATICID**;
typedef COLORREF **VS_COLOR**;
typedef class ClassOfSRPParaPackageInterface * **VS_PARAPKGPTR**;
typedef class ClassOfSRPBinBufInterface * **VS_BINBUFPTR**;
typedef void * **VS_OBJPTR;**

1.1.19 object's attribute related definitions

```
//---attribute type
#define VSTYPE_BOOL          1
#define VSTYPE_INT8          2
#define VSTYPE_UINT8         3
#define VSTYPE_INT16         4
#define VSTYPE_UINT16        6
#define VSTYPE_INT32         6
#define VSTYPE_UINT32        7
#define VSTYPE_FLOAT         8
#define VSTYPE_LONG          9
#define VSTYPE_ULONG         10
#define VSTYPE_LONGHEX       11
#define VSTYPE_ULONGHEX      12
#define VSTYPE_VSTRING       51
#define VSTYPE_PTR           14
#define VSTYPE_MEMORY        15
#define VSTYPE_STRUCT        16
#define VSTYPE_COLOR         19
#define VSTYPE_RECT          20
#define VSTYPE_FONT          21
#define VSTYPE_TIME          49
#define VSTYPE_CHAR          13
#define VSTYPE_UUID          41
#define VSTYPE_STATICID      29
#define VSTYPE_CHARPTR       30
#define VSTYPE_PARAPKGPTR    40
//--can be used for function parameter
#define VSTYPE_INT8PTR       55
#define VSTYPE_UINT8PTR      54
#define VSTYPE_INT16PTR      31
#define VSTYPE_UINT16PTR     52
#define VSTYPE_INT32PTR      32
#define VSTYPE_UINT32PTR     53
#define VSTYPE_FLOATPTR      33
#define VSTYPE_ULONGPTR      48
#define VSTYPE_LONGPTR       34
#define VSTYPE_STRUCTPTR     35
#define VSTYPE_COLORPTR      37
#define VSTYPE_RECTPTR       38
#define VSTYPE_FONTPTR       39
#define VSTYPE_TIMEPTR       50
#define VSTYPE_UUIDPTR       47
```

```
#define VSTYPE_VOID           254   //---not exist in CLE
#define VSTYPE_OBJPTR          57
#define VSTYPE_TABLE           56    //--- only for remote call delay return

#define VSTYPE_BINBUFPTR       59
#define VSTYPE_DOUBLE          58
#define VSTYPE_INT64           60
#define VSTYPE_UWORD           61

#define VSTYPE_DOUBLEPTR       63
#define VSTYPE_INT64PTR        62

#define VSTYPE_IGNORE          255   //--- not exist in CLE

//---variable edit
#define VSEDIT_EDIT           0x00
#define VSEDIT_COMBOBOX        0x01
#define VSEDIT_CHECKBOX        0x02
#define VSEDIT_HIDE           0x03  //---not display
#define VSEDIT_MASK            0x04

typedef struct{
   VS_INT32 NumberOfContent;
   VS_CHAR Content[256][50];
   VS_LONG ComboBoxToValueIndex[256];   // ComboBox subscript is the index value in the content
}VS_COMBOBOXITEM;

typedef struct{
   VS_CHAR  Name[DEFAULT_NAMELENGTH];
   VS_CHAR  Caption[DEFAULT_NAMELENGTH];
   VS_CHAR  DefaultString[DEFAULT_NAMELENGTH];  //  Only support string format
   VS_UINT8 Type;    //--attribute type
      VS_UINT8  EditType;  // attribute edit type, normal edit, combobox, button
   VS_UINT8  EditReadOnly; // ==0 can be edit    ==1 readonly
   VS_BOOL SyncType;         //--true syncpointer, false asynchronous pointer(In this case the value is not CLE
object)
   VS_UINT8 CreateNeedFlag;   //--1 needed when created   0 not needed
   VS_UINT8 ChangeNotifyFlag; //--0 without notification, 1 notify when changed, 2 notify before changed
   VS_INT32 Length;           //--attribute length
   VS_INT32 Offset;           //--The offset from the object address, which does not include the variables of CLE
   VS_UINT8 ComboBoxID[20];  //---comboox id
   VS_UUID  StructID;        //--- Structure ID, valid when the type is struct, or pointer
   VS_ULONG StaticID;    //-- Static data ID, valid when the attribute data type is static
   VS_UINT8 AtomicAttributeIndex; //--- Global property index, including properties defined in CLE.
   VS_UINT8 AttributeIndex;       //---The attribute's local index, not including properties defined in CLE.
   VS_UINT8 Reserved[2];
   void     *AtomicAttributeObject;  //---atomic attribute object
}VS_ATTRIBUTEINFO;

typedef struct{
   VS_CHAR  Name[DEFAULT_NAMELENGTH];
   VS_UINT8 CanNotBeOVLFlag;              //  can be overloaded
   VS_BOOL LuaFunctionFlag;               //  is lua function
   VS_BOOL CallBackFlag;                  //  is callback function
  VS_UUID FunctionID;           //  function id
   VS_UUID OriginFunctionID;      //  if is overloading function, then it is origin function id
```

```
   void *CallFunction;                  //  function address
}VS_FUNCTIONINFO;

typedef struct{
   VS_CHAR  Name[DEFAULT_NAMELENGTH];
   VS_BOOL  DynamicFlag;               // --true is dynamic event, --false  is static event.
   VS_UUID EventID;          //  event id
}VS_OUTEVENTINFO;
```

1.1.20 Query record

```
typedef struct{
   VS_INT8 Reserved[32];
}VS_QUERYRECORD;
```

1.1.21 client Qos

```
//--------------------------------------------------------------------------
//---------------------- Client Qos parameters. On the server side can be different for each client
//--------------------------------------------------------------------------
#define VSCLIENTQOS_SERVICECLASS_NORMAL     0   //---normal service class
#define VSCLIENTQOS_SERVICECLASS_PRIORITY   1   //---priority service class
#define VSCLIENTQOS_LOADRATE_MIN  1024     //---min value
#define VSCLIENTQOS_LOADRATE_MAX  32768    //---max value

typedef struct{
   VS_ULONG ServiceClass;  //---service class
   VS_ULONG UpLoadRatePerFrame;    //---client upload rate,  Bytes/Frame,[1024 –32768    ]
   VS_ULONG DownLoadRatePerFrame;  //--- client download rate,  Bytes/Frame,[1024 -32768    ]
   VS_ULONG Reserved[5];
}VS_CLIENTQOS;
```

1.1.22 client callback object operation

```
//--------------------------------------------------------------------------
//---------------------- Clients create, modify or delete an object callback function, valid at server side
//--------------------------------------------------------------------------
#define VSCLIENTOP_DELOBJECT     0
#define VSCLIENTOP_CHANGEOBJECT  1
#define VSCLIENTOP_CREATEOBJECT  2
#define VSCLIENTOP_CREATEITEMOBJECT  3
typedef VS_BOOL (SRPAPI *VS_ClientOperationCallBackProc)( VS_ULONG Para,VS_ULONG
uMsg,VS_ULONG ClientID,VS_ULONG ClientPrivateTag,void *Object,OBJECTATTRIBUTEINDEX
ParentIndex,VS_UUID *ClassID);
//---ClientPrivateTag: Used to determine the legality of client.
//---for delete, uMsg = VSCLIENTOP_DELOBJECT; Object is valid
//---for change, uMsg = VSCLIENTOP_CHANGEOBJECT; Object is valid
//---for create, uMsg = VSCLIENTOP_CREATEOBJECT;Object is parent object, ParentIndex is queue of
parent object, ClassID is id of class
//---for create, uMsg = VSCLIENTOP_CREATEITEMOBJECT; Object is service item, ParentIndex is queue
of parent object, ClassID is id of class
//--- If return false, not able to perform operations the above
```

1.1.23 client information

```
//-------------------------------------------------------------------------
//--------------------client information struct
//-------------------------------------------------------------------------
typedef struct{
   VS_ULONG ClientMachineID;
   SOCKADDR_IN ClientSockAddr;  //---client address
   VS_ULONG ClientModuleID;       //---client program ID
   VS_ULONG ClientServiceGroupID;  //---client service group id
   VS_INT32   DirectConnectFlag;    //--- ==1  direct connect,  ==0  connected through SRPDispatch(not
support for current version)
    SOCKADDR_IN SRPDispatchSockAddr;  //---SRP dispatcher address
   VS_ULONG SRPDispatchModuleID;       //---SRP dispatcher program id
   VS_ULONG SRPDispatchServiceGroupID;  //---SRP dispatcher service group id
   VS_SERVICEOSRUNINFO OsRunInfo;
   VS_INT8  Reserved[32];
}VS_CLIENTINFO;
```

1.1.24 Client side callback interface to connect

```
//-------------------------------------------------------------------------
//--------------------- Client side callback interface to connect
//-------------------------------------------------------------------------
#define VS_LINKINTERFACESTATUS_OK        0x00000000
#define VS_LINKINTERFACESTATUS_DOWNLOAD   0x00000001
#define VS_LINKINTERFACESTATUS_ERROR      0x00000002

#define VSCLIENTCONNECT_ONCONNECT      0   //--- After the event, The connection will be successfully established. Then
the following is service initialization message.
#define VSCLIENTCONNECT_ONFAILURE      1   //--- If ConnectionID equals to 0, then the request is released, no longer
continue to generate the callback, otherwise, would be to try
#define VSCLIENTCONNECT_ONINITFAILUER  2   //--- Client side service failed to initialize. When the connection does not
succeed. Application calls function DisConnect will also trigger the callback
#define VSCLIENTCONNECT_ONINITSUCCESS  3   //--- Client service is initialized successfully
#define VSCLIENTCONNECT_ONSERVICESYNC  4   //---Client side service completes synchronization
#define VSCLIENTCONNECT_ONDISCONNECT   5   //--- Client side connection is terminated

typedef void (SRPAPI * VS_ClientConnectCallBackProc)( VS_ULONG ServiceGroupID, VS_ULONG uMsg,VS_ULONG
ConnectionID, VS_ULONG LinkInterfaceStatus, VS_CHAR *ServerName, VS_UINT16 ServerPortNumber, VS_ULONG Para);
// Para is parameters to establish a connection
```

1.1.25 server-side redirect

```
//-------------------------------------------------------------------------
#define VSREDIRECT_ONCONNECT      0       //--- Redirect successfully. The connection to the client after
the callback message will be automatically closed.
#define VSREDIRECT_ONFAILURE      1       //--- Redirect failed

typedef void (SRPAPI * VS_RedirectCallBackProc)( VS_ULONG uMsg,VS_ULONG ClientID, VS_CHAR
*DesServerName,VS_UINT16 DesServerPortNumber, VS_ULONG Para);
```

1.1.26 Service Statistics

```
typedef struct{
   VS_ULONG AttributeNumber;  //---attribute number
   VS_ULONG FunctionNumber;   //---function number
   VS_ULONG InputEventNumber; //---input event number
```

```
    VS_ULONG OutputEventNumber;//---output event number
    VS_ULONG NameScriptNumber; //---name script number
    VS_ULONG ObjectNumber;     //---object number
    VS_ULONG Reserved[16];
}VS_SERVICEINFO;
```

1.1.27 file upload and download information

```
typedef struct{
    //------------ Upload information
    VS_INT32 UpDataFile;  // --0 static data  -- 1 file
    union{
        struct{
            VS_UUID ObjectID;                   //--- If invalid, then the object or service does not exist
            VS_ULONG UniqueDataUnitID;
            VS_STATICID Version;  //--- Only valid for static data
        }StaticData;
        struct{
            VS_CHAR FileName[256];  //---file name.     Equal to "" indicates invalid
        }FileData;
    }Up;
    VS_INT32  UpLoadFileSize;
    VS_INT32  UpLoadTransferSize;
    //-------------Download information
    VS_INT32 DownDataFile;  // --0 static data  -- 1 file
    union{
        struct{
            VS_UUID ObjectID;                   //--- If invalid, then the object or service does not exist
            VS_ULONG UniqueDataUnitID;
            VS_STATICID Version;  //--- Only valid for static data
        }StaticData;
        struct{
            VS_CHAR FileName[256];  //--- file name.     Equal to "" indicates invalid
        }FileData;
    }Down;
    VS_INT32  DownLoadFileSize;
    VS_INT32  DownLoadTransferSize;
}VS_UPDOWNFILEINFO;

//------------------------------------------------------------------------------
//--- Callback notification message
#define VSFILE_ONDOWNSTART      0    //---start download
#define VSFILE_ONDOWNPROGRESS   1    //---download process
#define VSFILE_ONDOWNFINISH     2    //---download finish
#define VSFILE_ONDOWNERROR      3    //---download error
#define VSFILE_ONUPSTART        4    //---start upload
#define VSFILE_ONUPPROGRESS     5    //---upload process
#define VSFILE_ONUPFINISH       6    //---upload finish
#define VSFILE_ONUPERROR        7    //---upload error

#define VSFILE_STATUSIDLE       0    //--- Not download or upload files
#define VSFILE_STATUSUP         1    //---Upload file
#define VSFILE_STATUSDOWN      -1    //---Download file

typedef struct{
    VS_INT32 DataFile;  // --0 static data  -- 1 file
    union{
```

```
        struct{
            VS_UUID ObjectID;
            VS_ULONG UniqueDataUnitID;
            VS_STATICID Version;  //--- Only valid for static data
             VS_UINT8 *DataBuf;      //---set only for
                  VSFILE_ONDOWNPROGRESS,VSFILE_ONDOWNFINISH,VSFILE_ONUPPROGRESS
        }StaticData;
        struct{
            VS_UUID ObjectID;                   //--- If invalid, then the object or service does not exist
            VS_CHAR FileName[256];  //---file name
             VS_UINT8 *FileBuf;      //---set only for
                   VSFILE_ONDOWNPROGRESS,VSFILE_ONDOWNFINISH,VSFILE_ONUPPROGRESS
        }FileData;
    }u;
    VS_INT32  DataSize;       //---file or data size

    VS_INT32  ReceiveOrSendOffset;    //--- Data received or uploaded
}VS_UPDOWNFILEMSG;

#define VSFILE_RET_OK      0    //---continue process
#define VSFILE_RET_ABORT   1    //--- Cancel the file download process. Return value is meaningful only in
the VSFILE_ONDOWNPROGRESS process, and for a single file

typedef VS_ULONG (SRPAPI *VS_FileUpDownLoadCallBackProc)( void *Object, VS_ULONG Para,
VS_ULONG uMsg, VS_UPDOWNFILEMSG *Msg);

//---used at server, support VSFILE_ONDOWNSTART and VSFILE_ONUPSTART
//   if return VS_FALSE, the process will be cancel by server
typedef VS_BOOL (SRPAPI *VS_FileUpDownLoadRequestCallBackProc)( VS_ULONG ClientID,
VS_ULONG ClientPrivateTag, VS_ULONG Para, VS_ULONG uMsg, VS_UPDOWNFILEMSG *Msg);
```

1.1.28 object module information struct

```
//------------------------------------------------------------------------------
//--------------------- object module information struct
//------------------------------------------------------------------------------
typedef struct{
    //--------object version
    VS_UINT8 ObjectVersion;
    VS_UINT8 ObjectSubVersion;
    VS_UINT16 ObjcetBugFixVersion;
    //-------platform version supported
    VS_UINT16 VSVersion;
    VS_UINT16 VSSubVersion;
    SYSTEMTIME CreateTime;
    SYSTEMTIME UpdateTime;
    VS_CHAR AuthorityInfo[128];
    VS_CHAR LicenseInfo[256];
    VS_CHAR ExtendInfo[256];     //--- Reserved
}VS_OBJECTMODULEINFO;
```

1.1.29 RawSocket parameters

```
#define VS_LINKINTERFACESTATUS_OK         0x00000000
#define VS_LINKINTERFACESTATUS_DOWNLOAD   0x00000001
```

#define VS_LINKINTERFACESTATUS_ERROR 0x00000002

#define VS_SOCKET_ONACCEPT 0x00000001
#define VS_SOCKET_ONCONNECT 0x00000002 //-- Successfully establish a connection.Mes points to struct, and MesLength equals to 0.
typedef struct{
SOCKADDR_IN SockAddr; // peer's IP address and port number.
}VS_SOCKETONCONNECT;

#define VS_SOCKET_ONDISCONNECT 0x00000003 //-- Close the connection, then no callback will be created, Mes = NULL, MesLength = 0
#define VS_SOCKET_ONFAILCONNECT 0x00000004 //-- Connection fails, then no callback will be created, Mes = NULL, MesLength = 0
#define VS_SOCKET_ONRECEIVE 0x00000005 //--Receive a packet of data, Mes point to ClassOfSRPParaPackageInterface, MesLength = 0
#define VS_SOCKET_ONRECEIVEBIN 0x00000006 //--Receive a packet of data, Mes point to BIN data area, data length is MesLength
#define VS_SOCKET_ONTIMER 0x00000007 //-- Timer messages, internal use. Mes = (VS_ULONG)TimerID,MesLength = 0

typedef void (SRPAPI *VS_SockEventCallBackProc)(VS_ULONG ServiceGroupID,void *Machine,VS_ULONG uMsg, VS_ULONG MachineID, VS_ULONG LinkInterfaceStatus, void *Mes, VS_INT32 MesLength, VS_ULONG Para);
typedef void (SRPAPI *VS_SockAcceptCallBackProc)(VS_ULONG ServiceGroupID,VS_ULONG uMsg, VS_ULONG ConnectionID, SOCKADDR_IN SockAddr, VS_ULONG Para, VS_ULONG MachineID, VS_SockEventCallBackProc *ClientCallBack, VS_ULONG *ClientPara); ConnectionID is id of server.
//---In Accept callback function, uMsg is Value of :VS_SOCKET_ONACCEPT or VS_SOCKET_ONDISCONNECT

1.1.30 Client login

#define VS_CLIENT_LOGIN 0x00000001
#define VS_CLIENT_LOADSERVICEOK 0x00000002
#define VS_CLIENT_LOGOUT 0x00000003 //-- Client connection is broken, CLE generates the callback before it is processed, the server can carry out some processing

typedef VS_BOOL (SRPAPI *VS_ClientMachineProcessProc)(void *Machine,void *Object,VS_ULONG Para,VS_ULONG uMes,VS_UUID *SrcServiceID, VS_ULONG SrcServiceAdd,VS_UINT16 SrcServicePort,class ClassOfSRPParaPackageInterface *ParaPkg ,VS_CHAR *UserName,VS_CHAR *UserPassword);
Machine : Corresponding state machine of the client, Para is parameter for the registration,
SrcServiceID: is source service id. If not redirected, this parameter is not a valid UUID.
ParaPkg: Para is parameter for the connection.
//-- Return parameter is only meaningful for VS_CLIENT_LOGIN. Returns VS_TRUE indicates application has deal with access requests. Return VS_FALSE indicates application does not deal with access requests.

1.2 Event constants

#define **VSMODULE_UNDEFINEPROC** Invalid process
#define **VSEVENTMANAGER_ATTACHBUFSIZE** associated buf size of event

1.2.1 Output event type definition (Note, current meaningless)

#define **VSEVENTTRIGGER_NORMAL** normal event

1.2.2 Event Processing results

#define **VSEVENTMANAGER_STOP** 1 // Stop dispatching events, subsequent events handler will not be triggered
#define **VSEVENTMANAGER_DISPATCH** 0 // Continue to dispatch events

1.2.3 Event handler prototype

typedef VS_INT32 (*__VSSystemEvent_EventProc__)(VS_ULONG FunctionChoice,void *EventPara); event handler
typedef void (*__VSSystemEvent_ParaFreeProc__)(void *EventRunParam);

1.2.4 Event request struct

```
typedef struct StructOfVSEventParamRunParam{
   VS_UWORD LParam;
   VS_UWORD SParam;
   VS_UWORD TParam;
   VS_UWORD FParam;
   VS_UWORD WParam;
   VS_UWORD Param6;
   VS_UWORD Param7;
   VS_UWORD Param8;
   VS_UWORD Param9;
   VS_UWORD Param10;
   VS_INT8 *AttachBuf;        /*  External trigger event processing set, and release automatically by the platform*/
#if defined(__cplusplus) || defined(c_plusplus)
   class ClassOfSRPParaPackageInterface *ParaPkg;  /*if not NULL, it will be released by platform*/
#else
   void *ParaPkg;
#endif
   VSSystemEvent_ParaFreeProc ParaFreeProc;  /* Events released, the release of the function is called to free outside of the parameters*/
   void *Context;                /*---The caller to set*/
   struct StructOfVSEventParamRunParam *Up,*Down;

   /*--version 2.2.0--*/
   VS_INT64 LParam_64;
   VS_INT64 SParam_64;
   VS_INT64 TParam_64;
   VS_INT64 FParam_64;
   VS_INT64 WParam_64;
   VS_INT64 Param6_64;
   VS_INT64 Param7_64;
   VS_INT64 Param8_64;
   VS_INT64 Param9_64;
   VS_INT64 Param10_64;
} VS_EVENTPARAM_RUNPARAM;
```

#define **INITVS_EVENTPARAM_RUNPARAM**(X) Initialize event request parameters
#define **INITVS_EVENTPARAM_COPYPARAM**(X,Y) Copy the event request parameter

1.2.5 Event struct

```
typedef struct Local_VSEventParam{
    void *SrcObject;
    void *DesObject;
    VS_BOOL ThisObject;              // If object which handles the event is same with DesObject, is true,
otherwise false.
    VS_EVENTPARAM_RUNPARAM RequestParam;      Request parameter
    VS_UUID EventID;
    VS_INT8 Reserver[112];           /
}VS_EVENTPARAM;
```

1.2.6 System object and its event

Only one system object, can not create new instances.

```
#define VSSYSOBJ_OBJNAME "VSSysObj"
#define VSSYSOBJ_WNDADJUSTNAME "OnWndAdjust"
#define VSSYSOBJ_WNDRESIZENAME "OnWndResize"
#define VSSYSOBJ_WNDCANBERESIZENAME "OnWndCanBeResize"
#define VSSYSOBJ_EDITSELECTNAME "OnEditSelect"
#define VSSYSOBJ_SETFOCUSNAME "OnSetFocus"
#define VSSYSOBJ_WNDMSGNAME "OnWinMsg"

extern VS_UUID VSSYSOBJ_OBJID;
//+++event
extern VS_UUID VSSYSOBJ_WNDADJUST;          //---Request to adjust the window client area
//--IN   none
//--OUT  none
extern VS_UUID VSSYSOBJ_WNDRESIZE;          //--- Management window size changes
//--IN   none
//--OUT  none
extern VS_UUID VSSYSOBJ_WNDCANBERESIZE;     //--- Determine whether to change the window size
//--IN   none
//--OUT  EventPara.LParam = 0 allow == 1 not allowed

extern VS_UUID VSSYSOBJ_EDITSELECT;
//--IN  EventPara.LParam = SelectObject

extern VS_UUID VSSYSOBJ_SETFOCUS;           //--- Request to set the focus

extern VS_UUID VSSYSOBJ_WNDMSG;             //---Windows Message
//+++ EventPara.LParam = uMes  EventPara.LParam = wParam  EventPara.LParam = lParam
//+++
```

System Doc Class:

```
#define VSSYSDOC_ONTEXTCHANGENAME "OnTextChange"
#define VSSYSDOC_ONTEXTSELECTNAME "OnTextSelect"
#define VSSYSDOC_ONGETTEXTNAME "OnGetText"
#define VSSYSDOC_ONSETTEXTNAME "OnSetText"
#define VSSYSDOC_LUA_GETTEXTNAME "Lua_GetText"
#define VSSYSDOC_LUA_SETTEXTNAME "Lua_SetText"

extern VS_UUID VSSYSDOC_CLASSID;
extern VS_UUID VSSYSDOC_ONGETTEXT;
```

```
//+++EventPara.LParam = class ClassOfSRPBinBufInterface *BinBuf  EventPara.SParam = 0 Merge  ==1
Refresh
extern VS_UUID VSSYSDOC_ONSETTEXT;
//+++Event:  EventPara.LParam = class ClassOfSRPBinBufInterface *BinBuf
extern VS_UUID VSSYSDOC_LUA_GETTEXT;
extern VS_UUID VSSYSDOC_LUA_SETTEXT;
extern VS_UUID VSSYSDOC_ONTEXTCHANGE;
//+++ Event:
extern VS_UUID VSSYSDOC_ONTEXTSELECT;
//+++ Event:  EventPara.LParam = StartPos SParam = EndPos
```

1.2.7 Platform statistics

```
typedef struct{
   VS_INT32    ClientConnectionNumber;   //--- The number of client connections
   VS_INT32    DebugConnectionNumber;    //--- The number of debug connections
   VS_INT32    ServerConnectionNumber;   //--- The number of server connections
   VS_INT32    DataConnectionNumber;     //--- The number of data connections
   VS_INT32    RawSocketServerNumber;    //--- The number of RawSocket server
   VS_INT32    RawSocketClientNumber;    //--- The number of RawSocket client

   //------ The following statistics are based on state machine
   VS_INT32    ReceiveMsgItemNumber;     //--- The number of messages received
   VS_INT32    ReceiveMsgItemBytes;      //--- The number of bytes received
   VS_INT32    SendMsgItemNumber;        //--- The number of messages sent
   VS_INT32    SendMsgItemBytes;         //--- The number of bytes sent
   VS_INT32    SysSendQueueOccupyRate;   //--- System send the buffer occupancy
   VS_INT32    ObjSendQueueOccupyRate;   //--- Object send the buffer occupancy

   VS_ULONG  PeerDelayTicket;          //--- Delay to the peer, (ms)
}VS_STATISTICINFO;
```

2 Event define

CLE is a distributed platform. Server and client synchronization through service item. Objects independent from each other. In the process, application can not expect other objects exist. References to other objects can not be assumed to exist, especially in the initial synchronization procedure.
In the following event processing, only VSEVENT_SYSTEMEVENT_ONBECOMESYNC and **VSEVENT_SYSTEMEVENT_ONACTIVATING** is safe. For **VSEVENT_SYSTEMEVENT_ONACTIVATING ,** Event processing can determine whether the object is in sync, and the existence of other objects, if these conditions are not meet, you can return failure. CLE will continue to activate the object subsequently; VSEVENT_SYSTEMEVENT_ONBECOMESYNC event is only generated to object itself. The event is triggered only once, when the cle becomes sync.

2.1 System event

“EventPara.ThisObject”, if DesObject is same with the object defines the handler, then is true, else is false.

2.1.1 before object first created

For global objects, the event is only at the server side. For local objects, the event can be triggered in the debugger, server and client (ie: only local objects create the event). The event is used to determine whether the object is allowed to create.

#define **VSEVENT_SYSTEMEVENT_ONBEFOREFIRSTCREATE**

//--IN EventPara.LParam = ParentAttributeIndex Index of parent queue
//--IN EventPara.SParam = *AttachBuf Object initialization parameter which may be NULL
//--IN EventPara.TParam = *NewObjectClassID (VS_UUID) Class id of the new object to be created
//--IN EventPara.FParam = 0, 1
//--IN EventPara.DesObject = ParentObject(FParam = 0) / Class(FParam = 1) SrcObject = NULL
//--OUT EventPara.LParam = = 0 Indicated allow to create, otherwise not allow to create, and it is error code

The event is only sent to the parent object of the object to be created.

2.1.2 object first created

The event is used to set initial parameters. Object under local control will trigger this event.

#define **VSEVENT_SYSTEMEVENT_ONFIRSTCREATE**
//--IN EventPara.LParam = *AttachBuf initialization parameters
//--IN EventPara.SParam = AttachBufSize initialization parameter size (bytes)
//--IN EventPara.TParam = ParentObject parent object
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn.

2.1.3 alloc memory

The event is used to initialize the object's memory. Only generated once
#define VSEVENT_SYSTEMEVENT_ONMALLOC
//--IN 无
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn.

2.1.4 free memory

The event is used to free the object's memory. Only generated once

#define VSEVENT_SYSTEMEVENT_ONFREE
//--IN 无
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn

2.1.5 create object[class or self]

The event is used to initialize the object's parameters.

#define VSEVENT_SYSTEMEVENT_ONCREATE
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn
In event hander, only its class object or parent object is safe. Other non-related class or object, might not exist at the moment

2.1.6 free object[class or self]

The event is used to clear the object's parameters.

#define **VSEVENT_SYSTEMEVENT_ONDESTROY**
//--OUT none
The event is send to all classes of the object. From root class, dispatch event in turn

2.1.7 create child object[class or self]

The event is used to notify parent object that a child object has been created. It is created after OnCreate event.

#define **VSEVENT_SYSTEMEVENT_ONCREATECHILD**
//--IN EventPara.LParam = ChildObject
//--IN EventPara.DesObject = Object SrcObject = NULL
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn.
The event will not be created unless child object event process mask is set.

2.1.8 destroy child object[class or self]

The event is used to notify parent object which a child object will be destroyed.

#define **VSEVENT_SYSTEMEVENT_ONDESTROYCHILD**
//--IN EventPara.LParam = ChildObject
//--IN EventPara.DesObject = Object SrcObject = NULL
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn.
The event will not be created unless child object event process mask is set.

2.1.9 before activating object[class or self]

The event is used to determine whether object can be activated or not

#define **VSEVENT_SYSTEMEVENT_ONACTIVATING**
//--IN EventPara.LParam = 0 general activate, 1 re-activate for loading object
Under normal circumstances, this parameter is 0, but when calls object loading function LoadFromBuf / File, the object will re-generated this event with the argument is set to 1.
/ / - OUT EventPara.LParam == 0 Indicated successful activation.
Otherwise, the activation fails (no response, is also considered to be successful)

The event is send to all classes of the object. From root class, dispatch event in turn.

2.1.10 before deactivate object[class or self]

The event is used to clear parameters when activating.

#define **VSEVENT_SYSTEMEVENT_ONDEACTIVATING**
//--IN EventPara.LParam = 0 general deactivate, 1 deactivate for loading object

Under normal circumstances, this parameter is 0, but when calls object loading function LoadFromBuf / File, the object will re-generated the event with the argument is set to 1.
//--OUT none
The event is send to all classes of the object. From root class, dispatch event in turn.
The event is only generated at synchronization state

2.1.11 activate object[class or self]

The event is used to initialize parameters after the activation.

#define **VSEVENT_SYSTEMEVENT_ONACTIVATE**
//--IN //--IN EventPara.LParam = 0 general activate 1 re-activate for loading object
Under normal circumstances, this parameter is 0, but when calls object loading function LoadFromBuf / File, the object will re-generated the event with the argument is set to 1.
//--OUT none
The event is send to all classes of the object. From root class, dispatch event in turn.
The event is only generated at synchronization state

2.1.12 deactivate object[class or self]

The event is used to clear parameters after the activation.

#define **VSEVENT_SYSTEMEVENT_ONDEACTIVATE**
//--IN EventPara.LParam = 0 general deactivate 1 re-deactivate for loading object
Under normal circumstances, this parameter is 0, but when calls object loading function LoadFromBuf / File, the object will re-generated the event with the argument is set to 1.
//--OUT none
The event is send to all classes of the object. From root class, dispatch event in turn.

2.1.13 activate child object[class or self]

The event is triggered after one child object is activated

#define **VSEVENT_SYSTEMEVENT_ONACTIVATECHILD**
//--IN EventPara.LParam = ChildObject
//--IN EventPara.DesObject = Object SrcObject = NULL
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn.
The event will not be created unless child object event process mask is set.

2.1.14 deactivate child object[class or self]

The event is triggered after one child object is deactivated

#define **VSEVENT_SYSTEMEVENT_ONDEACTIVATECHILD**
//--IN EventPara.LParam = ChildObject
//--IN EventPara.DesObject = Object SrcObject = NULL
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn.
The event will not be created unless child object event process mask is set.

2.1.15 before object attribute change[class or self]

The event is triggered before object attribute will be changed.
[Only generated to object defines the attribute which is marked to generate this event]

#define **VSEVENT_SYSTEMEVENT_ONATTRIBUTEBEFORECHANGE**
//--IN EventPara.LParam = AttributeIndex
//--IN EventPara.SParam = 0 reserved
//--IN EventPara.TParam = NewValue //---address of new value, for global pointer attribute, it is the addressof ClassID of the new object which will be created.
//--IN EventPara.FParam = DebugEditFlag; //---==0 normal ==1 changed by editor
//--IN EventPara.DesObject = Object SrcObject = NULL
//--OUT EventPara.LParam == 0 allow to change, otherwise not allow and is the errorcode. The return value is meaningful when object is under local control.
//--the corresponding attribute must be declared to accept this event, which can be defined using service description file or function "CreateAtomicAttribute"

2.1.16 object attribute change[class or self]

The event is triggered before object attribute is changed.
[Only generated to object defines the attribute which is marked to generate this event]

#define **VSEVENT_SYSTEMEVENT_ONATTRIBUTECHANGE**

//--IN EventPara.LParam =AttributeIndex
//--IN EventPara.SParam = (VS_ATTRIBUTEINDEXMAP *)AttributeIndexMap
//--OUT none
The handler may use AttributeIndex or AttributeIndexMap to help to process the event.
//--the corresponding attribute must be declared to accept this event, which can be defined using service description file or function "CreateAtomicAttribute"

2.1.17 before parent object change[class or self]

#define **VSEVENT_SYSTEMEVENT_ONPARENTBEFORECHANGE** 0x00000016
//--IN EventPara.LParam = (VS_ULONG)*ParentObject
//--IN EventPara.DesObject = Object SrcObject = NULL
//--OUT EventPara.LParam == 0 allow change, otherwise not allow and is the errorcode. The return value is meaningful when object is under local control.
The event is send to all classes of the object. From root class, dispatch event in turn.

2.1.18 parent object change[class or self]

#define **VSEVENT_SYSTEMEVENT_ONPARENTCHANGE** 0x00000017
//--IN EventPara.DesObject = Object SrcObject = NULL
//--OUT 无
The event is send to all classes of the object. From root class, dispatch event in turn.

2.1.19 object static data change[class or self]

#define **VSEVENT_SYSTEMEVENT_ONSTATICCHANGE**
//---Object static data change which only trigger to the class which defines the static data attribute.
//--IN EventPara.LParam = AttributeIndex

//-- Must be processed immediately

2.1.20 object name script change

#define **VSEVENT_SYSTEMEVENT_ONSCRIPTCHANGE**
//--IN EventPara.LParam = ScriptName
//--IN EventPara.SParam = Operation: 0 - Change 1 Create
//-- Must be processed immediately

2.1.21 object sync status change

#define **VSEVENT_SYSTEMEVENT_ONBECOMESYNC**
//-- Must be processed immediately
//---Generated when object changes to synchronization status, which is used to process initialization which should be done in synchronization status.
//----This initialization should be processed only once. Together with VSEVENT_SYSTEMEVENT_ONMODULEINIT to complete object initialization.
//----The event is only triggered to object self.
//----The event is triggered only once when object's status change to sync.
//----The event is not triggered to script function.

2.1.22 object sync group change

#define **VSEVENT_SYSTEMEVENT_ONSYNCGROUPCHANGE**
//--OUT none

2.1.23 child object sync group change

#define **VSEVENT_SYSTEMEVENT_ONCHILDSYNCGROUPCHANGE**
//--IN EventPara.LParam = ChildObject
//--OUT none

2.1.24 service item active set change[The event is triggered when system event process flag VSSYSEVENT_PROCESS_ACTIVESET is set]

#define **VSEVENT_SYSTEMEVENT_ONACTIVESETCHANGE**
//--IN EventPara.LParam = SysRootItem service item address
//--OUT 无

2.1.25 save object

#define **VSEVENT_SYSTEMEVENT_ONSAVE**
//--IN 无
//--OUT EventPara.LParam = Buf should be allocated using interface function Malloc. The memory will be freed by CLE.
//--OUT EventPara.SParam = BufSize memory size
//--OUT EventPara.TParam = 0 success, otherwise means failure

2.1.26 load object

#define **VSEVENT_SYSTEMEVENT_ONLOAD**
//--IN EventPara.LParam = Buf memory address
//--IN EventPara.SParam = BufSize memory size

//--OUT EventPara.LParam = 0 success, otherwise means failure

2.1.27 get object load mask

#define **VSEVENT_SYSTEMEVENT_ONLOADMASK** 0x00000052
//--IN EventPara.LParam = &VS_ATTRIBUTEINDEXMAP only need set AppAttributeMap. If bit is set, the corresponding attribute will not be covered at load. Application may use macro VS_SETAPPATTRMAP
//--OUT none

2.1.28 load object finish

#define **VSEVENT_SYSTEMEVENT_ONLOADFINISH**
//--IN none
//--OUTnone

2.1.29 receive peer message

#define **VSEVENT_SYSTEMEVENT_ONREMOTESEND**
//--IN EventPara.LParam = VS_PARAPKGPTR message
//--OUT 无

2.1.30 function call event

#define **VSEVENT_SYSTEMEVENT_ONCALL**
//--IN EventPara.LParam = ClassOfSRPFunctionParaInterface *
//--IN EventPara.SParam = FunctionName
//--IN EventPara.TParam = FunctionID(VS_UUID *)
//--OUT EventPara.LParam = Reault if type is VS_FLOAT, should use ((VS_FLOAT *)&LParam)[0]
//--OUT EventPara.SParam = RetType, if no return value, then RetType is set to VSTYPE_IGNORE and LParam is set to 0
//--ClassOfSRPFunctionParaInterface: is freed by CLE

2.1.31 Ticket event-only created to self

The event is triggered every 10ms.

#define **VSEVENT_SYSTEMEVENT_ONTICKET**
//--IN EventPara.LParam = CurrentTicket
//-- Must be processed immediately

2.1.32 frame pulse event - only created to self

The event is created by server. The interval is set by service.

#define **VSEVENT_SYSTEMEVENT_ONFRAMETICKET**
//--IN EventPara.LParam = CurrentTicket
//--IN EventPara.SParam = FrameTimer frame pulse counter
//-- Must be processed immediately

2.1.33 idle event- only created to self

//----Application is idle, not process any message
#define **VSEVENT_SYSTEMEVENT_ONIDLE**
//--IN EventPara.LParam = CurrentTicket
//-- Must be processed immediately
//--OUT EventPara.LParam = 1 continue to create IDLE event

2.1.34 application active event- only created to self

#define **VSEVENT_SYSTEMEVENT_ONAPPACTIVE**
//-- Must be processed immediately
The event is triggered to active service

2.1.35 application deactive event - only created to self

#define **VSEVENT_SYSTEMEVENT_ONAPPDEACTIVE**
//-- Must be processed immediately
The event is triggered to active service

2.1.36 service active event- only created to self

#define **VSEVENT_SYSTEMEVENT_ONSERVICEACTIVE**
//-- Must be processed immediately
The event is triggered to active service

2.1.37 service deactive event- only created to self

#define **VSEVENT_SYSTEMEVENT_ONSERVICEDEACTIVE**
//-- Must be processed immediately
The event is triggered to active service

2.1.38 object module init

#define **VSEVENT_SYSTEMEVENT_ ONMODULEINIT**
//--IN none
//-- Must be processed immediately

2.1.39 object module term

#define **VSEVENT_SYSTEMEVENT_ ONMODULETERM**
//--IN none
//-- Must be processed immediately

2.1.40 object module can be unload

#define **VSEVENT_SYSTEMEVENT_ ONMODULECANBEUNLOAD**
//--IN none
//--OUT EventPara.LParam = return result, ==0 can be unload ==1 should wait
//-- Must be processed immediately

2.1.41 event request parameter to script parameter

#define **VSEVENT_SYSTEMEVENT_ONVSTOSCRIPTINPUTPARA** //----translate to script parameter

//--IN EventPara.LParam = (VS_ULONG)&EventUUID
//--IN EventPara.SParam = (VS_ULONG)&EventParam
//--OUT EventPara.LParam = result, ==1 convert successful ==0 not convert
If no response is returned, default is successful.
The event is triggered before dispatching to script function. The handler may push parameter on Lua stack using Lua functions of CLE.

2.1.42 script parameter to event request parameter

#define **VSEVENT_SYSTEMEVENT_ONSCRIPTTOVSINPUTPARA** //---- translate to event parameter
//--IN EventPara.LParam = (VS_ULONG)&EventUUID
//--IN EventPara.SParam = (VS_ULONG)&EventParam
//--OUT EventPara.LParam = result, ==1 convert successful ==0 not convert
If no response is returned, default is successful.
The event is triggered when event is created by script function. Value of Lua stack top is parameter number, and in turn is parameter 0, parameter 1,.... The event handler should fetch the value, and assign it to field EventParam.RequestParam.

2.1.43 event output parameter to script parameter

#define **VSEVENT_SYSTEMEVENT_ONVSTOSCRIPTOUTPUTPARA** //----translate to script parameter
//--IN EventPara.LParam = (VS_ULONG)&EventUUID
//--IN EventPara.SParam = (VS_ULONG)&EventParam
//--OUT EventPara.LParam = result, ==1 convert successful ==0 not convert
If no response is returned, default is successful.
Called after event processing. The parameters were converted and saved in the Lua stack.. The arguments are pushed into the Lua stack using Lua interface functions

2.1.44 script parameter to event output parameter

#define **VSEVENT_SYSTEMEVENT_ONSCRIPTTOVSOUTPUTPARA**
//--IN EventPara.LParam = (VS_ULONG)&EventUUID
//--IN EventPara.SParam = (VS_ULONG)&EventParam
//--OUT EventPara.LParam = result, ==1 convert successful ==0 not convert
If no response is returned, default is successful.
Called after event processing. Value on top of the lua stack is the number of return value, which is value 0,value 1,...,and argment number in turn. When converting, fetch value, call function pSRP -> GetResponseBuf( ) to get a response buffer, fill the buffer, and then use function pSRP -> AttachResponseBuf to assign the buffer to EventParam.

2.2 Object edit event

2.2.1 edit query event

The event is used to determine whether object can be edit or not.

#define **VSEVENT_SYSTEMEDIEEVENT_ONQUERYEDIT**

//--IN DesObject = Object SrcObject = NULL;
//--IN EventPara.ThisObject == true self ==false instance
//--OUT EventPara.LParam = 0edit function not exist; ==1 edit function exist.

2.2.2 edit fill object

The event is used to fill init parameters of the new object to be created.

#define **VSEVENT_SYSTEMEDIEEVENT_ONEDITFILLADDBUF**
//--IN EventPara.LParam = *AttachBuf
//--OUT EventPara.LParam = 0 not processing; else is the size of filled data

2.2.3 object edit event

The event is used to launch external editor

#define **VSEVENT_SYSTEMEDIEEVENT_ONEDIT**

//--IN EventPara.LParam = *EditObject object to be edited
//--IN EventPara.SParam = AppData; defined by app, which is used to init editor status; when the event is triggered at debugerver, it will be assign to 0
//--IN EventPara.TParam = AppData1; defined by application, which is used to init editor status; when the event is triggered at debugerver, it will be assign to 0
//--IN DesObject = Object SrcObject = NULL;
//--IN EventPara.ThisObject == true self; ==false instance

2.2.4 object term edit event

#define **VSEVENT_SYSTEMEDIEEVENT_ONTERMEDIT**
//--IN DesObject = Object SrcObject = NULL;
//--IN EventPara.ThisObject == true; self ==false instance

2.2.5 object select event

The event is triggered when object is being selected

#define **VSEVENT_SYSTEMEDIEEVENT_ONEDITSELECT**
//--IN EventPara.LParam = (VS_UUID *)SelectObject
//--IN DesObject = Object SrcObject = NULL; //--DesObject is parent object, LParam is current choiced object.

2.2.6 object button event

The event is triggered when button of attribute of object is pressed.

#define **VSEVENT_SYSTEMEDIEEVENT_ONEDITEBUTTON**
//--IN EventPara.LParam = (VS_INT8 *)Buf, buffer to save result
//--IN EventPara.SParam = * OBJECTATTRIBUTEINDEX; //--attribute index
//--IN EventPara.TParam = AttributeDeepth; //--
//--IN DesObject = Object SrcObject = NULL;

2.2.7 object attribute display event

The event is trigger when debugserver needs to show object attribute.

#define **VSEVENT_SYSTEMEDIEEVENT_ONEDITSHOW**
//--IN EventPara.LParam = (VS_INT8 *)Buf, buffer used to save display string of attribute
//--IN EventPara.SParam = * OBJECTATTRIBUTEINDEX; //-- attribute index
//--IN EventPara.TParam = AttributeDeepth; //--
//--IN DesObject = Object SrcObject = NULL;

2.2.8 object preview event

The event is triggered when debugserver needs to preview the object.

#define **VSEVENT_SYSTEMEDIEEVENT_ONEDITPREVIEW**
//--IN EventPara.LParam = HDC //--window handle
//--IN EventPara.SParam = VS_RECT * //--display rect : Left,Top; Right 和 Bottom
//--IN DesObject = Object SrcObject = NULL;

2.2.9 object attribute can be edit event

The event is used to determine whether the attribute can be edit or visible.

#define **VSEVENT_SYSTEMEDIEEVENT_ONATTRIBUTEEDITFLAG**
//--IN EventPara.LParam = AttriubteIndex
//--IN DesObject = Object SrcObject = NULL;
//--OUT EventPara.LParam

#define VSSYSTEMEDIT_ATTRIBUTEEDITFLAG_NONE set based on service
#define VSSYSTEMEDIT_ATTRIBUTEEDITFLAG_INVISIBLE invisible and can not be edited
#define VSSYSTEMEDIT_ATTRIBUTEEDITFLAG_READONLY visible and can not be edited
#define VSSYSTEMEDIT_ATTRIBUTEEDITFLAG_EDIT can be edited

2.2.10 object first create event

The event is used to determine whether the object can be created.

#define **VSEVENT_SYSTEMEDIEEVENT_ONBEFOREFIRSTCREATE**
//--IN EventPara.LParam = ParentAttributeIndex
//--IN EventPara.SParam = *AttachBuf // may be NULL
//--IN EventPara.TParam = *NewObjectClassID (VS_UUID)
//--IN EventPara.FParam = ==0 : globaly created ==1 : locally created
//--IN EventPara.DesObject = ParentObject SrcObject = NULL
//--OUT EventPara.LParam == 0 allow to create; otherwise not allow and is the error code

3 ClassOfSRPControlInterface

The interface can be obtained through BasicSRPInterface, which provides the following functions:
class definition : **class ClassOfSRPControlInterface**

3.1 Get system type

3.1.1 get system type -GetOsType

VS_UINT32 SRPAPI **GetOsType**( )

3.2 CLE lock

3.2.1 lock CLE-SRPLock

void SRPAPI **SRPLock**( )

3.2.2 unlock CLE-SRPUnLock

void SRPAPI **SRPUnLock**( )

3.3 Global flow control

3.3.1 SRP message dispatch-SRPDispatch

VS_BOOL SRPAPI **SRPDispatch**(VS_BOOL WaitFlag); WaitFlag = true, if there are no messages in the queue, current thread will be suspended to wait.
return : true: Indicated there are messages have been handled, false : there is no message to be handled.
Note: after starcore initialization, the thread is in lock status. This function first unlock the thread. Therefore calling the function should be in the main thread, otherwise unlock and lock thread will be different.
If calling the function in other thread, SRPLock should be called first to get the lock.

3.3.2 create SRPIDLE event-SRPIdle

VS_BOOL SRPAPI **SRPIdle**();
When return value is true, there is no need to create idle event, or else, should continue create the idle event.

3.3.3 create SRPAppActive event-SRPAppActive

void SRPAPI **SRPAppActive**();

3.3.4 create SRPAppDeactive event -SRPAppDeactive

void SRPAPI **SRPAppDeactive**();

3.4 print error information -ProcessError

void **ProcessError**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format,…);
void **ProcessErrorVar**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format ,va_list argList);
void **ProcessLuaError**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format,…);

void **ProcessLuaErrorVar**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format ,va_list argList);

3.4.1 capture Lua display information-CaptureLuaDisp/ ReleaseLuaDisp

void SRPAPI **CaptureLuaDisp**(VS_LuaInfoDispProc DispProc,VS_ULONG Para);
void SRPAPI **ReleaseLuaDisp**(VS_LuaInfoDispProc DispProc,VS_ULONG Para);

typedef void (SRPAPI *VS_LuaInfoDispProc)(VS_CHAR *DispInfo,VS_ULONG Para);

3.5 Lua script pre-compile, edit, and execute

3.5.1 PreCompile

VS_BOOL SRPAPI **PreCompile**(VS_CHAR * ScriptInterface,VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize, VS_CHAR *Name, VS_CHAR **ErrorInfo);
If returns VS_FALSE, the caller should use ErrorInfo to determine whether the call is successful or not. If ErrorInfo returns NULL,then the input is incomplete, and futher input is expected; If ErrorInfo is not NULL, then the input contains errors.
If return value is VS_TRUE, then ErrorInfo will be set to NULL.
ScriptInterface: may be lua,python, or other online script language registered.
ScriptInterfaces length < 16

3.5.2 Open editor-OpenLuaEdit[win32]

VS_BOOL SRPAPI **OpenLuaEdit**(VS_CHAR *Module,VS_ULONG Config,VS_BOOL CloseEnable);
editor is compiled into sharelib SRPLuaEdit.DLL
Config is combination of the following values :
#define SRPLUAEDITMODULECONFIG_SCRIPTCONSOLE 0x00000001
#define SRPLUAEDITMODULECONFIG_PROJECT 0x00000002
#define SRPLUAEDITMODULECONFIG_SRPDOC 0x00000004
Module is reserved.

3.5.3 display information in editor -LuaEditDisp

void SRPAPI **LuaEditDisp**(VS_CHAR *Info);

3.5.4 display help information in editor - LuaEditHelp

void SRPAPI **LuaEditHelp** (VS_INT32 Type,VS_CHAR *HelpInfo);
Type=0 help info =1 Examples project

3.5.5 close editor -CloseLuaEdit

void SRPAPI **CloseLuaEdit**();

3.5.6 DoBuffer

VS_BOOL SRPAPI **DoBuffer**(VS_CHAR *ScriptInterface, VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize,VS_BOOL IsUTF8 , VS_CHAR *ModuleName);
void SRPAPI **PostDoBuffer**(VS_CHAR *ScriptInterface, VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize,VS_BOOL IsUTF8 , VS_CHAR *ModuleName);
PostDoBuffer runs in message loop of the cle

If ScriptInterface is NULL,default language is lua. If may be lua,python, or other online script language registered.
ScriptInterfaces length < 16
ModuleName is name of the module,may be NULL.
ModuleName should not set to “cmd”, case insensitive.

3.6 Service related functions

3.6.1 clear current service -ClearService

void SRPAPI **ClearService**( );

3.7 Run script file

3.7.1 run script file-DoFile

VS_BOOL SRPAPI DoFile(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8);
VS_BOOL SRPAPI DoFileEx(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8,VS_CHAR * ModuleName);
VS_BOOL SRPAPI PostDoFile(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8);
VS_BOOL SRPAPI PostDoFileEx(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8,VS_CHAR * ModuleName);
The function reads file content, and then it is same as PostDoBuffer
ScriptInterfaces length < 16
python,lua does not support parameter IsUTF8. Whether other scripts support IsUTF8 or not depends on their implementation.
ModuleName is name of the module,may be “” or NULL. This Parameter is only valid for lua and python and java and csharp. For java and csharp, ModuleName is the init class name. For example, the ModuleName is “com.srplab.www.test”.
ModuleName should not set to “cmd”, case insensitive.

3.8 BasicInterface function

3.8.1 Get Basic Interface-QueryBasicInterface

class ClassOfBasicSRPInterface *SRPAPI **QueryBasicInterface**(VS_ULONG ServiceGroupID);
Service group 0 is created by default.

3.8.2 Create Basic Interface-CreateBasicInterface

class ClassOfBasicSRPInterface *SRPAPI **CreateBasicInterface**(VS_ULONG ServiceGroupID, VS_UINT16 ProgramRunType);
ProgramRunType may take value from VS_SERVER or VS_CLIENT, which represents server service group or client service group. Note: VS_SERVER is not supported in current version.
If ServiceGroupID==0, cle will assign one automaticly, or else if the service group has existed, NULL will be returned.

3.8.3 Delete Basic interface-DeleteBasicInterface

void SRPAPI **DeleteBasicInterface**(VS_ULONG ServiceGroupID);
Service group 0 can not be deleted.

3.8.4 Query Basic interface-QueryFirstServiceGroup

VS_ULONG SRPAPI **QueryFirstServiceGroup**( );
VS_ULONG SRPAPI **QueryNextServiceGroup**( );
If the return value is VS_INVALID_SERVICEGROUPID, then there are no more service groups.

3.9 Lua function

3.9.1 get Lua stack-GetLuaStack

void *SRPAPI **GetLuaStack**( );

3.9.2 get integer from table-LuaGetTableInt

VS_ULONG SRPAPI **LuaGetTableInt**(void *L, VS_INT32 Index, VS_CHAR *ValueName);

3.9.3 get integer-LuaGetInt

VS_ULONG SRPAPI **LuaGetInt**(void *L, VS_INT32 Index);

3.9.4 get UpvalueIndex-LuaUpValueIndex

VS_INT32 SRPAPI **LuaUpValueIndex**(void *L, VS_INT32 Index);

3.9.5 get object's service group-GetObjectServiceGroupID

VS_ULONG SRPAPI **GetObjectServiceGroupID**(void *Object);

3.10 Get current Url

3.10.1 get current Url-GetUrl

void SRPAPI GetUrl(VS_CHAR *Buf,VS_INT32 BufSize);

3.11 set program run type (valid at server)

Curent service will be unloaded if the program type changes.

3.11.1 set program run type-SetProgramType[Reserved]

void SRPAPI SetProgramType(VS_UINT16 Type)
Type takes value from VS_SERVER_SERVER/VS_SERVER_USER.

3.11.2 get program run type-GetProgramType

VS_UINT16 SRPAPI GetProgramType();
Return value is VS_SERVER_SERVER/VS_SERVER_USER.

3.12 SRP application packing interface (valid at server)

3.12.1 query published services-SRPBuild_QueryPublicService

VS_BOOL SRPAPI SRPBuild_QueryPublicService(SRPBuild_QueryPublicServiceCallBackProc QueryPublicServiceCallBackProc,VS_ULONG CallBackPara,VS_BOOL FillUpdateInfo,SRPBuild_PrintProc PrintProc,VS_ULONG Para)
VS_BOOL SRPAPI SRPBuild_QueryPublicServiceEx(VS_CHAR *Url,SRPBuild_QueryPublicServiceCallBackProc QueryPublicServiceCallBackProc,VS_ULONG CallBackPara,VS_BOOL FillUpdateInfo,SRPBuild_PrintProc PrintProc,VS_ULONG Para)

This function is an asynchronous function. When download finish, callback function will be called. Print function is called during download procedure.
If you have run a query before, then this query will fail.
void (SRPAPI *SRPBuild_QueryPublicServiceCallBackProc)(VS_BOOL Result, VS_ULONG Para, struct VSPublicServiceDef *PublicServiceList);
struct VSPublicServiceDef {
  VS_CHAR ServiceName[DEFAULT_NAMELENGTH];
  VS_BOOL NeedUpdateFlag;
  VS_CHAR ServiceInfo[256];
};
If you set FillUpdateInfo, NeedUpdateFlag flag will be filled in return value.

3.12.2 start packing-SRPBuild_Start

VS_BOOL SRPAPI SRPBuild_Start(VS_CHAR *Name,VS_CHAR *Path,VS_BOOL SingleFlag,VS_BOOL ForceToDownLoadPublicService,SRPBuild_PrintProc PrintProc,VS_ULONG Para,struct VSImportServiceDef *PublicServiceList,VS_BOOL ExeFileFlag,VS_CHAR *ScriptInterface,VS_UINT32 SupportOsType);
Name: application name
Path: output path
SingleFlag: If equals to VS_TRUE, then output is a single file with suffix .srb; otherwise output results to a directory.
ForceToDownLoadPublicService: Force to download service published
PrintProc: display information callback function, prototype is
  typedef void (SRPAPI *SRPBuild_PrintProc)(VS_ULONG Para, VS_CHAR *Info);
PublicServiceList: service list published
ExeFileFlag: Meaningful when the SingleFlag == VS_TRUE. If it is VS_TRUE, then output is an executable file.
ScriptInterface: NULL, lua, python,or other script interface registered. String length should be less than 16 bytes.
SupportOsType: is combination of OS type supported.
for example:VSOS_WIN32 | VSOS_LINUX, If equals to 0, means all types are supported.

3.12.3 set packed service file-SRPBuild_InsertServiceFile

void SRPAPI SRPBuild_InsertServiceFile(VS_CHAR *DiskFileName,VS_CHAR *OutFileName ,VS_BOOL StartFileFlag,VS_BOOL ToUTF8,VS_UINT32 SupportOsType);
If ToUTF8 is set to VS_TRUE, then file will be coding to UTF8 be cle when packing.

StartFileFlag: Start file for the package. An OS type, should only set one.
Start file may be sharelib file, python, lua, java or csharp script file.
SupportOsType: is the combination of OS type supported. for example:VSOS_WIN32 | VSOS_LINUX, If equal to 0, means all types of OS.

3.12.4 set depended service-SRPBuild_InsertDependFile

void SRPAPI SRPBuild_InsertDependFile(VS_CHAR *Path,VS_CHAR *DependName);

3.12.5 set static data file-SRPBuild_InsertStaticDataFile

void SRPAPI SRPBuild_InsertStaticDataFile(VS_CHAR *DiskFileName,VS_CHAR *OutFileName ,VS_BOOL ToUTF8);
If ToUTF8 is set to VS_TRUE, then file will be coded to UTF8 by cle when packing.

3.12.6 set dynamic data file-SRPBuild_InsertDynaDataFile

void SRPAPI SRPBuild_InsertDynaDataFile(VS_CHAR *DiskFileName,VS_CHAR *OutFileName ,VS_BOOL ToUTF8);
If ToUTF8 is set to VS_TRUE, then file will be coded to UTF8 by cle when packing.

3.12.7 execute packing-SRPBuild_Execute

VS_BOOL SRPAPI SRPBuild_Execute();

3.13 UUID function

3.13.1 conversion function

```
VS_BOOL SRPAPI StringToUuid(VS_INT8 *String,VS_UUID *Uuid);
VS_INT8 *SRPAPI UuidToString(VS_UUID *Uuid);
VS_BOOL SRPAPI MD5ToUuid(VS_INT8 *String,VS_UUID *Uuid);
VS_INT8 *SRPAPI UuidToMD5(VS_UUID *Uuid);
VS_INT8 *SRPAPI GetMD5(VS_INT8 *Buf,VS_INT32 BufSize);

void SRPAPI CreateUuid(VS_UUID *UuidPtr);
```

3.14 Get other interface

3.14.1 get SXML interface

class ClassOfSRPSXMLInterface *GetSXMLInterface();

3.14.2 get comm interface

class ClassOfSRPCommInterface *SRPAPI GetCommInterface()

3.15 Script interface

VS_BOOL SRPAPI RegScriptInterface(VS_CHAR * ScriptInterface,struct StructOfVSScriptContext *ScriptContext,VS_ULONG Para,StarCoreScript_TermProc TermProc);
VS_BOOL SRPAPI UnRegScriptInterface(VS_CHAR *ScriptInterface,struct StructOfVSScriptContext *ScriptContext,VS_ULONG Para,StarCoreScript_TermProc TermProc);

TermProc is used to clear environment before end of the script. Different TermProc represents different script interface. CLE uses the interface which TermProc is last registered for the same interface.
Generally do not take the initiative to call UnRegScriptInterface. The registed script interface will be removed by the CLE before exit automatically.
UnRegScriptInterface may cause TermProc callback function is called.

VS_BOOL SRPAPI ActiveScriptInterface(VS_CHAR *ScriptInterface ,VS_BOOL *OnLineScriptFlag,void *VirtualMachine);
VS_CHAR *SRPAPI FirstScriptInterface(VS_QUERYRECORD *QueryRecord);
VS_CHAR *SRPAPI NextScriptInterface(VS_QUERYRECORD *QueryRecord);

“lua” and “python” need not be registered.
ScriptInterfaces length < 16
Scripting interface can be set via the configuration file starenvcfg.xml.

```
<?xml version="1.0" encoding="utf-8" ?>
<StarCoreEvnConfig Python="">
Python:Python Shared library name, if not set, for Win32, default is python27.dll, and for linux, default is libpython2.7.so. If you specify the shared library name,you should comply with this rules.
   <ExternScript>
      <script name="" Module="dll/so" para="XXX"/>
   </ExternScript>
   ExternScript: configuration script interpreter, which can be in addition to lua, python, or other types of scripts. If the last character of the Module is '/' or '\', then it is a path.  Script interpreter will be searched under the path. Default name is star_XXX.dll / libstar_XXX.so.
</StarCoreEvnConfig>
```

If not configured, then search star_ (ScriptInterface). Dll (windows), or libstar_ (ScriptInterface). So (linux) file to load

VirtualMachine：is address of virtual machine, may be set to NULL.

VS_BOOL SRPAPI **SetScriptInterface**(VS_CHAR *ScriptInterface,VS_CHAR *Module,VS_CHAR *Para);
This Function is provided to set script interface module dynamically

Module can be input “”
for java:
Para might be used to set jvm share library and class path, “jvm=XXXX”,” java.class.path=XXX”, ” java. library.path=XXX”, or both, separated by “,”. For example:
_SetScript("java","","jvm=C:\\Program Files\\Java\\jdk1.6.0_21\\jre\\bin\\server\\jvm.dll, java.class.path=.;c:\\srplab\\libs\\starcore.jar")

if java.library.path is set, star_java.dll/libstar_java.so should be in the path.

for python:

-S : (Py_NoSiteFlag)
-d : (Py_DebugFlag)
-s : (Py_NoUserSiteDirectory)
-v : (Py_VerboseFlag)
-i : (Py_InspectFlag)
-B : (Py_DontWriteBytecodeFlag)

forexample: "-S –v"

for ruby:
-m/-M: ruby share library name with full path
-v/-V : version of ruby, default is "1.9.3".

for other languages:

Para is ignored

3.16 Temporary file register

VS_BOOL SRPAPI RegTempFile(VS_CHAR *TempFileName,VS_CHAR *OriFileName); /*clear when process not exist*/
OriFileName: may be NULL
TempFileName: Is unique among multiple processes. TempFileName should be full path name.

VS_CHAR *SRPAPI GetRegTempFile(VS_CHAR *OriFileName ,VS_CHAR *Buf,VS_INT32 BufSize);
Obtain temporary file name registered by other process.
If extsts, then the calling process is automatically registered.

void SRPAPI UnRegTempFile(VS_CHAR * TempFileName); /*clear when process not exist*/
Unregister the temporary file. The file will be deleted by CLE.

3.17 Get cle config information

void SRPAPI GetConfigResult(VS_BOOL *DebugCfgResult,VS_BOOL *DirectClientCfgResult,VS_BOOL *TelnetCfgResult,VS_BOOL *WebServerCfgResult);
get config result

VS_CHAR *SRPAPI GetConfigEnvTag( );
Get config env tag which is set through VS_STARCONFIGEX at cle init procedure.
Currently, three predefined:
"", "noloop", "activex"

3.18 Replication

class ClassOfSRPControlInterface *SRPAPI Dup();

3.19 Get Interface

void *SRPAPI QueryInterface(VS_UUID *InterfaceID);
The input is ID of the interface. This function is reserved to extend

3.20 Set log file

void SRPAPI SetLogFile(VS_CHAR *FileName,VS_BOOL LogAll);

If FileName is NULL, the log is canceled.

LogAll:VS_TRUE, log all print information, or else, only log warning and error information.

3.21 Authorize

void SRPAPI GetSystemRegCode(VS_CHAR *Buf);
get local serial number
Note: From Version 2.0，this function should not be used.

VS_BOOL SRPAPI SetRegisterCode(VS_CHAR *Buf,VS_BOOL Single);
Note: From Version 2.0， Single is ignored.
The function should be called after starcore init and before creating any service.

VS_BOOL SRPAPI IsRegistered();
Whether CLE is registered.

VS_BOOL SRPAPI PreAuthorize(VS_CHAR *ServiceName,VS_UUID *ServiceID,VS_CHAR *Buf,VS_BOOL Single);
Note: From Version 2.0，ServiceName,ServiceID,and Single is ignored.
This function is depreciated

3.22 Set Locale of cle

void SRPAPI SetLocale(VS_CHAR *Lang);
VS_CHAR *SRPAPI GetLocale();
for windows:
Lang should be “utf8” or “ansi”,or “.XXX”, XXX is number of codepage, such as “.950”

3.23 Set/Get Script Interface Index

void SRPAPI SetScriptInterfaceIndex(VS_CHAR *ScriptInterfaceName);
VS_INT32 SRPAPI GetScriptInterfaceIndex(VS_CHAR *ScriptInterfaceName);

3.24 DetachCurrentThread

void SRPAPI DetachCurrentThread()
if c/c++ moule calls object’s script function in some thread, it should call this function before thread exist.

3.25 Reference Count

void SRPAPI AddRef();
Increase reference count;

VS_INT32 SRPAPI GetRef();
Get current reference count

void SRPAPI ReleaseOwner();
Release owner, this function will decrease reference count, but not free the instance.

3.26 Get Last Error

VS_INT32 SRPAPI GetLastError();
VS_CHAR *SRPAPI GetLastErrorInfo(VS_UINT32 *LineIndex,VS_CHAR **SourceName);

3.27 Set Operation Path

VS_CHAR *SRPAPI SetCoreOperationPath(VS_CHAR *Path);

This function returns old operation path.

4 ClassOfBasicSRPInterface

Class define: **class ClassOfBasicSRPInterface**

4.1 Get OS type

4.1.1 get os type-GetOsType

VS_UINT32 SRPAPI **GetOsType**( )

4.2 Print function

4.2.1 Print

void **Print**(VS_CHAR * format,…);
void **PrintVar** (VS_CHAR * format ,va_list argList);

4.2.2 PrintLua

void **PrintLua**(VS_CHAR * format,…); print lua information
void **PrintLuaVar**(VS_CHAR * format ,va_list argList); print lua information

4.2.3 set print info to lua-SetPrintToLua

void SRPAPI **SetPrintToLua**(VS_BOOL PrintFlag);
If PrintFlag ==true, then output is redirected to lua window,==false, the output is not redirected to lua window

4.2.4 MessageBox function-MessageBox

void SRPAPI **MessageBox**(VS_CHAR *Caption, VS_CHAR *format,...);
void SRPAPI **MessageBoxVar**(VS_CHAR *Caption, VS_CHAR *format ,va_list argList);

4.2.5 print error information-ProcessError

void **ProcessError**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format,…);
void **ProcessErrorVar**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format ,va_list argList);
void **ProcessLuaError**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format,…);

void **ProcessLuaErrorVar**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format ,va_list argList);

4.2.6 Lua display information capture and release -CaptureLuaDisp/ ReleaseLuaDisp

void SRPAPI **CaptureLuaDisp**(VS_LuaInfoDispProc DispProc,VS_ULONG Para);
void SRPAPI **ReleaseLuaDisp**(VS_LuaInfoDispProc DispProc,VS_ULONG Para);

typedef void (SRPAPI *VS_LuaInfoDispProc)(VS_CHAR *DispInfo,VS_ULONG Para);

4.3 Service default path

4.3.1 set default path -SetDefaultPath

void SRPAPI **SetDefaultPath**(VS_CHAR *DefaultPath); set default path,if DefaultPath is NULL,then service path is set to default value.

4.3.2 get default path -GetDefaultPath

void SRPAPI **GetDefaultPath**(VS_CHAR *DefaultPath,VS_INT32 BufSize);

4.3.3 determine whether default path is set -DefaultPathIsSet

VS_BOOL SRPAPI **DefaultPathIsSet**();

4.4 Get parapkg interface

4.4.1 get parapkg interface-GetParaPkgInterface

class ClassOfSRPParaPackageInterface *__GetParaPkgInterface__();

4.5 Client connect to server(Connect, DisConnect)

VS_ULONG **Connect**(VS_CHAR *ServerInterface,VS_CHAR *ServerName, VS_UINT16 ServerPortNumber,VS_INT32 RetrySecond, class ClassOfSRPParaPackageInterface * ParaPkg, VS_ClientConnectCallBackProc ClientConnectCallBack,VS_ULONG Para ,VS_CHAR *LoginName,VS_CHAR *LoginPassword);
VS_ULONG **ConnectEx**(VS_CHAR *ServiceName,VS_INT32 RetrySecond, class ClassOfSRPParaPackageInterface * ParaPkg, VS_ClientConnectCallBackProc ClientConnectCallBack,VS_ULONG Para ,VS_CHAR *LoginName,VS_CHAR *LoginPassword);

Valid at client service group,ParaPkg and ClientConnectCallBack may be set to NULL
ServerInterface is interface of link-layer, may be NULL
The format is as :
“Host=interface address;if=share lib file name(include extension);site=share lib download address, ftp/http address ;para=parameter string”
for example:" host=192.168.0.1;if=SRPTcpLinkInterface1.dll;site=ftp://127.0.0.1"

class ClassOfSRPInterface *SRPAPI **Connect2**(VS_CHAR *ServerInterface,VS_CHAR *ServerName,VS_UINT16 ServerPortNumber,class ClassOfSRPParaPackageInterface *ParaPkg,VS_CHAR *LoginName,VS_CHAR *LoginPassword , VS_CHAR *SysRootItemName);

class ClassOfSRPInterface *SRPAPI **ConnectEx2**(VS_CHAR *ServiceName,class ClassOfSRPParaPackageInterface *ParaPkg,VS_CHAR *LoginName,VS_CHAR *LoginPassword , VS_CHAR *SysRootItemName);

SysRootItemName may be NULL, otherwise will wait service item to be synchronized.

4.5.1 sync connect-SConnect,valid at client service group

VS_ULONG SRPAPI **SConnect**(VS_CHAR *ServerInterface ,VS_CHAR *ServerName,VS_UINT16 ServerPortNumber, class ClassOfSRPParaPackageInterface *ParaPkg ,VS_CHAR *LoginName,VS_CHAR *LoginPassword);
VS_ULONG SRPAPI **SConnectEx**(VS_CHAR *ServiceName,class ClassOfSRPParaPackageInterface *ParaPkg ,VS_CHAR *LoginName,VS_CHAR *LoginPassword);

Returns 0 for false
The function has been waiting for clients to connect, and service completes its initialization procedure before returning.
ServerInterface is interface of link-layer, may be NULL
The format is as :
“Host=interface address;if=share lib file name(include extension);site=share lib download address, ftp/http address ;para=parameter string”
for example:" host=192.168.0.1;if=SRPTcpLinkInterface1.dll;site=ftp://127.0.0.1"

void **DisConnectEx**(VS_ULONG ConnectionID);
void **DisConnect**();

VS_BOOL SRPAPI IsConnect(); //-- true, client has connected to server; false indicates that no connection to the server.

4.6 Lua script function

4.6.1 GetLua

void ***GetLua**( );

4.6.2 DoBuffer

VS_BOOL SRPAPI **DoBuffer**(VS_CHAR *ScriptInterface, VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize,VS_BOOL IsUTF8 , VS_CHAR * ModuleName);
void SRPAPI **PostDoBuffer**(VS_CHAR *ScriptInterface, VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize,VS_BOOL IsUTF8 , VS_CHAR * ModuleName);
void SRPAPI **PostDoBufferEx**(VS_CHAR *ScriptInterface, VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize,VS_LuaInfoDispProc DispProc,VS_ULONG Para,VS_BOOL IsUTF8 , VS_CHAR *Name);
PostDoBuffer: execute in the platform message loop.
If ScriptInterface is NULL, default is lua. The interface may be lua,python, or other online script language registered.
ScriptInterfaces length < 16
ModuleName is name of the module,may be NULL.

4.6.3 LuaToBool

VS_BOOL **LuaToBool**(VS_INT32 Index);

4.6.4 LuaToString

VS_CHAR ***LuaToString**(VS_INT32 Index);

4.6.5 LuaToNumber

double **LuaToNumber**(VS_INT32 Index);

4.6.6 LuaToInt

VS_INT32 **LuaToInt**(VS_INT32 Index);

4.6.7 LuaIsBool

VS_BOOL **LuaIsBool**(VS_INT32 Index);

4.6.8 LuaIsString

VS_BOOL **LuaIsString**(VS_INT32 Index);

4.6.9 LuaIsNumber

VS_BOOL **LuaIsNumber**(VS_INT32 Index);

4.6.10 LuaIsInt

VS_BOOL **LuaIsInt**(VS_INT32 Index);
If returns VS_TRUE, then LuaIsNumber returns VS_TRUE too.

4.6.11 LuaGetGlobal

void **LuaGetGlobal** (VS_CHAR *Name);

4.6.12 LuaGetSrvGroupTable

void SRPAPI **LuaGetSrvGroupTable**(VS_CHAR *Name);
Get variables, supported formats: AAA.BBB.CCC etc., in which AAA is lua variable name of SrvGroup which is obtained by script command _GetSrvGroup ().

4.6.13 LuaPop

void **LuaPop**(VS_INT32 Index);

4.6.14 LuaPushString/ LuaPushLString

void **LuaPushString**(VS_CHAR *Value);
void SRPAPI **LuaPushLString**(VS_CHAR *Value, VS_ULONG Len);

4.6.15 LuaPushNumber

void **LuaPushNumber**(double Value);

4.6.16 LuaPushInt

void **LuaPushInt**(VS_INT32 Value);

4.6.17 LuaPushNil

void **LuaPushNil**( );

4.6.18 LuaSetGlobal

void **LuaSetGlobal** (VS_CHAR *Name);

4.6.19 LuaSetSrvGroupTable

void **LuaSetSrvGroupTable**(VS_CHAR *Name);

4.6.20 LuaIsNil

VS_BOOL **LuaIsNil**(VS_INT32 Index);

4.6.21 LuaIsTable

VS_BOOL **LuaIsTable**(VS_INT32 Index);

4.6.22 LuaNewTable

void **LuaNewTable**( );

4.6.23 LuaGetTop

VS_INT32 **LuaGetTop**( );

4.6.24 get lua object length-LuaObjLen

VS_INT32 SRPAPI LuaObjLen(VS_INT32 TableIndex);

4.7 Script function hook

void SRPAPI **LuaRegHook**(void *FuncAddr);
void SRPAPI **LuaUnRegHook**(void *FuncAddr);

FuncAddr is a Lua function, which format is VS_INT32 FuncAddr (void *L);
If the hook function processes the script, then should return VS_TRUE as the first result (uses LuaPushBool). In this case, CLE does not continue to call other hook functions. Otherwise, CLE continues to search other hook functions.
Values in the calling Lua stack is in turn:Object,ServiceGroupID, ScriptName,nOutArgs(return parameter number,<0 means any number), Para1,Para2…

4.8 GC collect hook

VS_BOOL SRPAPI **RegGCProc**(VS_GCProc GCProc, VS_ULONG Para);
void SRPAPI **UnRegGCProc**(VS_GCProc GCProc, VS_ULONG Para);

Prototype of VS_GCProc is :
typedef void (SRPAPI *VS_GCProc)(VS_ULONG Para);

In the callback function, should be a full garbage collection
Callback function called in the following cases:
1. service is unloaded
2. service is deactivated
3. service group is released.

4.9 System object and its event

4.9.1 get event request buffer -GetRequestBuf

When generate event, you should alloc request buffer using the function, or else it will have unintended consequences.
VS_EVENTPARAM_RUNPARAM * GetRequestBuf ();

4.9.2 release event response buffer -FreeResponseBuf

void FreeResponseBuf(VS_EVENTPARAM_RUNPARAM *ResponseParam);

4.9.3 release event request buffer -FreeRequestBuf

void FreeRequestBuf (VS_EVENTPARAM_RUNPARAM * RequestParam);

4.9.4 ProcessSysObjectEvent

VS_EVENTPARAM_RUNPARAM *SRPAPI ProcessSysObjectEvent(VS_UUID *EventID,VS_EVENTPARAM_RUNPARAM *RequestParam);

EventID : ID of the event.

VSSYSOBJ_WNDADJUST,VSSYSOBJ_WNDRESIZE,VSSYSOBJ_WNDCANBERESIZE,VSSYSOBJ_EDITSELECT

4.9.5 get system object-GetSysObject

void *SRPAPI GetSysObject();

4.10 Object function

4.10.1 get object name -GetName

VS_CHAR * **GetName** (VS_UUID *ObjectID);

4.11 get service group ID

VS_ULONG SRPAPI **GetServiceGroupID**( );

4.12 Service query function

4.12.1 query first service -QueryFirstService

VS_CHAR ***QueryFirstService**(VS_UUID &RetUuid);

4.12.2 query next service -QueryNextService

VS_CHAR ***QueryNextService**(VS_UUID &RetUuid);

4.12.3 query active service-QueryActiveService

VS_CHAR * **QueryActiveService** (VS_UUID &RetUuid);

4.13 Service management function

These functions should only be called at the server service group.

4.13.1 Import service -ImportServiceEx

VS_BOOL SRPAPI **ImportServiceEx**(VS_UUID *ServiceID, VS_BOOL LoadRunModule);
Import a service,ServiceID is ID of service. Server should not load or create any service before the function is called. ServiceID should be registered at registry under key SOFTWARE\SRPLab\SRPServer.
If LoadRunModule=true, the function will load bin module .DLL/.so, otherwise not load.

4.13.2 Import service -ImportServiceWithPath

VS_BOOL SRPAPI **ImportServiceWithPath** (VS_CHAR *ServicePath, VS_CHAR *ServiceName, VS_BOOL LoadRunModule);
Import a service,ServiceID is ID of service. Server should not load or create any service before the function is called. ServiceID should be registered at registry under key SOFTWARE\SRPLab\SRPServer.
If LoadRunModule=true,the function will load bin module .DLL/.so, otherwise not load
ServicePath may be local path or network path, such as:http://www.XXX.com. If it equals to “http://srplab”, then service is located at default srplab website.
If service name contains “.”, then it is dynamic service, the function is same as ImportDynaService.
If service name starts with char ”@”, then the service file locates at local disk.

4.13.3 Import service -ImportService

VS_BOOL SRPAPI **ImportService**(VS_CHAR *ServiceName, VS_BOOL LoadRunModule);
Import a service,ServiceID is UUID of service. Server should not load or create any service before the function is called. ServiceID should be registered at registry under key SOFTWARE\SRPLab\SRPServer.
If LoadRunModule=true,the function will load bin module .DLL/.so, otherwise not load
If service name contains ".", then it is dynamic service, the function is same as ImportDynaService.
If service name starts with char "@", then the service file locates at local disk.

4.13.4 Import service From Xml String - ImportServiceFromXmlBuf

VS_BOOL SRPAPI **ImportServiceFromXmlBuf**(VS_CHAR *Buf, VS_BOOL LoadRunModule);
Import service from xml string. Server should not load or create any service before the function is called.

4.13.5 Import dynamic service -ImportDynaService

VS_CHAR ***ImportDynaService**(VS_CHAR *Url)
Import dynamic service, may be local file or network file. Return value is service name.
If Url starts with char "@", then the service file locates at local disk.

4.13.6 Create service -CreateService/ CreateServiceEx

VS_BOOL SRPAPI **CreateService/CreateServiceEx**(VS_CHAR *ServicePath,VS_CHAR *ServiceName,VS_UUID *ServiceID, VS_CHAR *RootPass,VS_INT32 FrameInterval,VS_INT32 NetPkgSize,VS_INT32 UploadPkgSize,VS_INT32 DownloadPkgSize ,VS_INT32 DataUpPkgSize,VS_INT32 DataDownPkgSize);

Server should not load or create any service before the function is called. RootPass is password of root user. FrameInterval is interval(10ms) between frame of the service. NetPkgSize is size of network package, which default is 10240(bytes); UploadPkgSize is size of data uploaded per frame by client, unit is byte and default is 2048;DownloadPkgSize is size of data downloaded per frame by client, unit is byte, default is 2048. Range of FrameIntervale is [2,100], NetPkgSize is [1024,100*1024]; UploadPkgSize/ DataUpPkgSize is [1024,100*1024]; DownloadPkgSize/DataDownPkgSize is [1024,100*1024].
DataUpPkgSize, DataDownPkgSize is valid only for independent data server.

ServicePath: is path of new service, may be ““, in this case, default path is used..
Suppose the new created service will be saved at directory:
C:AAA\BBB\service name\service files, then ServicePath is set to "C:AAA\BBB"

CreateServiceEx: if BIN type files exist in the service path, then they are not deleted. Service should use ClearStatic function to clear static data.
CreateService: if BIN type files exist in the service path, then they will be deleted.

4.13.7 Load service -LoadServiceEx

VS_BOOL SRPAPI **LoadServiceEx**(VS_UUID vServiceID , VS_CHAR *UserName, VS_CHAR *UserPass , VS_BOOL LoadRunModule);
Load a service. Server should not load or create any service before the function is called. ServiceID should be registered in registry under key SOFTWARE\SRPLab\SRPServer.

4.13.8 Load service -LoadServiceWithPath

VS_BOOL SRPAPI **LoadServiceWithPath** (VS_CHAR *ServicePath, VS_CHAR *ServiceName , VS_CHAR *UserName, VS_CHAR *UserPass , VS_BOOL LoadRunModule);
Load a service. Server should not load or create any service before the function is called. ServiceID should be registered in registry under key SOFTWARE\SRPLab\SRPServer.

4.13.9 Load service -LoadService

VS_BOOL SRPAPI LoadService(VS_CHAR *ServiceName , VS_CHAR *UserName, VS_CHAR *UserPass , VS_BOOL LoadRunModule);
Load a service. Server should not load or create any service before the function is called. ServiceID should be registered in registry under key SOFTWARE\SRPLab\SRPServer.

4.13.10 export service header(.h) -ExportServiceHeader

VS_BOOL SRPAPI **ExportServiceHeader**(VS_CHAR *ServiceName, VS_CHAR *Path);

4.13.11 export service define(.h) -ExportServiceDefine

VS_BOOL SRPAPI **ExportServiceDefine** (VS_CHAR *ServiceName, VS_CHAR *FileName);

4.13.12 clear service -ClearService

void SRPAPI **ClearService**( );
For server service group, service will be unloaded.
For client service group, connection will be closed.

4.13.13 clear service -ClearServiceEx

void SRPAPI **ClearServiceEx**( );
For server service group, service will be unloaded.
For client service group, connection will be closed.
The function will process all service groups existed.

4.13.14 clear Lua global -ClearLuaGlobal

void SRPAPI ClearLuaGlobal( )

4.14 service register and alloc Cooperator

4.14.1 register service -RegisterServer

VS_BOOL SRPAPI RegisterServer(VS_CHAR *ServiceName);

4.14.2 alloc or free Cooperator -AllocCooperator/FreeCooperator

void SRPAPI AllocCooperator(VS_CHAR *ServiceName);
void SRPAPI FreeCooperator(VS_CHAR *ServiceName);

4.14.3 get server information -GetServerUrlInfo

void SRPAPI GetServerUrlInfo(struct StructOfVSServerUrlInfo *ServerUrlInfo)
Called at server service group to get parameters for client to connect to server, which are set through config file, includes DirectClientInterface, m_Host, ServerPortNumber.

4.15 Output content refresh

4.15.1 WebService object refresh -WebServiceRefresh

void SRPAPI WebServiceRefresh();
The function should be called when the output object is changed, such as created, deleted, attribute changed or object name changed.

4.16 Webservice interface.

4.16.1 get wsdl-GetWSDL

VS_BOOL SRPAPI GetWSDL(VS_ULONG WSDLVersion, VS_CHAR * WebServiceHost,VS_UINT16 PortNumber,class ClassOfSRPBinBufInterface *BinBuf);

WSDLVersion:reserved,current support 1.1

Host: If equals to NULL, then uses default or configuration host

map type:

If object sets its WebServiceFlag to true, then it acts as a PortType service mapped to WSDL.

The functions defined in object acts as Operation.

WebServiceHost, which format is ip address or url:port.

4.17 Get service interface-SRPInterface

4.17.1 Get service interface by ID -GetSRPInterfaceEx

class ClassOfSRPInterface *SRPAPI GetSRPInterfaceEx(VS_UUID *ServiceID,, VS_CHAR *UserName, VS_CHAR *UserPass);

UserPass is the password of the user

At client or debug,UserName and UserPass may be NULL

If ServiceID equals NULL, then the function returns the active service interface of current service group.

4.17.2 get service interface by name -GetSRPInterface

class ClassOfSRPInterface *SRPAPI GetSRPInterface(VS_CHAR *ServiceName, VS_CHAR *UserName, VS_CHAR *UserPass);

UserPass is the password of the user

At client or debug,UserName and UserPass may be NULL

If ServiceName is NULL, then the function returns the active service interface of current service group.

4.17.3 Get service interface by ID-GetSRPInterfaceEx2

class ClassOfSRPInterface *SRPAPI GetSRPInterfaceEx2(VS_UUID *ServiceID , VS_GetUserInfoCallBackProc CallBackProc);

The prototype of callback function refers to 5.1

At client and debug side,CallBackProc may be NULL

If ServiceID equals NULL, then the function returns the active service interface of current service group.

4.17.4 get service interface by name -GetSRPInterface2

class ClassOfSRPInterface *SRPAPI GetSRPInterface2(VS_CHAR *ServiceName , VS_GetUserInfoCallBackProc CallBackProc);

The prototype of callback function refers to 5.1

At client and debug side,CallBackProc may be NULL

If ServiceName is NULL, then the function returns the active service interface of current service group.

4.18 depended service function

4.18.1 change depended service-ChangeDepend

VS_BOOL SRPAPI **ChangeDepend** (VS_UUID *OldDependServiceID, VS_UUID *NewDependServiceID,VS_CHAR *NewServiceName);
The function is used to change ID of depended service. When the change is finish, if ID is changed, then current service will be saved and unloaded, and re-loaded. The function should be called at server side**. The function can not change dynamic depended services, in this case, application should use ImportService function.**

4.18.2 add depended service-AddDepend

VS_BOOL SRPAPI **AddDepend** (VS_UUID *DependServiceID,VS_CHAR *NewServiceName);
Add a depended service. The function should be called at server side**. The function can not change dynamic service, in this case, should use ImportService function.**

4.19 Encryption functions

4.19.1 MD5 encryption -GetMD5

VS_INT8 *SRPAPI **GetMD5**(VS_INT8 *Buf,VS_INT32 BufSize);

4.20 Get current Ticket

4.20.1 get current Ticket-GetTickCount

VS_ULONG SRPAPI GetTickCount(); unit is ms
If high precision is supported, then it will return high precision timer.

4.21 Get object ID,and callback function when the ID is changed

void SRPAPI **GetID**(void *Object,VS_UUID *UuidPtr);/ VS_UUID *SRPAPI GetIDEx(void *Object)

VS_BOOL **RegObjectIDChangeNotify**(VS_ObjectIDChangeNotifyProc ChangeNotifyProc,VS_ULONG Para);
void **UnRegObjectIDChangeNotify**(VS_ObjectIDChangeNotifyProc ChangeNotifyProc,VS_ULONG Para);
The prototype of function is:
typedef void (SRPAPI *VS_ObjectIDChangeNotifyProc)(void *Object,VS_ULONG Para,VS_UUID *NewObjectID);

4.22 Object free callback function

VS_BOOL SRPAPI **RegObjectFreeNotify**(VS_ObjectFreeNotifyProc FreeNotifyProc,VS_ULONG Para);
void SRPAPI **UnRegObjectFreeNotify**(VS_ObjectFreeNotifyProc FreeNotifyProc,VS_ULONG Para);
The prototype of function is:
typedef void (SRPAPI *VS_ObjectFreeNotifyProc)(void *Object,VS_ULONG Para);

4.23 Registry function

4.23.1 get string from registry-GetRegStr

VS_CHAR *SRPAPI GetRegStr(VS_CHAR *SubKey, VS_CHAR *ValueName, VS_CHAR *DefaultValue);
The format of subkey is as “Software\\SRPLab\\SRPServer"

When the SubKey is not exist, DefaultValue is returned.

4.23.2 get integer from reistry-GetRegInt

VS_ULONG SRPAPI GetRegInt(VS_CHAR *SubKey, VS_CHAR *ValueName,VS_ULONG DefaultValue);
The format of subkey is as “Software\\SRPLab\\SRPServer"
When the SubKey is not exist, DefaultValue is returned.

4.23.3 setup timer-SetupTimer

VS_ULONG **SetupTimer**(VS_INT32 Ticket, VS_TimerProc FunctionAddr, VS_ULONG Para1,VS_ULONG Para2,VS_ULONG Para3,VS_ULONG Para4);
Returns 0 indicates failure, otherwise the function returns TimerID. The unit of Ticket is 10ms
typedef void (SRPAPI *VS_TimerProc)(void *Object, VS_ULONG TimerID, VS_ULONG Para1,VS_ULONG Para2,VS_ULONG Para3,VS_ULONG Para4);
In callback function,Object is set to NULL.
If TimerID = 0xFFFFFFFF, indicates timer is cleared.

4.23.4 kill the timer-KillTimer

void **KillTimer**(VS_ULONG TimerID);

4.24 RawSocket function

The function provides socket connection based on CLE, which may be used to interconnect between cle programs.
The maximum length of one package, should less than 32767 bytes for unreliable transmit.
RawSocket does not retry if the connection fails. If error occurs, application should re-setup the connection.

4.24.1 setup Socket server-SetupSocketServer

VS_ULONG SRPAPI **SetupSocketServer**(VS_CHAR *Interface, VS_UINT16 PortNumber, VS_ULONG *LinkInterfaceStatus,VS_SockAcceptCallBackProc CallBackProc,VS_ULONG Para);
Returns 0 indicates failure, or else returns server connection ID, which can be used to close the connection.
Interface: is link-layer interface, may be NULL.
The prototype of the callback:
typedef void (SRPAPI *VS_SockAcceptCallBackProc)(VS_ULONG uMes,VS_ULONG ConnectionID, SOCKADDR_IN SockAddr, VS_ULONG Para, VS_ULONG MachineID, VS_SockEventCallBackProc *ClientCallBack, VS_ULONG *ClientPara);
If a client connection request is received, the callback is called, which returns the function address to process the client connection. If the callback returns NULL, then the client is denied to connect.
MachineID: is ID of the new client machine.

Interface is interface of link-layer, may be NULL
The format is as :
“Host=interface address;if=share lib file name(include extension);site=share lib download address, ftp/http address ;para=parameter string”
for example:" host=192.168.0.1;if=SRPTcpLinkInterface1.dll;site=ftp://127.0.0.1"

4.24.2 setup Socket client-SetupSocketClient

VS_ULONG SRPAPI **SetupSocketClient**(VS_CHAR *ServerInterface, VS_CHAR *ServerName, VS_UINT16 PortNumber, VS_SockEventCallBackProc CallBackProc,VS_ULONG Para);
Returns 0 for failure, otherwise returns the request ID, which can be used to close the connection, but it will be invalid after callback function is triggered.
Interface is interface of link-layer, may be NULL
The prototype of the callback:
typedef void (SRPAPI *VS_SockEventCallBackProc)(void *Machine, VS_ULONG uMsg, VS_ULONG MachineID, VS_ULONG LinkInterfaceStatus,void *Mes, VS_INT32 MesLength, VS_ULONG Para);
MachineID is the ID of the connection.

4.24.3 close Socket connection-CloseSocketConnect

void SRPAPI **CloseSocketConnect**(VS_ULONG ConnectionID/MachineID);
If server connection is closed, then all clients on the connection are closed.

4.24.4 send data-SocketSend

VS_BOOL SRPAPI **SocketSend**(VS_ULONG MachineID, class ClassOfSRPParaPackageInterface *ParaPkg , VS_BOOL Assure); Assure == true , retransmission will be done by CLE. ==false, does not retransmit

4.24.5 send binary data-SocketSendBin

VS_BOOL SRPAPI **SocketSendBin**(VS_ULONG MachineID , VS_INT32 BinDataSize, VS_INT8 *BinDataBuf , VS_BOOL Assure); Assure == true , retransmission will be done by CLE. ==false, does not retransmit

4.24.6 setup timer-SetupSocketTimer

VS_ULONG SRPAPI **SetupSocketTimer**(VS_ULONG MachineID, VS_INT32 Ticket,VS_INT32 Counter);
Returns 0 for failure, otherwise returns the timer ID. The unit of Ticket is 10ms. If Counter equals to 0, then the timer is generated periodicly, or else is the times of timer to be generated

4.24.7 kill timer-KillSocketTimer

void SRPAPI **KillSocketTimer**(VS_ULONG MachineID, VS_ULONG TimerIndex);

4.25 Change data server address

4.25.1 set data server address-SetDataServerAddr

VS_BOOL SRPAPI **SetDataServerAddr**(VS_BOOL DirectConnect,VS_CHAR *DataServerInterface,VS_CHAR *DataServerName,VS_UINT16 DataServerPort,VS_CHAR *LocalDataServerInterface,VS_UINT16 LocalDataServerPort);
DirectConnect:==true , if superior data server port number is valid, client will connect to data server directly.Otherwise is relayed by local server.
DataServerInterface: Interface of superior data server, may be NULL.
DataServerName: superior data server name, ip address or url.
DataServerPort: superior data server port number

LocalDataServerInterface: Interface of local data server, may be NULL.

LocalDataServerPort: local data server port. If equals to 0, means invalid

The function is valid at server.

4.25.2 Register static data query callback -RegQueryStaticDataProc

void SRPAPI **RegQueryStaticDataProc** (VS_QueryObjectStaticDataProc Proc,VS_ULONG Para);
void SRPAPI **UnRegQueryStaticDataProc**(VS_QueryObjectStaticDataProc Proc,VS_ULONG Para)
prototype of the function:
typedef void (SRPAPI *VS_QueryObjectStaticDataProc)(VS_UUID *ObjectID,VS_ULONG UniqueDataUnitID, VS_STATICID *DataVersion,VS_ULONG *DataSize, VS_UINT8 *StaticSaveFlag, VS_BOOL AutoDownLoad, VS_BOOL *ContinueDefaultProcess , VS_STATICID *LocalDataVersion, VS_CHAR *Token, VS_BOOL *CachedToDiskFlag,VS_ULONG Para);
If ContinueDefaultProcess returns true,then default process is used, and the pointer returned is ignored.
If returns false, no longer executing default process.
StaticSaveFlag: static data save flag, which is returned from application .
Token: is token of the static data, such as its filename.The token may not exist, in this case, the parameter should be set to NULL.
LocalDataVersion: If data does not exist at local, then it is invalid, or else is the local data version.
CachedToDiskFlag: If returns VS_TRUE, then CLE cache the data,or else not cache and this function will be called for each query.
In the function, application should use SetStaticData to set the object static data.

4.25.3 Register static data save function-RegSaveStaticDataProc/ UnRegSaveStaticDataProc

void SRPAPI **RegSaveStaticDataProc**(VS_SaveObjectStaticDataProc Proc,VS_ULONG Para);
void SRPAPI **UnRegSaveStaticDataProc**(VS_SaveObjectStaticDataProc Proc,VS_ULONG Para);
function prototype:
typedef void (SRPAPI *VS_SaveObjectStaticDataProc)(VS_UUID *ObjectID,VS_ULONG UniqueDataUnitID, VS_STATICID Version,VS_ULONG DataSize,VS_INT8 *DataBuf, VS_UINT8 StaticSaveFlag, VS_BOOL *ContinueDefaultProcess, VS_BOOL *CachedToDiskFlag,VS_ULONG Para);
If ContinueDefaultProcess returns true,then default process is used.
If returns false, no longer executing default process.
StaticSaveFlag : is static data save flag, which is returned from application .
CachedToDiskFlag: If returns VS_TRUE, then cle cache the data,or else not cache. and this function is called for each query.

4.25.4 register static data clear function-RegClearStaticDataProc/ UnRegClearStaticDataProc

void SRPAPI **RegClearStaticDataProc**(VS_ClearObjectStaticDataProc Proc,VS_ULONG Para);
void SRPAPI **UnRegClearStaticDataProc**(VS_ClearObjectStaticDataProc Proc,VS_ULONG Para);
Function prototype:
void (SRPAPI *VS_ClearObjectStaticDataProc)(VS_UUID *ObjectID,VS_ULONG UniqueDataUnitID,VS_BOOL *ContinueDefaultProcess);
If ContinueDefaultProcess returns true, then default process is continued.
If UniqueDataUnitID equals to 0, then static data of all objects are cleared.

4.25.5 set server parameter-SetServerPara

void SRPAPI **SetServerPara**(VS_INT32 MaxClientNumber,VS_INT32 MaxDataServerConnectionNumber,VS_INT32 DataServerOverTime);

Valid at server
MaxDataServerConnectionNumber : Number of connection permitted
DataServerOverTime: Data connection timeout (s)

4.26 File callback function

4.26.1 register callback-RegFileCallBack

VS_BOOL **RegFileCallBack**(VS_UUID *ServiceID,VS_FileUpDownLoadCallBackProc CallBackProc , VS_ULONG Para);
void **UnRegFileCallBack**(VS_UUID *ServiceID,VS_FileUpDownLoadCallBackProc CallBackProc , VS_ULONG Para);
The function is valid at client or server which is called when the download or upload happens.
At server side, the function is called for http download.
For http download,ServiceID may be NULL.

VS_BOOL SRPAPI **RegFileReqCallBack**(VS_FileUpDownLoadRequestCallBackProc FileCallBackProc, VS_ULONG Para);
void SRPAPI **UnRegFileReqCallBack**(VS_FileUpDownLoadRequestCallBackProc FileCallBackProc, VS_ULONG Para);
Register at server to handle client request,which prototype is:
typedef VS_BOOL (SRPAPI *VS_FileUpDownLoadRequestCallBackProc)(VS_ULONG ClientID, VS_ULONG ClientPrivateTag, VS_ULONG Para, VS_ULONG uMsg, VS_UPDOWNFILEMSG *Msg);
If returns VS_FALSE, then the action is not permitted

4.27 Get statistic information

4.27.1 query statistic information-QuyeryStatisticInfo

void SRPAPI **QuyeryStatisticInfo**(void *Machine,VS_STATISTICINFO *InfoBuf)
Valid at server and client. When the function is called at server with Machine set to NULL, then it returns statistic information of all clients or else get statistic information of corresponding client. At client side, Machine should be always set to NULL.

4.27.2 get file upload or download infomation-GetFileInfo

void **GetFileInfo**(VS_UPDOWNFILEINFO *InfoPtr);
valid at client.

4.28 compression and decompression functions(uses zlib)

4.28.1 Compress

VS_BOOL **Compress**(VS_UINT8 *dest,VS_ULONG *destLen,VS_UINT8 *source,VS_ULONG sourceLen);
destLen is destination buffer size

4.28.2 UnCompress

VS_BOOL **UnCompress**(VS_UINT8 *dest,VS_ULONG *destLen,VS_UINT8 *source,VS_ULONG sourceLen);
destLen is destination buffer size

4.29 String conversion functions

4.29.1 convert string to UUID-StringToUuid

VS_BOOL SRPAPI **StringToUuid**(VS_INT8 *String,VS_UUID *Uuid);

4.29.2 Convert UUID to string-UuidToString

VS_INT8 *SRPAPI **UuidToString**(VS_UUID *Uuid);

4.29.3 convert string to Utf8-StringToUtf8

VS_INT8 *SRPAPI StringToUtf8(VS_INT8 *String)
If returns NULL, then the buffer should be freed use interface function Free.

4.29.4 convert Utf8 to string-Utf8ToString

VS_INT8 *SRPAPI Utf8ToString(VS_INT8 *String)
If returns NULL, then the buffer should be freed use interface function Free.

4.30 Miscellaneous functions

4.30.1 get program type-GetProgramType

VS_UINT16 GetProgramType( );

4.30.2 determine whether server is the default server -IsDefaultServer

VS_BOOL **IsDefaultServer**();; //---Whether server is default server.

4.30.3 whether the manager window is visible-IsWindowVisible

VS_BOOL IsWindowVisible();

4.30.4 hide manager window-HideWindow

void **HideWindow**();

4.30.5 show manager window-ShowWindow

void **ShowWindow**();

4.30.6 set manager window caption-SetWindowCaption

void **SetWindowCaption**(VS_CHAR *Caption);
When create or load service, application's caption will be set automicly. Therefore the function should be called after service is created or loaded.

4.30.7 exit program-ExitVSSystem

void **ExitVSSystem**(VS_CHAR *ErrorInfo);
ErrorInfo may be set to NULL

4.30.8 whether the application is active-IsAppActive

VS_BOOL **IsAppActive**();

4.30.9 running when no message - SetIdleActive

void **SetIdleActive**(VS_BOOL true/false);

4.30.10 get version-GetVersion

//----get platform version
void **GetVersion**(VS_UINT8 *MainVersion,VS_UINT8 *SubVersion,VS_UINT16 *BuildVersion);

4.30.11 get platform version -GetVersionInfo

void SRPAPI **GetVersionInfo**(VS_CHAR *InfoBuf,VS_INT32 InfoBufSize);

4.30.12 get manager window handle-GetWindowHandle

VS_HWNDGetWindowHandle( );

4.30.13 get manager window size-GetWindowSize

void **GetWindowSize**(VS_INT32 *Width, VS_INT32 *Height);

4.30.14 set menu and staus bar-ShowStatusMenu

void SRPAPI **ShowStatusMenu**(VS_BOOL MenuShowFlag, VS_BOOL StatusShowFlag);
== true display; == false hide

4.30.15 set text color-SetColor

void SRPAPI **SetColor**(VS_COLORText, VS_COLORExplane, VS_COLORObjName, VS_COLOR, VS_COLORNumber, VS_COLORError);

4.30.16 set text background color-SetBkColor

void SRPAPI SetBkColor(VS_COLORBkColor);

4.30.17 set window parameter-SetWindowStyle

void SRPAPI SetWindowStyle(VSWINDOWSTYLE *Style);

```
typedef struct StructOfVSWindowStyle{
   VS_BOOL SystemMenuFlag;
   VS_BOOL MinimizeFlag;
   VS_BOOL MaximizeFlag;
   VS_BOOL ShowBorderFlag;
   VS_BOOL SizeableFlag;
}VSWINDOWSTYLE;
```

4.30.18 move window-MoveWindow

```
void SRPAPI MoveWindow(VSWINDOWPOSITION *Position,VS_BOOL RepaintFlag);
typedef struct StructOfVSWindowPosition{
   VS_INT32 X;
   VS_INT32 Y;
   VS_INT32 nWidth;
   VS_INT32 nHeight;
}VSWINDOWPOSITION;
```

If nWidth and nHeight equal to 0, then application should not change current window width and height.

4.30.19 get window position-GetWindowPos

```
void SRPAPI GetWindowPos(VSWINDOWPOSITION *Position);
```

4.30.20 set window status-SetWindowStatus

```
void SRPAPI SetWindowStatus(VS_INT32 Status);  //--0 normal; 1  minimize;  other value, maximize
```

4.31 Client window function(GetClientWndHandle,GetClientWndSize,SetClientWndSize,SetClientWndFocus,KillClientWndFocus)

```
VS_HWND GetClientWndHandle(  );
void GetClientWndSize( VS_INT32 *Width, VS_INT32 *Height );
void SetClientWndSize( VS_INT32 Width, VS_INT32 Height );
void SetClientWndFocus(VS_HWND hWnd, VS_BOOL NeedAction );
```

If hWnd is NULL, then uses the last hWnd. If NeedAction=VS_TRUE, then the focus should be set, or else is only notification

```
void KillClientWndFocus(VS_HWND hWnd, VS_BOOL NeedAction );  NeedAction=VS_TRUE, then the focus should be canceled, or else is only notification
void SRPAPI ClearClientWnd( )
    void SRPAPI HideClientWnd( );
    void SRPAPI ShowClientWnd( );
    void SRPAPI SetClientBkColor( VS_COLOR BkColor );
```

4.32 Message hook(SetMessageHook,GetMessageHook)

```
typedef VS_BOOL (*VS_SRPMessageProcessHookProc)( VS_HWND hWnd,VS_ULONG message,
VS_ULONG wParam, VS_ULONG lParam );
//---If the message has been processed, the function should return true; otherwise returns false.
void SetMessageHook(VS_SRPMessageProcessHookProc HookProc);
VS_SRPMessageProcessHookProc GetMessageHook( );
```

4.33 Get control interface(ClassOfSRPControlInterface)

4.33.1 get control interfaceGetSRPControlInterface

class ClassOfSRPControlInterface* **GetSRPControlInterface** ();
The returned interface should be released with its Release function.

4.34 Synchronization related functions (valid at client)

4.34.1 Service group is being synchronized-IsInSync

VS_BOOL SRPAPI **IsInSync**( ); //--- Returns true that synchronization is in progress, otherwise returns false.

4.34.2 whether service is synchronous-IsServiceSync,valid at client

VS_BOOL SRPAPI IsServiceSync();

4.34.3 wait for service synchronization -WaitServiceSync,valid at client

VS_BOOL SRPAPI **WaitServiceSync**(VS_INT32 WaitTimeMs);
The function returns until the client finish its sync process. WaitTimeMs is the max time to wait, which unit is ms.
If WaitTimeMs equal to 0, then wait forever.

4.35 Global flow control

4.35.1 create SRPIDLE event-SRPIdle

VS_BOOL SRPAPI **SRPIdle**();
Returns true indicates no longer need to generate IDLE events, or else continue to generate IDLE events.

4.35.2 create SRPAppActive event-SRPAppActive

void SRPAPI **SRPAppActive**();

4.35.3 create SRPAppDeactive event-SRPAppDeactive

void SRPAPI **SRPAppDeactive**();

4.36 Hyper connection

4.36.1 trigger hyper connection-HyperLink

void SRPAPI **HyperLink**(VS_CHAR *HyperLink,VS_BOOL CreateNewWindow);

4.36.2 trigger app event-AppEvent

void SRPAPI **AppEvent**(VS_ULONG EventID,VS_CHAR *EventInfo);

4.37 service parse interface

4.37.1 get service name-GetServiceName

VS_CHAR *SRPAPI **GetServiceName**(VS_UUID *ServiceID);
If returns NULL, then service is not found or loaded.

4.37.2 get service ID-GetServiceID

VS_BOOL SRPAPI GetServiceID(VS_CHAR *ServiceName,VS_UUID *ServiceID);

4.37.3 service parse interface-XmlToService

class ClassOfSRPInterface * SRPAPI XmlToService(class ClassOfSRPSXMLInterface *SXMLInterface,VS_CHAR *DataPath, VS_CHAR *SegmentName, SRPParse_PrintProc PrintProc);
The function is another format to crwate service. Its syntax refers to related documents.
typedef void (SRPAPI *SRPParse_PrintProc)(VS_ULONG Para, VS_CHAR *Info);

4.37.4 exception handler-SetExceptHandler

void SRPAPI **SetExceptHandler**(VS_ExceptHandlerProc ExceptHandlerProc);
VS_ExceptHandlerProc is defined as:
typedef void (SRPAPI *VS_ExceptHandlerProc)(VS_CHAR *ErrorInfo);
ErrorInfo is error information.
The function is called when exception occurs.

4.38 Index and search function

4.38.1 create an Indexer(CreateIndex_Nor,CreateIndexCmp_Nor ,CreateIDIndex_Nor,CreateIDIndexEx_Nor)

void ***CreateIndex_Nor**(VS_INT32 KeyNumber, VS_UINT16 HashTableBits); KeyNumber represents number of the key, which supports 1,2,3.
void *SRPAPI **CreateIndexCmp**_Nor(VS_INT32 KeyNumber, VS_UINT16 HashTableBits, VS_IndexCompareProc CompareProc);
void ***CreateIDIndex_Nor**(VS_UINT16 HashTableBits);
void ***CreateIDIndexEx_Nor**(VS_UINT16 HashTableBits); uses UUID + VS_ULONG as index

Prototype of CompareProc is :
typedef VS_INT32 (SRPAPI *VS_IndexCompareProc)(void *NodeBuf1,void *NodeBuf2); -1 Buf1 < Buf2, 0 Buf1 == Buf2, 1 Buf1 > Buf2. CreateIndexCmp does not support _F series searching functions.

HashTableBits represents Hash table size,2^(HashTableBits) number, which is used to speed up the search process. It may be set to 0. The parameter will occupy extra memory about 2^(HashTableBits)*4 bytes.

4.38.2 create a indexer(CreateIndex_Dbg,CreateIndexCmp_Dbg ,CreateIDIndex_Dbg,CreateIDIndexEx_Dbg)

void ***CreateIndex_Dbg***(VS_INT32 KeyNumber, VS_UINT16 HashTableBits, VS_CHAR *FileName,VS_INT32 LineNumber); KeyNumber represents number of the key, which supports 1,2,3.
void ***CreateIDIndex_Dbg*** (VS_UINT16 HashTableBits,VS_CHAR *FileName,VS_INT32 LineNumber);
void ***CreateIDIndexEx_Dbg*** (VS_UINT16 HashTableBits,VS_CHAR *FileName,VS_INT32 LineNumber);
uses UUID + VS_ULONG as index
void *SRPAPI **CreateIndexCmp**_Dbg(VS_INT32 KeyNumber, VS_UINT16 HashTableBits, VS_IndexCompareProc CompareProc, VS_CHAR *FileName,VS_INT32 LineNumber);

Prototype of CompareProc is :
typedef VS_INT32 (SRPAPI *VS_IndexCompareProc)(void *NodeBuf1,void *NodeBuf2); -1 Buf1 < Buf2, 0 Buf1 == Buf2, 1 Buf1 > Buf2. CreateIndexCmp_Dbg does not support _F series searching functions.

HashTableBits represents Hash table size,2^(HashTableBits) number, which is used to speed up the search process; it may be set to 0. The parameter will occupy extra memory about 2^(HashTableBits)*4 bytes.

4.38.3 Add,delete or find an index(one key)(InsertOneKey,FindOneKey,DelOneKey)

void **InsertOneKey**(void *IndexContext,VS_ULONG MainKey,VS_INT8 *Buf);
VS_INT8 ***FindOneKey**(void *IndexContext,VS_ULONG MainKey);
VS_INT8 ***DelOneKey**(void *IndexContext,VS_ULONG MainKey);
VS_INT8 ***QueryFirstOneKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); Small to large order
VS_INT8 ***QueryNextOneKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); Small to large order
VS_INT8 ***QueryFirstOneKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); large to small order
VS_INT8 ***QueryNextOneKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); large to small order
Context is Indexer, MainKey is the key, Buf is the buffer of the index points to.

4.38.4 Add,delete or find an index(two keywords)(InsertTwoKey,FindTwoKey,DelTwoKey)

void **InsertTwoKey**(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_INT8 *Buf);
VS_INT8 ***FindTwoKey**(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey);
VS_INT8 ***DelTwoKey**(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey);
VS_INT8 ***QueryFirstTwoKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ***QueryNextTwoKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ***QueryFirstTwoKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); large to small order
VS_INT8 ***QueryNextTwoKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); large to small order
VS_INT8 ***QueryFirstTwoKey_F**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ***QueryNextTwoKey_F** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ***QueryFirstTwoKeyA_F** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); large to small order

VS_INT8 ***QueryNextTwoKeyA_F*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); large to small order

4.38.5 Add,delete or find an index(three keywords)(InsertThreeKey,FindThreeKey,DelThreeKey)

void **InsertThreeKey**(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG ThirdKey,VS_INT8 *Buf);
VS_INT8 ***FindThreeKey***(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG ThirdKey);
VS_INT8 ***DelThreeKey***(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG ThirdKey);
VS_INT8 ***QueryFirstThreeKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryNextThreeKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryFirstThreeKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryNextThreeKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryFirstThreeKey_F***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryNextThreeKey_F*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryFirstThreeKeyA_F*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryNextThreeKeyA_F*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryFirstThreeKey_S***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryNextThreeKey_S*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryFirstThreeKeyA_S*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryNextThreeKeyA_S*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); large to small order

4.38.6 Add,delete or find an index(UUID keywords)(InsertIDKey,FindIDKey,DelIDKey,InsertIDKeyEx,FindIDKeyEx,DelIDKeyEx).

void **InsertIDKey**(void *IndexContext,VS_UUID *UuidKey,VS_INT8 *Buf);
VS_INT8 ***FindIDKey***(void *IndexContext,VS_UUID *UuidKey);
VS_INT8 ***DelIDKey***(void *IndexContext,VS_UUID *UuidKey);
VS_INT8 ***QueryFirstIDKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); Small to large order
VS_INT8 ***QueryNextIDKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); Small to large order
VS_INT8 ***QueryFirstIDKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); large to small order
VS_INT8 ***QueryNextIDKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); large to small order

void **InsertIDKeyEx**(void *IndexContext,VS_UUID *UuidKey,VS_ULONG ExKey,VS_INT8 *Buf);

VS_INT8 ***FindIDKeyEx***(void *IndexContext,VS_UUID *UuidKey,VS_ULONG ExKey);
VS_INT8 ***DelIDKeyEx***(void *IndexContext,VS_UUID *UuidKey,VS_ULONG ExKey);
VS_INT8 ***QueryFirstIDKeyEx***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order
VS_INT8 ***QueryNextIDKeyEx***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order
VS_INT8 ***QueryFirstIDKeyEx_F***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order,use special UuidKey value
VS_INT8 ***QueryNextIDKeyEx_F***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order,use special UuidKey value
VS_INT8 ***QueryFirstIDKeyExA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); large to small order
VS_INT8 ***QueryNextIDKeyExA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); large to small order

4.38.7 get index number-GetKeyNumber

VS_INT32 SRPAPI GetKeyNumber(void *IndexContext)

4.38.8 delete all index-DelAllKey

void **DelAllKey**(void *IndexContext);

4.38.9 delete indexer-DestoryIndex

void **DestoryIndex**(void *IndexContext);

4.38.10 get Hash value-GetHashValue

VS_ULONG SRPAPI GetHashValue(void *Key,VS_ULONG Length,VS_ULONG InitValue);

4.39 Memory manager function

4.39.1 create memory manager-CreateMemory_Nor

void ***CreateMemory_Nor***(VS_INT32 ItemSize);
ItemSize is size of each memory block

4.39.2 create memory manager -CreateMemory_Dbg

void ***CreateMemory_Dbg***(VS_INT32 ItemSize, VS_CHAR *FileName,VS_INT32 LineNumber);
ItemSize is size of each memory block

4.39.3 alloc memory block-GetMemoryPtr_Nor

void ***GetMemoryPtr_Nor***(void *MemoryContext);
MemoryContext is memory manager.

4.39.4 alloc memory block -GetMemoryPtr_Dbg

void ***GetMemoryPtr_Dbg**(void *MemoryContext, VS_CHAR *FileName,VS_INT32 LineNumber);
MemoryContext is memory manager.

4.39.5 query first memory block-QueryFirstMemoryPtr

void ***QueryFirstMemoryPtr**(void *MemoryContext,VS_QUERYRECORD *QueryRecord);

4.39.6 query next memory block -QueryNextMemoryPtr

void ***QueryNextMemoryPtr**(void *MemoryContext,VS_QUERYRECORD *QueryRecord);

4.39.7 free memory block-FreeMemoryPtr

void **FreeMemoryPtr**(void *MemoryContext,void *Ptr);

4.39.8 clear memory block-ClearMemory

void SRPAPI **ClearMemory**(void *MemoryContext);
Free all memory blocks in the memory manager.

4.39.9 delete memory manager-DestoryMemory

void **DestoryMemory**(void *MemoryContext);

4.39.10 alloc memory-Malloc_Nor

void ***Malloc_Nor**(VS_INT32 MemorySize);

4.39.11 alloc memory-Malloc_Dbg

void ***Malloc_Dbg**(VS_INT32 MemorySize VS_CHAR *FileName,VS_INT32 LineNumber);

4.39.12 free memory-Free

void **Free**(void *MemoryPtr);

4.39.13 get memory used-GetMemoryUsed

void SRPAPI **GetMemoryUsed**(VS_ULONG *KernelAllocSize,VS_ULONG *DataAllocSize,VS_ULONG *AppAllocSize, VS_ULONG *ScriptMemoryUsed);
KernelAllocSize: memory of core.
DataAllocSize: memory of static data
AppAllocSize: memory of application allocated using interface function Malloc.
ScriptMemoryUsed: memory of lua script.

4.40 State machine function

4.40.1 get machine ID-GetMachineID

VS_ULONG GetMachineID(void *Machine); MachineID is ClientID.

4.40.2 find machine by ID-FindMachine

void ***FindMachine**(VS_ULONG MachineID);

4.41 Other interface

4.41.1 Get Lock interface-GetSRPLockInterface

class ClassOfSRPLockInterface *SRPAPI **GetSRPLockInterface**(void);

4.41.2 convert MD5 string to UUID-MD5ToUuid, UuidToMD5

VS_BOOL SRPAPI **MD5ToUuid**(VS_INT8 *String,VS_UUID *Uuid);
VS_INT8 *SRPAPI **UuidToMD5**(VS_UUID *Uuid);

4.42 BinBuf interface

4.42.1 get binbuf interface-GetSRPBinBufInterface

class ClassOfSRPBinBufInterface *SRPAPI **GetSRPBinBufInterface**(void);

4.42.2 push binbuf interface to Lua stack-LuaPushBinBuf

VS_BOOL SRPAPI **LuaPushBinBuf**(class ClassOfSRPBinBufInterface *BinBuf , VS_BOOL AutoRelease);
If AutoRelease equals to true, then the function passes address of BinBuf to lua script, and lua is responsible for releasing the pointer. If equals to false,Lua does not responsible for releasing the pointer.

4.42.3 Pop the value of lua stack to binbuf-LuaToBinBuf

class ClassOfSRPBinBufInterface *SRPAPI **LuaToBinBuf**(VS_INT32 Index);

4.42.4 determine whether is binbuf -LuaIsBinBuf

VS_BOOL SRPAPI LuaIsBinBuf(VS_INT32 Index);;

4.43 Parapkg,QueryRecord,SXml,FunctionPara,CommInterface

VS_BOOL SRPAPI LuaPushParaPkg(class ClassOfSRPParaPackageInterface *ParaPkg, VS_BOOL AutoRelease);
VS_BOOL SRPAPI LuaPushQueryRecord(VS_QUERYRECORD *QueryRecord, VS_BOOL AutoRelease);
class ClassOfSRPParaPackageInterface *SRPAPI LuaToParaPkg(VS_INT32 Index);
VS_QUERYRECORD *SRPAPI LuaToQueryRecord(VS_INT32 Index);
VS_BOOL SRPAPI LuaIsParaPkg(VS_INT32 Index);
VS_BOOL SRPAPI LuaIsQueryRecord(VS_INT32 Index);
VS_BOOL SRPAPI LuaPushSXml(class ClassOfSRPSXMLInterface *SXml, VS_BOOL AutoRelease);
class ClassOfSRPSXMLInterface *SRPAPI LuaToSXml(VS_INT32 Index);

VS_BOOL SRPAPI LuaIsSXml(VS_INT32 Index);

VS_BOOL SRPAPI LuaPushFunctionPara(class ClassOfSRPFunctionParaInterface *FunctionPara, VS_BOOL AutoRelease);
class ClassOfSRPFunctionParaInterface *SRPAPI LuaToFunctionPara(VS_INT32 Index);
VS_BOOL SRPAPI LuaIsFunctionPara(VS_INT32 Index);

VS_BOOL SRPAPI LuaPushCommInterface(class ClassOfSRPCommInterface *CommInterface, VS_BOOL AutoRelease);
class ClassOfSRPCommInterface *SRPAPI LuaToCommInterface(VS_INT32 Index);
VS_BOOL SRPAPI LuaIsCommInterface(VS_INT32 Index);

4.44 Restart interface

4.44.1 restart interface

VS_BOOL SRPAPI ProgramRestart();
The function is valid only when cle is started by manager program.

4.45 Http/Ftp download

4.45.1 http/ftp download-HttpDownLoad

VS_BOOL SRPAPI HttpDownLoad(VS_UUID *AttachObjectID, VS_CHAR *ServerUrl,VS_CHAR *ClientPath,VS_CHAR *FileName, VS_FileUpDownLoadCallBackProc CallBackProc, VS_UUID *ObjectID, VS_ULONG Para, VS_BOOL SaveFileFlag)
ObjectID may be set to NULL
VS_UUID *AttachObjectID may be set to NULL
SaveFileFlag: ==true, save to file, otherwise not save.
for example:
HttpDownLoad(NULL,"http://www.srplab.com/Files","e:","srirrlicht_index.htm",NULL,NULL,0,VS_TRUE);

void SRPAPI HttpDownLoadAbort( );
Cancel all Http/FTP downloads.

4.46 Monitor Http download

4.46.1 monitor Http download-RegWebDownFunction/ UnRegWebDownFunction

void SRPAPI RegWebDownFunction(VS_WebDownInfoProc CallBackProc,VS_ULONG Para)
void SRPAPI UnRegWebDownFunction(VS_WebDownInfoProc CallBackProc,VS_ULONG Para)
typedef void (SRPAPI * VS_WebDownInfoProc)(VS_ULONG Para, VS_ULONG uMes, VS_CHAR *FileName, VS_ULONG MaxSize, VS_ULONG CurSize);
uMes:
#define VSFILE_ONDOWNSTART 0 //---start download
#define VSFILE_ONDOWNPROGRESS 1 //---download process
#define VSFILE_ONDOWNFINISH 2 //---finish
#define VSFILE_ONDOWNERROR 3 //---error

4.46.2 print download information-WebDownPrint

void SRPAPI WebDownPrint(VS_ULONG uMes, VS_CHAR *FileName, VS_ULONG MaxLength, VS_ULONG CurLength)
Information will be printed if the function registered.

4.47 Static data download

4.47.1 set static data-SetStaticData

VS_BOOL SRPAPI CanSetStaticData(void *Object,VS_ULONG DataSize);
called at client side to determine whether is permitted to upload.

VS_BOOL **SetStaticData**(VS_UUID *ObjectID,VS_ULONG UniqueDataUnitID,VS_ULONG DataSize,VS_INT8 *DataBuf, VS_STATICID *RetDataVersion);
Updates object static data. If success, it returns the version of the static data. The function will change the corresponding static attribute of the object automaticly.

4.47.2 get static data

VS_INT8 *SRPAPI GetStaticDataEx(void *Object,VS_ULONG UniqueDataUnitID,VS_STATICID *DataVersion,VS_ULONG *DataSize, VS_BOOL AutoDownLoad,VS_CHAR *Token);
Refers to GetStaticData.
Token may be set to NULL, if set, it will be passed to the function registered with **SetQueryStaticDataProc**.

4.48 save static data

4.48.1 save static data-SaveServiceData

void SRPAPI SaveServiceData(VS_UUID *ServiceID);

4.49 get key state[Windows]

4.49.1 get key state-GetKeyState

VS_BOOL SRPAPI **GetKeyState**(VS_INT32 Key)
If the key is pressed, then it returns VS_TRUE. Virtual key codes is defined as follows:

```
#define VS_LBUTTON  0x01
#define VS_RBUTTON  0x02
#define VS_MBUTTON          0x04

#define VS_ESCAPE           0x1B
#define VS_BACKSPACE        0x08
#define VS_TAB              0x09
#define VS_ENTER            0x0D
#define VS_SPACE            0x20

#define VS_SHIFT            0x10
#define VS_CTRL             0x11
#define VS_ALT              0x12
```

```
#define VS_LWIN          0x5B
#define VS_RWIN          0x5C
#define VS_APPS          0x5D

#define VS_PAUSE         0x13
#define VS_CAPSLOCK      0x14
#define VS_NUMLOCK       0x90
#define VS_SCROLLLOCK    0x91

#define VS_PGUP          0x21
#define VS_PGDN          0x22
#define VS_HOME          0x24
#define VS_END           0x23
#define VS_INSERT        0x2D
#define VS_DELETE        0x2E

#define VS_LEFT          0x25
#define VS_UP            0x26
#define VS_RIGHT         0x27
#define VS_DOWN          0x28

#define VS_0             0x30
#define VS_1             0x31
#define VS_2             0x32
#define VS_3             0x33
#define VS_4             0x34
#define VS_5             0x35
#define VS_6             0x36
#define VS_7             0x37
#define VS_8             0x38
#define VS_9             0x39

#define VS_A             0x41
#define VS_B             0x42
#define VS_C             0x43
#define VS_D             0x44
#define VS_E             0x45
#define VS_F             0x46
#define VS_G             0x47
#define VS_H             0x48
#define VS_I             0x49
#define VS_J             0x4A
#define VS_K             0x4B
#define VS_L             0x4C
#define VS_M             0x4D
#define VS_N             0x4E
#define VS_O             0x4F
#define VS_P             0x50
#define VS_Q             0x51
#define VS_R             0x52
#define VS_S             0x53
#define VS_T             0x54
#define VS_U             0x55
#define VS_V             0x56
#define VS_W             0x57
#define VS_X             0x58
```

```
#define VS_Y                0x59
#define VS_Z                0x5A

#define VS_GRAVE            0xC0
#define VS_MINUS            0xBD
#define VS_EQUALS           0xBB
#define VS_BACKSLASH        0xDC
#define VS_LBRACKET           0xDB
#define VS_RBRACKET         0xDD
#define VS_SEMICOLON        0xBA
#define VS_APOSTROPHE       0xDE
#define VS_COMMA            0xBC
#define VS_PERIOD           0xBE
#define VS_SLASH            0xBF

#define VS_NUMPAD0    0x60
#define VS_NUMPAD1    0x61
#define VS_NUMPAD2    0x62
#define VS_NUMPAD3          0x63
#define VS_NUMPAD4          0x64
#define VS_NUMPAD5          0x65
#define VS_NUMPAD6          0x66
#define VS_NUMPAD7          0x67
#define VS_NUMPAD8          0x68
#define VS_NUMPAD9          0x69

#define VS_MULTIPLY   0x6A
#define VS_DIVIDE           0x6F
#define VS_ADD                0x6B
#define VS_SUBTRACT   0x6D
#define VS_DECIMAL  0x6E

#define VS_F1               0x70
#define VS_F2               0x71
#define VS_F3               0x72
#define VS_F4               0x73
#define VS_F5               0x74
#define VS_F6               0x75
#define VS_F7               0x76
#define VS_F8               0x77
#define VS_F9               0x78
#define VS_F10        0x79
#define VS_F11        0x7A
#define VS_F12        0x7B
```

4.50 Service redirect

valid at server service group.

4.50.1 Get depended information of service -GetEnvDependCheckInfo

void *SRPAPI **GetEnvDependCheckInfo**(VS_CHAR *ServiceName,VS_ULONG *Size);
Returns buffer pointer and size, the buffer should be freed using interface function Free.
If ServiceName equals to NULL, then returns information of all depended services. Otherwise, returns information of specific service.

4.50.2 set depended information of new service -SetEnvDependCheckInfo

void SRPAPI **SetEnvDependCheckInfo**(VS_ULONG Size,void *Buf);
The function should be called by host.

4.50.3 get service start type-GetEnvStartType

VS_UINT8 SRPAPI **GetEnvStartType**(void);
Return value is list as follows:
#define SRPENVSTART_FROMDIR ((VS_UINT8)0x00) start from directory contains multiple files.
#define SRPENVSTART_FROMSINGLE ((VS_UINT8)0x01) star from single file
#define SRPENVSTART_FROMCHILDDIR ((VS_UINT8)0x10) return from child service (directory contains multiple files)
#define SRPENVSTART_FROMCHILDSINGLE ((VS_UINT8)0x11) return from child service (single file)

4.50.4 get service parameter-GetEnvPara

class ClassOfSRPParaPackageInterface *SRPAPI **GetEnvPara**(void);
The return value should not be released.

4.50.5 set service parameter-SetEnvPara

void SRPAPI **SetEnvPara**(class ClassOfSRPParaPackageInterface *Para);
Input is parapkg which should be released by the caller.

4.50.6 Get service input parameter-GetEnvInputPara

class ClassOfSRPParaPackageInterface *SRPAPI **GetEnvInputPara**(void);
The return parapkg should not be released.

4.50.7 get parent Url-GetEnvParentUrl

VS_CHAR *SRPAPI **GetEnvParentUrl**(void);

4.50.8 set current Url-SetEnvCurrentUrl

void SRPAPI **SetEnvCurrentUrl**(VS_CHAR *Url);

4.50.9 redirect service-RedirectToUrlRequest[reserved]

VS_INT32 SRPAPI **RedirectToUrlRequest**(VS_CHAR *Url,class ClassOfSRPParaPackageInterface *ParaPkg,RedirectToUrl_InfoProc CallBackProc,VS_ULONG Para,VS_CHAR *WorkDirectory,VS_CHAR *ChildTermScript);
ParaPkg is parameter. When child service is started, it can get the parameter using function GetEnvInputPara.
WorkDirectory may be set to NULL.
RedirectToUrl prototype is as follow:
typedef void (SRPAPI *RedirectToUrl_InfoProc)(class ClassOfBasicSRPInterface *BasicSRPInterface, VS_ULONG Para, VS_CHAR *Url,class ClassOfSRPParaPackageInterface *ParaPkg);

ParaPkg is response information from child service, which format is as folows:
The first parameter is type:
#define SRPENVREDIRECT_DISPINFO 0
#define SRPENVREDIRECT_START 1
#define SRPENVREDIRECT_FAIL 2
#define SRPENVREDIRECT_ABORT 3
//---------If it is DISPINFO,then the second parameter is string, the third parameter is MaxLength, and the forth parameter is CurLength
If current redirect process is not complete, then calling this function will cancel current process.

ChildTermScript is lua script to be executed before child service returns, which may be NULL. In the script, current service is represented by lua variable ”_gService”.

return value:
SRPLOADPROCESS_OK
SRPLOADPROCESS_BUSY
SRPLOADPROCESS_DISABLE
SRPLOADPROCESS_FAIL

4.50.10 redirect fail-RedirectToUrlFail[reserved]

void SRPAPI **RedirectToUrlFail**(VS_CHAR *Url);

4.50.11 display information at child service starup-SetRedirectToUrlInfo[reserved]

void SRPAPI **SetRedirectToUrlInfo**(class ClassOfSRPParaPackageInterface *ParaPkg);

4.50.12 cancel redirect-RedirectToUrlAbort[reserved]

void SRPAPI **RedirectToUrlAbort**();

4.51 Lua script precompile and edit

4.51.1 PreCompile

VS_BOOL SRPAPI **PreCompile**(VS_CHAR * ScriptInterface,VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize, VS_CHAR *Name, VS_CHAR **ErrorInfo);
If returns VS_FALSE, the caller should use ErrorInfo to determine whether success or not. If ErrorInfo returns NULL,then the input is incomplete, and futher input is expected; If is not NULL, then the input contains errors.
if returns value is VS_TRUE, then ErrorInfo will be set to NULL.
ScriptInterface: may be lua,python, or other online script language registered.
ScriptInterfaces length < 16

VS_BOOL SRPAPI **OpenLuaEdit**(VS_CHAR *Module,VS_ULONG Config,VS_BOOL CloseEnable);
Editor is compiled to sharelib SRPLuaEdit.DLL
Config is combination of the following value :
#define SRPLUAEDITMODULECONFIG_SCRIPTCONSOLE 0x00000001
#define SRPLUAEDITMODULECONFIG_PROJECT 0x00000002
#define SRPLUAEDITMODULECONFIG_SRPDOC 0x00000004
Module is reserved.

4.51.2 display information in editor -LuaEditDisp

void SRPAPI **LuaEditDisp**(VS_CHAR *Info);

4.51.3 display help information in editor - LuaEditHelp

void SRPAPI **LuaEditHelp** (VS_INT32 Type,VS_CHAR *HelpInfo);
Type=0 help info; =1 Examples project

4.51.4 close editor -CloseLuaEdit

void SRPAPI **CloseLuaEdit**();

4.52 Register DLL callback

4.52.1 register DLL callback-RegDllCallBack

void SRPAPI RegDllCallBack(VS_MsgCallBackProc MsgCallBackProc, VS_ULONG MsgCallBackPara)
Callback prototype
typedef VS_ULONG (SRPAPI *VS_MsgCallBackProc)(VS_ULONG ServiceGroupID, VS_ULONG uMsg, VS_ULONG wParam, VS_ULONG lParam, VS_BOOL &IsProcessed, VS_ULONG Para);

4.52.2 unregister DLL callback-UnRegDllCallBack

void SRPAPI UnRegDllCallBack(VS_MsgCallBackProc MsgCallBackProc, VS_ULONG MsgCallBackPara)
Callback prototype
typedef VS_ULONG (SRPAPI *VS_MsgCallBackProc)(VS_ULONG ServiceGroupID, VS_ULONG uMsg, VS_ULONG wParam, VS_ULONG lParam, VS_BOOL &IsProcessed, VS_ULONG Para);

4.53 Execute script file

4.53.1 execute script file-DoFile

VS_BOOL SRPAPI DoFile(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8);
VS_BOOL SRPAPI DoFileEx(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8,VS_CHAR *ModuleName);
VS_BOOL SRPAPI PostDoFile(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8);
VS_BOOL SRPAPI PostDoFileEx(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8,VS_CHAR *ModuleName);
PostDoFile will read file content, and then it is same as PostDoBuffer.
ScriptInterfaces length < 16
Python,lua does not support parameter IsUTF8. Whether other scripts support IsUTF8 or not depends on their implementation.
ModuleName is name of the module,may be “” or NULL. This Parameter is only valid for lua and python and java and csharp. For java and csharp, ModuleName is the init class name. For example, the ModuleName is “com.srplab.www.test”.
ModuleName should not set to “cmd”, case insensitive.

4.54 Get UUID,directory,and localip

4.54.1 get UUID-CreateUuid

void SRPAPI CreateUuid(VS_UUID *UuidPtr);

4.54.2 get temporary directory-GetSRPTempPath

void SRPAPI GetSRPTempPath(VS_ULONG BufSize,VS_CHAR *Buf);

4.54.3 get config directory-GetSRPConfigurePath

void SRPAPI GetSRPConfigurePath(VS_ULONG BufSize,VS_CHAR *Buf);

4.54.4 GetCorePath

VS_CHAR *SRPAPI GetCorePath()

4.54.5 GetUserPath

VS_CHAR *SRPAPI GetUserPath()

4.54.6 GetLocalIP

VS_CHAR *SRPAPI GetLocalIP()
note: This function always returns “127.0.0.1” for android

4.54.7 GetLocalIPEx

VS_INT32 SRPAPI GetLocalIPEx(SOCKADDR_IN *SockAddr,VS_INT32 ItemNumber)
note: This function always returns the addressof localhost for android.

4.55 Get current Url

4.55.1 get current Url-GetUrl

void SRPAPI GetUrl(VS_CHAR *Buf,VS_INT32 BufSize);

4.56 Memory file manager: file name is not case sensitive

4.56.1 create memory file-CreateMemoryFile

class ClassOfSRPMemoryFileInterface *SRPAPI CreateMemoryFile();
ClassOfSRPMemoryFileInterface:

1. VS_INT32 SRPAPI GetNumber(); get file number
2. VS_BOOL SRPAPI InsertFile(VS_CHAR *FileName, VS_UINT8 *FileBuf, VS_ULONG FileBufSize, VS_UUID *FileID);
   insert memory data to file
3. VS_BOOL SRPAPI SetFromDisk(VS_CHAR *FileListInfo, VS_CHAR *DiskFileName, VS_ULONG FileStartOffset);
   Load from disk
   //FileListInfo format is:
FileName,0(1B),UnCompressLength(4B),CompressLength(4B),DiskFileOffset(4B),VS_UUID(16B,current is reserved), SupportOSType(4B); 0(1B) to end.
4. VS_BOOL SRPAPI SetFromMemory(VS_CHAR *FileListInfo, VS_UINT8 *FileMemory, VS_ULONG FileStartOffset);
   Load from memory

5. VS_BOOL SRPAPI IsExist(VS_CHAR *FileName);
6. VS_ULONG SRPAPI GetSize(VS_CHAR *FileName) ;
7. VS_ULONG SRPAPI Read(VS_CHAR *FileName, VS_UINT8 *ReadBuf);

4.56.2 get memory file of environment-GetEnvMemoryFile

class ClassOfSRPMemoryFileInterface *SRPAPI GetEnvMemoryFile();
The returned pointer should not be released.

4.56.3 set memory file of environment -SetEnvMemoryFile

void SRPAPI SetEnvMemoryFile(class ClassOfSRPMemoryFileInterface *MemoryFile);
MemoryFile will be released by cle, which should only set one. New value will replace the old one.

4.57 Load and run service

4.57.1 get complete URL-ToAbsoluteUrl

VS_BOOL SRPAPI ToAbsoluteUrl(VS_CHAR *InputUrl,VS_CHAR *OutputUrl,VS_INT32 OutputUrlBufSize);

4.57.2 Load and run Url-RunFromUrl

VS_INT32 SRPAPI RunFromUrl(VS_CHAR *Url,VS_INT8 RestartFlag,VS_BOOL WaitFlag);
Url: may be service files on the web site or local disk.
Service can be packed with starsrvpack tools. The tools can pack service files into a directory, which may be uploaded to website.
For example, pack service Demo to directory c:\Output
Upload files in c:\output to website http://www.XXX.com/Service
Then, uses function RunFromUrl(http://www.xxx.com/service/Demo) to load and run the service.
Service may be also a single file on the website.
Service can be packed with starsrvpack tools to a single file.
For example, pack service Demo to directory c:\Output, which will generate demo.srb
Upload demo.srb in c:\output to website http://www.XXX.com/Service.
Then uses function RunFromUrl(http://www.xxx.com/service/Demo.srb) to load and run the service.
If the extension .srb is limited by the webserver, then the extension may be change to others.
The service may be also a script file on local disk
RunFromUrl("aaa.srp")

The service may be also may be a sharelib on local disk
DLL should exports interface:
VS_BOOL (SRPAPI *SRPServerService_InitProc)(class ClassOfSRPServer *SRPServer)and

Url may be attach parameter such as"?parameter";
hostip=XXX: Redirect host IP, which is valid for files on http or ftp server. It the parameter exists, then url uses this ip address other than returned by DNS.
depend=#depend service 1,# depend service 2, depend service 3; '#' is in front of depend service name, which indicates to download from starcore site, otherwise download from parent url ["depend=" without spaces]
script=lua/python/java/csharp; ["script=" without spaces]
The last string is command string, service can read it from EnvInputPara
for examples:

http://XXX/XX?depend=AAA,#bbb;para1=111
http://XXX/XX?depend=AAA,#bbb;script=python;para1=111
http://XXX/XX?para1=111

depend,script,and command string are seperated by ";".

If use FTP, the user name and password is input as follow:
ftp://XXX?USER=XXX;PASS=XXX/XXX/XXX
ftp://XXX:21?USER=XXX;PASS=XXX/XXX/XXX

If hostip is set, then when download, url will be replaces by hostip. For example:
ftp://XXX?USER=XXX;PASS=XXX/XXX/XXX?hostip=127.0.0.1,
ftp://XXX:21?USER=XXX;PASS=XXX/XXX/XXX?hostip=127.0.0.1,
Download will be started from ftp://127.0.0.1 和 ftp://127.0.0.1:21

The return value is:
SRPLOADPROCESS_OK
SRPLOADPROCESS_BUSY
SRPLOADPROCESS_FAIL

RestartFlag:
VS_RUMFROMURL_NORESTART 0
VS_RUMFROMURL_RESTART 1
VS_RUMFROMURL_WAITRESTART 2

If Url starts with char "@", then the service file locates at local disk.

4.57.3 Load and run from Buf-RunFromBuf[reserved]

VS_INT32 SRPAPI RunFromBuf(VS_CHAR *Buf,VS_ULONG BufSize,VS_INT8 RestartFlag,VS_BOOL WaitFlag);
Run from memory buffer which may be lua file or single service file packed.

Result:
SRPLOADPROCESS_OK
SRPLOADPROCESS_BUSY
SRPLOADPROCESS_FAIL

4.57.4 whether current platform is busy-IsLoadServiceBusy

VS_BOOL SRPAPI IsLoadServiceBusy();
It returns VS_TRUE, then function RunFromUrl,RunFromBuf, RedirectToUrlRequest will be failed.

4.57.5 whether current services are being loaded -IsLoadServiceIdle

VS_BOOL SRPAPI IsLoadServiceIdle();

4.57.6 Set dependent services need to download -SetDepend[reserved]

void SRPAPI SetDepend(VS_CHAR *ServiceName,VS_BOOL DefaultUrlFlag)
If DefaultUrlFlag==VS_TRUE, then download from SRP website.
or else download from current url
valid at server service group,and is used to load service files on the net. For example,
http://www.XXX.com/XXX.SRP

4.58 Service path

4.58.1 Get service path by service name -GetServicePathByName

VS_BOOL SRPAPI GetServicePathByName(VS_CHAR *ServiceName,VS_CHAR *ServicePath,VS_ULONG ServicePathSize)
If returns VS_FALSE, then the service does not exist.

4.58.2 Set service search path-InsertSearchPath

void SRPAPI InsertSearchPath(VS_CHAR *SearchPath);

4.58.3 clear all search path-ClearSearchPath

void SRPAPI ClearSearchPath();

4.58.4 Get first search path-FirstSearchPath

VS_CHAR *SRPAPI FirstSearchPath(VS_QUERYRECORD *QueryRecord);

4.58.5 Get next search path-NextSearchPath

VS_CHAR *SRPAPI NextSearchPath(VS_QUERYRECORD *QueryRecord);

4.59 Open or save file dialog [Window]

4.59.1 Get save file name-GetSaveFile[win32]

VS_BOOL SRPAPI GetSaveFile(VS_CHAR *Caption, VS_CHAR *Filter, VS_ULONG FilterIndex, VS_CHAR *DefExt, VS_CHAR *FileNameBuf,VS_INT32 FileNameBufSize);

4.59.2 Get load file name-GetOpenFile[win32]

VS_BOOL SRPAPI GetOpenFile (VS_CHAR *Caption, VS_CHAR *Filter, VS_ULONG FilterIndex, VS_CHAR *DefExt, VS_CHAR *FileNameBuf,VS_INT32 FileNameBufSize);

4.60 Get static data version

4.60.1 get static data version-GetStaticVersion

void SRPAPI GetStaticVersion(VS_ULONG DataSize,VS_INT8 *DataBuf,VS_STATICID *RetDataVersion);

4.61 Get system Doc object

Doc object is an instace of class VSSYSDOC_CLASSID.

4.61.1 Get Doc class-GetSysDocClass

void *SRPAPI GetSysDocClass();

4.61.2 Get first registered DOC object-FirstDoc

void *SRPAPI FirstDoc(VS_QUERYRECORD *QueryRecord,VS_CHAR **DocName);

4.61.3 Get next registered DOC object -NextDoc

void *SRPAPI NextDoc(VS_QUERYRECORD *QueryRecord,VS_CHAR **DocName);

4.61.4 Register Doc object-RegisterDoc

void SRPAPI RegisterDoc(void *DocObject,VS_CHAR *DocName);

4.61.5 Unregister Doc object -UnRegisterDoc

void SRPAPI UnRegisterDoc(void *DocObject,VS_CHAR *DocName);

4.61.6 Create Doc object event-ProcessSysDocEvent

VS_EVENTPARAM_RUNPARAM *SRPAPI ProcessSysDocEvent(VS_UUID *DocObjectID,VS_UUID *EventID,VS_EVENTPARAM_RUNPARAM *RequestParam);
VSSYSDOC_ONGETTEXT or VSSYSDOC_ONSETTEXT

4.61.7 Register or unregister Doc event handler-RegDocEventFunction/ UnRegDocEventFunction

VS_BOOL SRPAPI RegDocEventFunction(VS_UUID *DocObjectID,VS_UUID *EventID, void *FuncAddr,VS_ULONG Para);
void SRPAPI UnRegDocEventFunction(VS_UUID *DocObjectID,VS_UUID *EventID, void *FuncAddr,VS_ULONG Para);

4.62 Garbage collect

4.62.1 garbage collect-GCCollect

void SRPAPI GCCollect();

4.63 ClipperBoard[Windows]

4.63.1 Copy string to clipboard-ToClipBoard

void SRPAPI ToClipBoard(VS_CHAR *Info);

4.63.2 Paste string from clipboard-FromClipBoard

VS_CHAR *SRPAPI FromClipBoard();

4.64 Windowless Site[windows reserved]

4.64.1 Whether current environment is windowless mode-IsWindowlessSite

VS_BOOL SRPAPI IsWindowlessSite( );

4.64.2 Whether current environment is transparent-IsWindowlessTransparent

VS_BOOL SRPAPI IsWindowlessTransparent( );

4.64.3 Capture/release DC

void SRPAPI Windowless_GetDC(void **hDC,VS_RECT *rEct);
void SRPAPI Windowless_ReleaseDC(void *hDC);

4.64.4 Register/unegister callback-RegWindowlessSiteCallBack/ UnRegWindowlessSiteCallBack

void SRPAPI RegWindowlessSiteCallBack(struct VSWindowlessSiteCallBackInfo *CallBackInfo, VS_UUID *ObjectID, VS_ULONG Para);
void SRPAPI UnRegWindowlessSiteCallBack(struct VSWindowlessSiteCallBackInfo *CallBackInfo, VS_UUID *ObjectID, VS_ULONG Para);
Prototype:
typedef void (SRPAPI *SRPWindowless_DrawProc)(VS_UUID *ObjectID, VS_ULONG Para, void *hDC, void *rcBounds, void *rcInvalid);
typedef VS_BOOL (SRPAPI *SRPWindowless_MessageProc)(VS_UUID *ObjectID, VS_ULONG Para, VS_ULONG uMes,VS_ULONG wParam,VS_ULONG LParam,VS_ULONG *Result);
If returns VS_TRUE, then the callback processes the message, or else not.
typedef VS_BOOL (SRPAPI *SRPWindowless_GetDropTargetProc)(VS_UUID *ObjectID, VS_ULONG Para, void **DropTarget);
If returns VS_TRUE, Droptarget interface is returned, or else Droptarget is not impletemted.

```
struct VSWindowlessSiteCallBackInfo{
   SRPWindowless_DrawProc DrawProc;
   SRPWindowless_MessageProc MessageProc;
   SRPWindowless_GetDropTargetProc GetDropTargetProc;
};
```

4.64.5 Windowless mode function

```
   void SRPAPI Windowless_Draw( void *hDC, void *rcBounds, void *rcInvalid );
   VS_BOOL SRPAPI Windowless_Message( VS_ULONG uMes,VS_ULONG wParam,VS_ULONG
LParam,VS_ULONG *Result );
  VS_BOOL SRPAPI Windowless_GetDropTarget( void **DropTarget );
   void SRPAPI Windowless_Redraw( VS_BOOL fErase );
```

4.65 Get SXML interface

```
class ClassOfSRPSXMLInterface *GetSXMLInterface();
```

4.66 Get FunctionPara interface

```
class ClassOfSRPFunctionParaInterface *SRPAPI GetFunctionParaInterface()
```

4.67 Set port number for debug and client

Valid only on service group 0.

Interface is string, which format is :
“Host=interface address;if=share lib file name(include extension);site=share lib download address, ftp/http address ;para=parameter string”
for example:" host=192.168.0.1;if=SRPTcpLinkInterface1.dll;site=ftp://127.0.0.1"

4.67.1 Set client port number-SetClientPort

```
VS_BOOL SRPAPI SetClientPort(VS_CHAR *ClientInterface,VS_UINT16 ClientPortNumber);
```

4.67.2 Set debug port number-SetDebugPort

```
VS_BOOL SRPAPI SetDebugPort(VS_CHAR *DebugInterface,VS_UINT16 DebugPortNumber);
```

4.68 Get communication interface

```
class ClassOfSRPCommInterface *SRPAPI GetCommInterface()
```

4.69 Set Telnet,Web and output port

Valid only on service group 0.

4.69.1 set Telnet port-SetTelnetPort

```
VS_BOOL SRPAPI SetTelnetPort (VS_UINT16 ClientPortNumber);
```

4.69.2 Set output port-SetOutputPort

VS_BOOL SRPAPI SetOutputPort(VS_CHAR *OutputHost,VS_UINT16 OutputPortNumber);
Using syslog server to receive information, which is coded to UTF-8.

4.69.3 Set Web port-SetWebServerPort

VS_BOOL SRPAPI SetWebServerPort(VS_CHAR *WebServerHost,VS_UINT16 WebServerPortNumber,VS_INT32 ConnectionNumber,VS_ULONG PostSize);
WebServerHost is Url,may be set to NULL
ConnectionNumber: the maximum connection number
PostSize : permitted upload size, unit is Kbytes.

4.70 Get predefined object ID

VS_UUID *SRPAPI GetVSObjectID(VS_INT32 Which);

Which takes vales from:
```
#define VSSYSID_VSSYSOBJ_OBJID          0
#define VSSYSID_VSSYSOBJ_WNDADJUST      1
#define VSSYSID_VSSYSOBJ_WNDCANBERESIZE 2
#define VSSYSID_VSSYSOBJ_WNDRESIZE      3
#define VSSYSID_VSSYSOBJ_EDITSELECT     4
#define VSSYSID_VSSYSOBJ_SETFOCUS       5
#define VSSYSID_VSSYSOBJ_WNDMSG         6

#define VSSYSID_VSSYSDOC_CLASSID        7
#define VSSYSID_VSSYSDOC_ONGETTEXT      8
#define VSSYSID_VSSYSDOC_ONSETTEXT      9
#define VSSYSID_VSSYSDOC_LUA_GETTEXT    10
#define VSSYSID_VSSYSDOC_LUA_SETTEXT    11
#define VSSYSID_VSSYSDOC_ONTEXTCHANGE   12
#define VSSYSID_VSSYSDOC_ONTEXTSELECT   13
```

4.71 Temporary file

VS_BOOL SRPAPI RegTempFile(VS_CHAR *TempFileName,VS_CHAR *OriFileName); /*clear when process not exist*/
OriFileName may be NULL
TempFileName is unique between processes. TempFileName must be full name.

VS_CHAR *SRPAPI GetRegTempFile(VS_CHAR *OriFileName ,VS_CHAR *Buf,VS_INT32 BufSize);
Get temporary filename registered by other process.
If exists, then register it for current process automatically.

void SRPAPI UnRegTempFile(VS_CHAR * TempFileName); /*clear when process not exist*/
Unregister temporary file, the file will be deteled by cle platform.

4.72 Get platform config info

void SRPAPI GetConfigResult(VS_BOOL *DebugCfgResult,VS_BOOL *DirectClientCfgResult,VS_BOOL *TelnetCfgResult,VS_BOOL *WebServerCfgResult);

Get config result.

VS_CHAR *SRPAPI GetConfigEnvTag( );
Get env tag, which is configured using VS_STARCONFIGEX struct at init procedure of starcore.
Three type is defined:
"", "noloop", "activex"

void SRPAPI GetConfig(class ClassOfSRPSXMLInterface *XmlInterface)
void SRPAPI GetConfigHost(VS_CHAR *Buf,VS_INT32 BufSize); Get host + Web port

4.73 Duplicate

class ClassOfSRPControlInterface *SRPAPI Dup();

4.74 Register Dispatch callback

void SRPAPI RegDispatchCallBack(VS_SRPDispatchCallBackProc CallBack,VS_ULONG Para);
void SRPAPI UnRegDispatchCallBack(VS_SRPDispatchCallBackProc CallBack,VS_ULONG Para);

4.75 Query interface

void *SRPAPI QueryInterface(VS_UUID *InterfaceID);
Input is service ID.The function is reserved and returns NULL for current cle version.

4.76 Interactive with environment[reserved]

void SRPAPI RegRunEnv_FromParentCallBack(VS_RunEnvCallBackProc CallBack,VS_ULONG Para);
void SRPAPI UnRegRunEnv_FromParentCallBack(VS_RunEnvCallBackProc CallBack,VS_ULONG Para);
//---real function
VS_BOOL SRPAPI RunEnvToChild(VS_UUID *ObjectID,struct StructOfVSRunEnv *RunEnvInfo);
VS_BOOL SRPAPI RunEnvToParent(struct StructOfVSRunEnv *RunEnvInfo);

4.77 Lock lua table

VS_BOOL SRPAPI LockLuaTable( );
VS_BOOL SRPAPI UnLockLuaTable( );
After being locked, the lua table is readonly, and is not writeable. Lua table includes global table and objects.

4.78 Whether is root service

VS_BOOL SRPAPI IsRootService();
If the servie is import as dynamic service, then the function returns VS_FALSE. Otherwise returns VS_TRUE.

4.79 Reference Count

void SRPAPI AddRef();
Increase reference count;

VS_INT32 SRPAPI GetRef();
Get current reference count

void SRPAPI ReleaseOwner();
Release owner, this function will decrease reference count, but not free the instance.

4.80 Object Reference Count

void SRPAPI AddRefEx(void *Object);
void SRPAPI DelRefEx(void *Object);
VS_INT32 SRPAPI GetRefEx(void *Object);

AddRefEx is same as LockGC
DelRefEx is same as UnLockGC

VS_CHAR *SRPAPI GetRefInfo(void *Object)
object is referenced by scripts, return value is script name string, separated by ","
if object is SLockGC by the script, then there has a "*" prefix before the script name.
for example:
"lua,*python"

VS_BOOL SRPAPI ReleaseOwnerEx(void *Object);
This function should be used when cle object is create by one language and will not used in this language and output to other languages
For c/c++:
This function decreases reference count of object,but not cause object to be freed.

VS_BOOL SRPAPI ReleaseOwnerExForScript(VS_CHAR *ScriptInterface,void *Object)
void SRPAPI CaptureOwnerExForScript(VS_CHAR *ScriptInterface,void *Object)
The above two function should be used in script bridge module. When _ ReleaseOwnerEx function is called, the bridge call ReleaseOwnerExForScript with interface name.
If _SUnLockGC is called, script bridge should call CaptureOwnerExForScript to clear record before.

void SUnLockGC()
This function is provided for check all cle object's with ReleaseOwnerEx record and trigger _SUnLockGC function of the corresponding script interface.

4.81 Get Last Error

VS_INT32 SRPAPI GetLastError();
VS_CHAR *SRPAPI GetLastErrorInfo(VS_UINT32 *LineIndex,VS_CHAR **SourceName);

4.82 Script RawType Function

VS_BOOL SRPAPI **InitRaw**(VS_CHAR *ScriptInterface,class ClassOfSRPInterface *SRPInterface);
Init script interface for special service;

VS_BOOL SRPAPI **LoadRawModule**(VS_CHAR *ScriptInterface,VS_CHAR *ModuleName,VS_CHAR *FileOrString,VS_BOOL IsString,VS_CHAR **ErrorInfo)
Load Raw Module, which may be lua module, python module, java class library, csharp class library. for version 1.90.1, the above four scripts are supported.

ModuleName : if has been loaded, then the function does nothing. Module name may be input NULL for global name space.
RunString : may be file or string buf, which will be executed.
IsString : true for string, false for file.
ErrorInfo : may be null;

for lua :
if RunString == true,then load and execute lua code.
if RunString == false,then load and execute lua file.

for python :

if IsString == false, and FileOrString is valid, then load python file as module.
if IsString == true/false, and FileOrString is invalid, using python import function to load module
if IsString == true, and FileOrString is valid, then load python code as module.

for java :
IsString is ignored.
if FileOrString is valid, then take FileOrString as class library file. Or else, take ModuleName as class library file.

for csharp :
IsString is ignored.
if ModuleName is valid and FileOrString is invalid,then load Assembly by name.
if FileOrString is valid, then load Assembly by file.

VS_BOOL SRPAPI **LoadRawModuleEx**(VS_CHAR *ScriptInterface,VS_CHAR *ModuleName,void *Object,VS_CHAR **ErrorInfo)
This function is valid for csharp, object should be raw assembly object.

void *SRPAPI **GetRawContextBuf**(void *Object,VS_CHAR *ScriptInterface);
This function is provided for script interface.

VS_BOOL SRPAPI **DefScriptRawType**(VS_CHAR *ScriptInterface,VS_CHAR *ModuleName,VS_CHAR *FileOrString,VS_BOOL IsString,VS_CHAR **ErrorInfo);

Custom RawType is a script module. Details refer to script interface document.

VS_INT32 SRPAPI **NewScriptRawTypeGroup**();

Create a groupfor custom rawtype. This function is provided for script interface.

VS_INT32 SRPAPI **GetScriptRawTypeGroup**(VS_INT32 ScriptRawType);

Get rawtype group. This function is provided for script interface.

VS_INT32 SRPAPI **RegScriptRawType**(VS_CHAR *ScriptInterface,VS_CHAR *TypeGroupName,VS_INT32 GroupIndex,VS_CHAR *TypeName);

Register a newrawtype. This function is provided for script interface.

VS_INT32 SRPAPI **GetScriptRawType**(VS_CHAR *ScriptInterface,VS_CHAR *TypeGroupName,VS_CHAR *TypeName)

Get rawtype defined by special script.

VS_CHAR *SRPAPI **GetScriptRawTypeEx**(VS_INT32 RawType,VS_CHAR **ScriptInterface,VS_CHAR **TypeGroupName)

Get rawtype information defined by special script.

4.83 UnLockGC Log

void LogObjectFreeByUnLock(VS_BOOL Flag);
if Flag == true and object is freed by UnLockGC/DelRefEx or script, a message is output to console.

5 ClassOfSRPInterface

Class define : **class ClassOfSRPInterface**

5.1 Get system type

5.1.1 Get system type-GetOsType

VS_UINT32 SRPAPI **GetOsType**( )

5.2 Module interface function

5.2.1 Get object information(current no meaning)—GetObjectRegisterInfo

VS_INT32 **GetObjectRegisterInfo** (VS_UUID ObjectID,VS_INT8 *InBuf,VS_INT32 InBufLength,VS_INT8 *OutBuf,VS_INT32 OutBufSize);
If succeeds, the function returns 0. Otherwise returns other value.

5.2.2 Register object dependency-RegisterObjectDependency

void **RegisterObjectDependency** (VS_UUID ModuleID,VS_UUID ObjectID, VS_INT32 DependIndex,VS_UUID DependObjectID,VS_INT32 DependType,VS_INT32 DependItemNumber,VS_DEPENDATTRIBUTE *DependItemList);

5.2.3 Register object function address-RegisterObjectFunction

void **RegisterObjectFunction** (VS_UUID ModuleID,VS_UUID ObjectID,VS_UUID FunctionID,void *CallBackPtr,VS_INT32 DependIndexNumber,VS_INT32 *DependIndex);

5.2.4 Register object system event address-RegisterObjectSysEvent

void **RegisterObjectSysEvent** VS_UUID ModuleID,VS_UUID ObjectID,void *CallBackPtr,VS_INT32 DependIndexNumber,VS_INT32 *DependIndex);

5.2.5 Register object system edit event address-RegisterObjectSysEditEvent

void **RegisterObjectSysEditEvent** (VS_UUID ModuleID,VS_UUID ObjectID,void *CallBackPtr,VS_INT32 DependIndexNumber,VS_INT32 *DependIndex);

5.2.6 Register object event address-RegisterObjectInEvent

void **RegisterObjectInEvent** (VS_UUID ModuleID,VS_UUID ObjectID,VS_UUID InEventID,void *CallBackPtr,VS_INT32 DependIndexNumber,VS_INT32 *DependIndex);

5.2.7 Register object information address-RegisterQueryObjectInfo

void **RegisterQueryObjectInfo** (VS_UUID ModuleID,VS_UUID ObjectID,void *CallBackPtr)
Prototype is :
typedef void (***VSModuleFunction_QueryObjectInfoProc**)(VS_OBJECTMODULEINFO *VSObjectModuleInfo);

5.2.8 Register object dynamic module-RegisterDynamicModule

VS_INT32 **RegisterDynamicModule**(VS_UUID ModuleID, VSModuleFunction_ModuleInitProc ModuleInitProc, VSModuleFunction_ModuleTermProc ModuleTermProc, VSModuleFunction_RequestRegisterObjectProc RequestRegisterObjectProc);
If succeeds, the function returns 0.
The module must be defined in the service, and it has not been loaded from share library. Three functions must be valid,or else the function will fail.

5.3 UUID string convert function

5.3.1 Convert string to UUID-StringToUuid

VS_BOOL SRPAPI **StringToUuid**(VS_INT8 *String,VS_UUID *Uuid);

5.3.2 Convert UUID to string-UuidToString

VS_INT8 *SRPAPI **UuidToString**(VS_UUID *Uuid);

5.3.3 Convert string to Utf8-StringToUtf8

VS_INT8 *SRPAPI StringToUtf8(VS_INT8 *String)
If the return value is not NULL, then it should be freed by interface function Free.

5.3.4 Convert Utf8 to string-Utf8ToString

VS_INT8 *SRPAPI Utf8ToString(VS_INT8 *String)
If the return value is not NULL, then it should be freed by interface function Free.

5.4 Object common function

5.4.1 Get parent object-GetParent

void ***GetParent**(void *Object);
Parent may be service item or object, which can be distinguished by function IsObject.If the function returns True, then it is normal object, or else is service item.

5.4.2 Get queue index of parent which object belongs to -GetIndex

OBJECTATTRIBUTEINDEX **GetIndex**(void *Object);
Index starts from 0.

5.4.3 Get order in queue which object belongs to-GetOrder

VS_UINT16 GetOrder(void *Object);
Order starts from 0.

5.4.4 Get object class-GetClass

void ***GetClass**(void *Object);

5.4.5 Get object class ID-GetClassID

void **GetClassID**(void *Object, VS_UUID *UuidPtr);

5.4.6 Get objet ID-GetID

void **GetID**(void *Object, VS_UUID *UuidPtr);
VS_UUID *SRPAPI GetIDEx(void *Object)

5.4.7 Get object address-GetObject/ GetSRPObject

void ***GetObject/ GetSRPObject** (VS_UUID *ObjectID);

5.4.8 Get object address by name-GetObjectEx/ GetSRPObjectEx

void ***GetObjectEx/ GetSRPObjectEx** (void *ParentObject,VS_CHAR *Name);
If ParentObject is not NULL, then the function returns the object under ParentObject.
If Name is UUID string, then the function returns the object corresponding to the ID

5.4.9 Get object from special service-GetObjectEx2

void *SRPAPI **GetObjectEx2**(VS_CHAR *ServiceName,VS_CHAR *Name);
If ServiceName is NULL, then the function is same as GetObjectEx.
If Name is UUID string, then the function returns the object corresponding to the ID

5.4.10 Get previous object with same name-GetPrevEx

void *SRPAPI **GetPrevEx**(void *Object);

5.4.11 Get next object with same name -GetNextEx

void *SRPAPI **GetNextEx**(void *Object);

5.4.12 Get firstQueryFirst),next(QueryNext)and previous(QueryPrev)child object

void ***QueryFirst** (void *VSObject); VSObject is address of the object in parent struct.
void ***QueryFirstChild** (void *Object,OBJECTATTRIBUTEINDEX AttributeIndex);
Get first child object from queue of object, which attribute must be pointer.
void ***QueryNext** (void *Object); Object is the return value of QueryFirst or QueryNext function previous called.

void ***QueryPrev*** (void *Object); Object is the return value of QueryFirst or QueryNext function previous called.

5.4.13 Get first(QueryFirstEx)and next(QueryNextEx)object of service

void ***QueryFirstEx*** (VS_QUERYRECORD *QueryRecord);
void ***QueryNextEx*** (VS_QUERYRECORD *QueryRecord);
Returns object or service item in the service, which may be distinguished by function IsObject.

5.4.14 Whether is object-IsObject

VS_BOOL SRPAPI **IsObject**(void *Object);
It returns true, input is object, otherwise is not.

5.4.15 Get first(QueryFirstActiveChild)and next(QueryNextActiveChild) active child object

void *SRPAPI **QueryFirstActiveChild**(void *Object,VS_ULONG *Context)
void *SRPAPI **QueryNextActiveChild**(VS_ULONG *Context)
Context should not be NULL

5.4.16 Whether object is in active set of service item-IsObjectInActiveSet

VS_BOOL SRPAPI IsObjectInActiveSet(void *Object);

5.4.17 Get first instance-QueryFirstInst

void ***QueryFirstInst***(VS_QUERYRECORD *QueryRecord,void *ClassObject);

5.4.18 Get next instance-QueryNextInst

void ***QueryNextInst***(VS_QUERYRECORD *QueryRecord,void *ClassObject);
In this process, if any instance is deleted or cerated, then the traverse will restart again.

5.4.19 Get first instance by class ID-QueryFirstInstEx

void ***QueryFirstInstEx***(VS_QUERYRECORD *QueryRecord,VS_UUID *ObjectClassID);

5.4.20 Get next instance by class ID -QueryNextInstEx

void ***QueryNextInstEx***(VS_QUERYRECORD *QueryRecord,VS_UUID *ObjectClassID);
In this process, if any instance is deleted or cerated, then the traverse will restart again.

5.4.21 close the query-QueryInstClose

//---
void **QueryInstClose** (VS_QUERYRECORD *QueryRecord);
After the above four query function finish, this function should be called to close the query procedure.

5.4.22 Get or set object name(GetName,SetName)

VS_CHAR ***GetName**(void *Object);
void **SetName**(void *Object, VS_CHAR *Name);
//---Can only set name of object dynamicly created.

5.4.23 Whether object is instance of another object-IsInst

VS_BOOL **IsInst**(VS_UUID *ObjectClassID,void *Object);

5.4.24 Whether object is direct instance of another object -IsDirectInst

VS_BOOL IsDirectInst(VS_UUID *ObjectClassID,void *Object);

5.4.25 Whether object is child of another object -IsChild

VS_BOOL IsChild(void *ParentObject,void *Object);

5.4.26 Get service item ID which object belongs to-GetObjectSysRootItemID

void **GetObjectSysRootItemID**(void *Object,VS_UUID *UuidPtr);

5.4.27 Get service item which object belongs to -GetObjectSysRootItem

void ***GetObjectSysRootItem**(void *Object);

5.4.28 Whether object belongs to the service-IsThisService

VS_BOOL **IsThisService** (void *Object);

5.4.29 Whether object belongs to active service-IsActiveServiceObject

VS_BOOL SRPAPI IsActiveServiceObject(void *Object);

5.4.30 Whether object belongs to this client -IsThisClient

VS_BOOL IsThisClient(void *Object); Valid at client side.

5.4.31 Get object client ID(SRPClientID)-GetClientID

VS_ULONG GetClientID(void *Object);

5.4.32 Get or set object WebServiceflag

VS_BOOL SRPAPI GetWebServiceFlag(void *Object);
VS_BOOL SRPAPI SetWebServiceFlag(void *Object,VS_BOOL WebServiceFlag);

5.5 Save or Load object

For object to be saved, its child object is saved only when it is global dynamic or local object. If the child is static object, it is not saved.

5.5.1 Set object save flag - SetSaveFlag

void SRPAPI **SetSaveFlag**(void *Object,VS_UINT8 SaveFlag);
Takes value from VSSAVE_NONE；VSSAVE_LOCAL；VSSAVE_GLOBAL,VSSAVE_SAVE.
For new created object, its save flag is init as VSSAVE_SAVE ; if it need not be saved, you should change its save flag to VSSAVE_NONE.

5.5.2 Get object save flag - GetSaveFlag

VS_UINT8 SRPAPI **GetSaveFlag**(void *Object);

5.5.3 Save object into buffer-SaveToBuf

VS_INT8 *SRPAPI **SaveToBuf**(void *Object, VS_INT32 *ObjectSize , VS_CHAR *Password, VS_UINT8 SaveFlag , VS_BOOL SaveNameValue);
Object's attribute,child objects may be packed into the buffer. In this procedure, event ONSAVE will be triggered to get object's private data;
The function returns buffer pointer and size.It returns NULL if error occurs. The reason may be object's static data need download. The returned buffer pointer should be freed by interface function Free.
SaveFlag : VSSAVE_SAVE, depends on object's save flag.
VSSAVE_LOCAL,saves object and its child objects [except the child object which save flag is VSSAVE_NONE]. The object is saved as local object.
VSSAVE_GLOBAL, saves object and its child objects [except the child object which save flag is VSSAVE_NONE]. The object is saved as global object.
VSSAVE_NONE,Only saves the object, does not save child objects.
The function can not save static object or client dynamic object.
If SaveNameValue = true, then the function saves object' name value, or else not save.

5.5.4 Save object to file-SaveToFile

VS_BOOL SRPAPI **SaveToFile**(void *Object, VS_CHAR *FileName , VS_CHAR *Password, VS_UINT8 SaveFlag , VS_BOOL SaveNameValue);

5.5.5 Load object from buffer-LoadFromBuf/DeferLoadFromBuf

Bool SRPAPI **LoadFromBuf** (void *Object, VS_INT8 *Buf, VS_INT32 BufSize, VS_CHAR *Password, VS_BOOL LoadAsLocal, VS_BOOL LoadNameValue,VS_BOOL UpdateFlag);
void SRPAPI **DeferLoadFromBuf**(void *Object, VS_INT8 *Buf, VS_INT32 BufSize, VS_CHAR *Password, VS_BOOL LoadAsLocal,VS_BOOL LoadNameValue, VS_BOOL UpdateFlag)
The function first clear object's child objects which save flag is VSSAVE_LOCAL and VSSAVE_GLOBAL.
At server side, the function will restore all the object in the buffer.
At client and debug side, the function does not restore global object, and only restore local object.
If LoadAsLocal == true, all the object is restored as local object, otherwise restore object according to its save flag.
If UpdateFlag == true, then updates objects, which uses the saved child object ID. It object with same ID exists and is child of other object, then the function will return fails.
If UpdateFlag ==false, then object's ID will be reallocated.

The function will cause object deactive, and then reactive. So if the function is called in the ACTIVE event hanler, application must using function DeferLoadFromBuf, which will restore object at next Ticket(10ms). In the ACITVE event handler, application should check whether LParam is 1 or not.

5.5.6 Load object from file-LoadFromFile/Defer LoadFromFile

Bool SRPAPI **LoadFromFile**(void *Object, VS_CHAR *FileName, VS_CHAR *Password, VS_BOOL LoadAsLocal, VS_BOOL LoadNameValue,VS_BOOL UpdateFlag, VS_BOOL StaticDataUseFile);
void SRPAPI DeferLoadFromFile(void *Object, VS_CHAR *FileName, VS_CHAR *Password, VS_BOOL LoadAsLocal,VS_BOOL LoadNameValue, VS_BOOL UpdateFlag, VS_BOOL StaticDataUseFile);
If StaticDataUseFile == true, then the file is as cache for static data if exists.
== false,then static data will be loaded in the service data file.

The function will cause object deactivated, and then reactivated. So if the function is called in the ACTIVE event process, must using function DeferLoadFromBuf, which will restore object at next Ticket(10ms). In the ACITVE event process, should check whether LParam is 1 or not.

5.5.7 Clear child object and static data loaded-ResetLoadObject

void SRPAPI **ResetLoadObject**(void *Object);

5.5.8 Register map function of object attribute name-RegLoadMapAttrNameProc

void SRPAPI **RegLoadMapAttrNameProc**(VS_LoadMapAttrNameProc LoadMapAttrNameProc);
The prototype is :
typedef VS_CHAR *(SRPAPI *VS_LoadMapAttrNameProc)(void *Object,VS_CHAR *LoadAttrName);
In normal case, the function is not used.

5.6 Object control

5.6.1 Object control

5.6.2 Object is under local control-IsLocalControl

VS_BOOL SRPAPI **IsLocalControl**(void *Object);
If objet is under local control, the function returns true,otherwise is controlled by server.

5.6.3 Object is created by remote -IsRemoteCreate

VS_BOOL SRPAPI **IsRemoteCreate**(void *Object);
If object is created by remote server, the function returns true.Otherwise object is created locally.
Global static object is always created at server.
Local object is created locally.

5.6.4 Object ID in parent

void SRPAPI **SetIDInParent**(void *Object,VS_UINT16 IDInParent);
VS_UINT16 SRPAPI **GetIDInParent**(void *Object);
void *SRPAPI **GetChildByID**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UINT16 IDInParent); //Get child object based on ID from parent.

0 is invalid.

5.7 Get event or function ID by name

5.7.1 Get object function ID -GetFunctionID

VS_BOOL SRPAPI **GetFunctionID**(void *Object,VS_CHAR *FuntionName,VS_UUID *FunctionID);

5.7.2 Get object event ID-GetInEventID

VS_BOOL SRPAPI **GetInEventID**(void *Object,VS_CHAR *InEventName,VS_UUID *InEventID);

5.7.3 Get object out event ID -GetOutEventID

VS_BOOL SRPAPI **GetOutEventID**(void *Object,VS_CHAR *OutEventName,VS_UUID *OutEventID);

5.7.4 Get object out event name-GetOutEventName

VS_CHAR *SRPAPI GetOutEventName(VS_UUID *OutEventID);

5.8 Get and set object function

5.8.1 Get object function pointer-GetFunctionEx

void ***GetFunctionEx**(void *Object,VS_UUID *FunctionID,VS_BOOL *GlobalFunctionFlag = NULL); //---Get overloaded function pointer.
If GlobalFunctionFlag returns VS_TRUE, then application shoud not attach object as first parameter when calls the function returned.

VS_BOOL SRPAPI **IsGlobalFunctionEx**(void *Object,VS_UUID *FunctionID);

5.8.2 Get function pointer-GetFunction

void ***GetFunction**(VS_UUID *FunctionID ,VS_BOOL *GlobalFunctionFlag = NULL);
// can not get pointer of lua function
If GlobalFunctionFlag returns VS_TRUE, then application shoud not attach object as first parameter when calls the function returned.
VS_BOOL SRPAPI **IsGlobalFunction**(VS_UUID *FunctionID);

5.8.3 Get origin function pointer-GetOriFunction

void ***GetOriFunction**(VS_UUID *FunctionID ,VS_BOOL *GlobalFunctionFlag = NULL);
//---If function OvlProc overloads function OriProc, then the function returns the pointer of OriProc.
If GlobalFunctionFlag returns VS_TRUE, then application shoud not attach object as first parameter when calls the function returned.

5.8.4 Whether function is global function

VS_BOOL SRPAPI **IsGlobalFunction**(VS_UUID *FunctionID);
VS_BOOL SRPAPI **IsGlobalFunctionEx**(void *Object,VS_UUID *FunctionID);

5.8.5 Set function address-SetFunction

void **SetFunction**(VS_UUID *FunctionID,void *FuncAddress);
//----should use carefully.

5.8.6 Ser event handler address-SetInEvent

void **SetInEvent**(VS_UUID *InEventID,void *InEventAddress);
//----should use carefully.

5.8.7 Get system event handler address-GetSysEvent

void ***GetSysEvent**(void *Object , VS_ULONG *Para);
//----should use carefully.

5.8.8 Set system event handler address-SetSysEvent

void **SetSysEvent**(void *Object,void *SysEventAddress , VS_ULONG Para);
//---- should use carefully. Para will be as FunctionChoice passed to event handler.
Event handler prototype:

```
VS_INT32 OnSystemEvent(VS_ULONG FunctionChoice,void *EventPara)
{
VS_EVENTPARAM *LocalEventParaPtr;

LocalEventParaPtr = (VS_EVENTPARAM *)EventPara;
switch( SRPInterface -> GetSysEventID(LocalEventParaPtr) ){
}
return 0;
}
```

Set object's system event address, the object status will be reset. If object is actived, then the following events will be created: ONDEACTIVATING -> ONDEACTIVATE -> ONDESTROY -> ONCREATE -> ONACTIVATING ->ONACTIVATE; otherwise create events ONDESTROY -> ONCREATE.

Note for event :
VSEVENT_SYSTEMEVENT_ONATTRIBUTEBEFORECHANGE, and
VSEVENT_SYSTEMEVENT_ONATTRIBUTECHANGE
The corresponding attribute must be declared to accept this event, which can be defined using service description file or function "CreateAtomicAttribute"

You can uses function "RegBeforeChangeCallBack" and "RegChangeCallBack" to capture the change of object's attributes.

5.8.9 Set object system event mask-SetSysEventMask, GetSysEventMask

void **SetSysEventMask**(void *Object,VS_ULONG EventMask, VSSystemEvent_EventProc EventProc);
EventProc can not be set to NULL.

EventProc only process the following events:
#define VSSYSEVENT_PROCESS_TICKET 0x0001
#define VSSYSEVENT_PROCESS_FRAMETICKET 0x0002
#define VSSYSEVENT_PROCESS_IDLE 0x0004
#define VSSYSEVENT_PROCESS_APPACTIVE 0x0008
#define VSSYSEVENT_PROCESS_APPDEACTIVE 0x0010
#define VSSYSEVENT_PROCESS_SERVICEACTIVE 0x0020
#define VSSYSEVENT_PROCESS_SERVICEDEACTIVE 0x0040
#define VSSYSEVENT_PROCESS_SELFEVENT 0x0080
#define VSSYSEVENT_PROCESS_ACTIVESET 0x0100

SetSysEventMask(Object,Mask,Proc) means set
SetSysEventMask(Object,0, Proc) means cancel

VS_ULONG SRPAPI GetSysEventMask(void *Object, VSSystemEvent_EventProc EventProc);

5.8.10 Set or get child object system event mask-SetChildEventMask,GetChildEventMask

void **SetChildEventMask**(void *Object ,VS_ULONG ClassLayer,VS_ULONG EventMask);
Event is processed by object's event handler.
VS_ULONG **GetChildEventMask**(void *Object ,VS_ULONG ClassLayer);

5.9 Overload object function and event

5.9.1 Overload object function-CreateOVLFunction

VS_BOOL SRPAPI **CreateOVLFunction**(void *Object,VS_UUID *OriginFunctionID,void *FuncAddress,VS_UUID *NewFunctionID).
Dynamicly overload object function, valid at local, and will not be synchronized to client or other server.
OriginFunctionID is the ID of function being overloaded.
Lua function and private function can not be overloaded.

5.9.2 Overload object event process-CreateOVLInEvent

VS_BOOL SRPAPI **CreateOVLInEvent**(void *Object,VS_UUID *OutEventID,void *InEventAddress,VS_UUID *NewInEventID).
Dynamicly overload object event handler , valid at local, and will not be synchronized to client or other server.
OutEventID is the ID of the event being overloaded**[note: event should be static event,or else,can not be overloaded.Application can only register its handler using function RegEventFunction]**

5.10 Object embeding script and its running

5.10.1 Name script format

Script is lua script, and the function likes:
Function ScriptName(self, Para1, Para2, …)

End

For executing, cle will pop arguments which total number is nargs from lua stack, and then call the corresponding script.

5.10.2 EventID to VS_ULONG-EventIDToVS_ULONG

VS_ULONG EventIDToVS_ULONG(VS_UUID *OutEventID);
Event must be defined, or else the function returns 0.

5.10.3 Event Name to VS_ULONG-EventNameToVS_ULONG

VS_ULONG **EventNameToVS_ULONG**(VS_UUID *OutEventID);
Event must be defined, or else the function returns 0.

5.10.4 Create object name script-CreateNameScript

VS_BOOL **CreateNameScript**(void *Object, VS_CHAR *ScriptName, VS_CHAR *ScriptBuf);
Valid at client and server, which is used to create lua script function.
ScriptBuf format:
script stars by prefix '@', follows by script interface name, and a space. then script body.
if '@' does not exist, default is lua script.
script interface length should be less than 15 bytes.
note: current version only supports lua.

5.10.5 Create object name script -CreateNameScriptEx

VS_BOOL **CreateNameScriptEx**(void *Object, VS_CHAR *ScriptName, VS_CHAR *FileName);

5.10.6 Delete object name script -DeleteNameScript

void **DeleteNameScript**(void *Object, VS_CHAR *ScriptName);

5.10.7 Change object name script -ChangeScriptName

void SRPAPI **ChangeScriptName**(void *Object, VS_CHAR *ScriptName, VS_CHAR *NewScriptName);

5.10.8 Get content of name string-GetNameScript

VS_INT8 *SRPAPI GetNameScript(void *Object, VS_CHAR *ScriptName);

5.10.9 Execute object name script-ExecNameScript

VS_BOOL **ExecNameScript**(void *Object, VS_CHAR *ScriptName, VS_INT32 nArgs,VS_INT32 nOutArgs);
//--If nOutArgs is less than 0, then the number of values returned is determined by script. Otherwise the numberof value returned is nOutArgs. The order of script executed is as follow:
1:Attribute define in Lua.
2:Function define in Lua.
3:Attribute defined in Lua of the class of the object.
4:Attribute defined in C/C++
6:{Function defined in Lua of the class of the object.
7: Name script defined in the class of the object }

8:Dynamic registered function
9:Function defined in the object
10:Object's name script
11:Dynamic function registered in the class of the object.
12:Function defined in the class of the object
13:Name script defined in the class of the object
If the searching result is attribute, not function, then the function call will be failed.
The function pops nArgs argments from Lua stack, and push the result to Lua stack(if return value exists).Returns true, indicates the function is executed, or else not.

The caller is responsible for maintain the consistency of Lua stack after the function call, excess return values should be pop up and discarded.

Applications may register hook function using LuaRegHook.
It hook function returns true, then the function is processed, and the call procedure has been finished.

VS_BOOL **ExecNameScriptEx**(void *Object, VS_CHAR *ScriptName, VS_INT32 nArgs,VS_INT32 nOutArgs);
The function does not process hook function.

5.10.10 Execute script segment-ExecScript

VS_BOOL SRPAPI **ExecScript**(void *Object, VS_CHAR *FuncName,VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize,VS_INT32 nArgs,VS_INT32 nOutArgs);
Compiles and executes script segment. Function with FuncName should be defined in the script segment

```
Function FuncName(self,Para,..)

end
```

For example:
ExecScript(Object,"ObjPrint","function ObjPrint(self) print(self) end",100,0,0);
If there is registered compile callback function, then before the script is compiled, the function will be called.

VS_BOOL SRPAPI ExecScriptEx(void *Object,VS_CHAR *FuncName,VS_CHAR *FileName,VS_INT32 nArgs,VS_INT32 nOutArgs);

5.10.11 Query first name script-QueryFirstNameScript

VS_CHAR ***QueryFirstNameScript**(void *Object, VS_CHAR **ScriptPtr);
Return the pointer of script name, and save buffer pointer in ScriptPtr.

5.10.12 Query first name script-QueryNextNameScript

VS_CHAR ***QueryNextNameScript**(VS_CHAR **ScriptPtr);
Return the pointer of script name, and save buffer pointer in ScriptPtr.

5.10.13 Force to recompile the script-ForceReCompile

void **ForceReCompile**(void *Object, VS_CHAR *ScriptName);
In normal case, the script is compiled only once. The function will force to recompile the script.

5.10.14 Register script compile callback-RegCompileFunc

void **RegCompileFunc**(VSModuleFunction_ScriptCompileHookProc HookProc,VS_ULONG Para);
Prototype of the function is:

typedef void (SRPAPI *VSModuleFunction_ScriptCompileHookProc)(void *L, VS_ULONG Para,void *Object, VS_CHAR *ScriptName, VS_CHAR *ScriptBuf);
The name of the script is : objectname+'$'+scriptname.

5.11 Object remote call

5.11.1 Get client ID of the remotecall-GetRemoteID

VS_ULONG **GetRemoteID**(void *Object);
The function is used to determine which client initiates the remotecall.

5.11.2 Remotecall object function-RemoteCall,RemoteCallEx

void **RemoteCall**(VS_ULONG ClientID, void *Object,VS_UUID *FunctionID,...);
void **RemoteCallVar**(VS_ULONG ClientID, void *Object,VS_UUID *FunctionID ,va_list argList);
//----At client or debug side, ClientID is ignored; at server side, if ClientID equals to 0,then call the function of all clients, or else is the specific client. Server can only remotecall the function of client, and can not remotecall the function of debugserver.

For the caller:
If parameter is parapkg or binbuf, the pointer should be freed by the caller.

Value types supported by remotecall is list below:
VS_BOOL,VS_INT8, VS_UINT8, VS_INT16, VS_UINT16,VS_INT32,VS_UINT32,VS_LONG,VS_ULONG, VS_FLOAT
string(VSTYPE_CHARPTR),parapkg(VSTYPE_PARAPKGPTR),binbuf(VSTYPE_BINBUFPTR),object(VS_OBJPTR)

void **RemoteCallEx**(VS_ULONG ExcludeClientID, void *Object,VS_UUID *FunctionID,...);
void **RemoteCallExVar**(VS_ULONG ExcludeClientID, void *Object,VS_UUID *FunctionID ,va_list argList);
Remotecall function of all the clients except ExcludeClientID.

VS_OBJPTR: for global and client object, UUID is passed by remotecall; for local object, the object will be packed into buffer to send to client.

5.11.3 Get remotecall source tag-GetRemoteSourceTag

VS_UINT16 SRPAPI **GetRemoteSourceTag** (void *Object);
Current definition is VSRCALLSRC_C,VSRCALLSRC_LUA,VSRCALLSRC_WEBSERVICE

5.11.4 Whether the function is remotecalled-IsRemoteCall

VS_BOOL SRPAPI IsRemoteCall(void *Object);

5.11.5 Get remotecll attach parameter-GetRemoteAttach

void *SRPAPI GetRemoteAttach(void *Object)
Valid for VSRCALLSRC_WEBSERVICE, which is defined as follows:
struct StructOfVSRemoteCallRequestAttach_WebService{
  struct StructOfSRPComm_HttpOnRequest *HttpRequest;

```
    class ClassOfSRPSXMLInterface *SoapInfo;
    VS_CHAR *OperationName;
    struct{
        VS_ULONG MimeDataSize;
        VS_INT8  *MimeDataBuf;
    }MimeData;
    VS_CHAR  *MimeContentType;
};
```

5.11.6 Get remotecall name-GetRemoteCallName

VS_CHAR *SRPAPI GetRemoteCallName(void *Object);
Returns the function name being called, which will be used in response.

5.12 Client Tag

5.12.1 Get client tag of remotecall-GetRemotePrivateTag

VS_ULONG SRPAPI **GetRemotePrivateTag**(void *Object);
Get client tag of remotecall, which is used to determine whether the client is legal.

5.12.2 Set client tag-SetPrivateTag

void SRPAPI **SetPrivateTag**(VS_ULONG ClientPrivateTag);
Valid at client side. The tag will be used in remotecall, attribute change of dynamic object, or the creation of dynamic object, which is used by server to check the client legality.

5.13 Local dynamic value set and get

5.13.1 Get class layer-GetLayer

VS_ULONG **GetLayer**(void *Object);
Object layer of class, start from 1. If the layer is 1, then the object has no parent class.

5.13.2 Set object private value-SetPrivateValue

void **SetPrivateValue**(void *Object,VS_ULONG ClassLayer,VS_ULONG Index, VS_ULONG Value);
Index may not be repeated with the same layer
Index, the higher 4 bits is defined as follows:
0000 - Reserved
0001-0111: Reserved
1000: used by editor
1001-1111:reserved

5.13.3 Get object private value-GetPrivateValue

VS_BOOL **GetPrivateValue**(void *Object,VS_ULONG ClassLayer,VS_ULONG Index, VS_ULONG *Value , VS_ULONG DefaultValue);
Return false, if not exists.

5.13.4 alloc object private buf-MallocPrivateBuf

void *SRPAPI **MallocPrivateBuf**(void *Object,VS_ULONG ClassLayer,VS_ULONG Index, VS_INT32 BufSize);
Index may not be repeated with the same layer
Index, the higher 4 bits is defined as follows:
0000 - Reserved
0001-0111: Reserved
1000: used by editor
1001-1111:reserved

5.13.5 Get object private buffer-GetPrivateBuf

void *SRPAPI **GetPrivateBuf**(void *Object,VS_ULONG ClassLayer,VS_ULONG Index, VS_INT32 *BufSize) BufSize may be set to NULL

5.13.6 Free object private buffer or value-FreePrivate

void SRPAPI **FreePrivate**(void *Object,VS_ULONG ClassLayer,VS_ULONG Index);

5.14 Set or get object global value

The function should be called at client or server side. If it is called at server, the value will be sync to client automatically. If is called at client, then the change is at local.

LocalChange is only valid at server, ==VS_TRUE change locally, not sync to client.

Type is defined as follows:
#define SRPPARATYPE_INT 1 //--integer
#define SRPPARATYPE_FLOAT 2 //--float
#define SRPPARATYPE_BIN 3
#define SRPPARATYPE_CHARPTR 4 //--string
#define SRPPARATYPE_TIME 5 //--time
#define SRPPARATYPE_BOOL 6 //--bool

5.14.1 Set Bool value-SetNameBoolValue

VS_BOOL SRPAPI **SetNameBoolValue**(void *Object, VS_CHAR *Name, VS_BOOL Value, VS_BOOL LocalChange);

5.14.2 Get Bool value-GetNameBoolValue

VS_BOOL SRPAPI **GetNameBoolValue**(void *Object, VS_CHAR *Name, VS_BOOL *Value, VS_BOOL DefaultValue);

5.14.3 Set integer value-SetNameIntValue

VS_BOOL SRPAPI **SetNameIntValue**(void *Object, VS_CHAR *Name, VS_INT32 Value, VS_BOOL LocalChange);

5.14.4 Get integer value-GetNameIntValue

VS_BOOL SRPAPI **GetNameIntValue**(void *Object, VS_CHAR *Name, VS_INT32 *Value, VS_INT32 DefaultValue);

5.14.5 Set float value-SetNameFloatValue

VS_BOOL SRPAPI **SetNameFloatValue**(void *Object, VS_CHAR *Name, VS_DOUBLE Value, VS_BOOL LocalChange);

5.14.6 Get float value-GetNameFloatValue

VS_BOOL SRPAPI **GetNameFloatValue**(void *Object, VS_CHAR *Name, VS_DOUBLE *Value, VS_DOUBLE DefaultValue);

5.14.7 Set binbuf value-SetNameBinValue

VS_BOOL SRPAPI **SetNameBinValue**(void *Object, VS_CHAR *Name, VS_INT8 *Value, VS_UINT16 ValueSize, VS_BOOL LocalChange);

5.14.8 Get binbuf value-GetNameBinValue

VS_INT8 *SRPAPI **GetNameBinValue**(void *Object, VS_CHAR *Name, VS_UINT16 *ValueSize);

5.14.9 Set string value-SetNameStrValue

VS_BOOL SRPAPI **SetNameStrValue**(void *Object, VS_CHAR *Name, VS_CHAR *Value, VS_BOOL LocalChange);

5.14.10 Get string value-GetNameStrValue

VS_CHAR *SRPAPI **GetNameStrValue**(void *Object, VS_CHAR *Name, VS_CHAR *DefaultValue);

5.14.11 Set time value-SetNameTimeValue

VS_BOOL SRPAPI **SetNameTimeValue**(void *Object, VS_CHAR *Name, VS_TIME *Value, VS_BOOL LocalChange);

5.14.12 Get time value-GetNameTimeValue

VS_BOOL SRPAPI **GetNameTimeValue**(void *Object, VS_CHAR *Name, VS_TIME *Value, VS_TIME *DefaultValue);

5.14.13 Delete name value-FreeNameValue

void SRPAPI **FreeNameValue**(void *Object, VS_CHAR *Name);

5.14.14 Delete all name value-FreeAllNameValue

void SRPAPI **FreeAllNameValue**(void *Object);

5.14.15 Get name value type-GetNameValueType

VS_UINT8 SRPAPI GetNameValueType(void *Object,VS_INT8 *Name);

5.14.16 Query first name value-QueryFirstNameValue

VS_CHAR *SRPAPI QueryFirstNameValue(void *Object, VS_ULONG *Context, VS_UINT8 *Type);

5.14.17 Query next name value -QueryNextNameValue

VS_CHAR *SRPAPI QueryNextNameValue(void *Object, VS_ULONG *Context, VS_UINT8 *Type);

5.14.18 Register name value change callback-RegNameValueChangeCallBack

VS_BOOL SRPAPI **RegNameValueChangeCallBack**(void *Object,VS_ObjectNameValueChangeNotifyProc ObjectNameValueChangeNotifyProc,VS_ULONG Para)

5.14.19 Unregister name value change callback -UnRegNameValueChangeCallBack

void SRPAPI **UnRegNameValueChangeCallBack**(void *Object,VS_ObjectNameValueChangeNotifyProc ObjectNameValueChangeNotifyProc,VS_ULONG Para)

The prototype of callback is as:
typedef void (SRPAPI *VS_ObjectNameValueChangeNotifyProc)(void *Object,VS_ULONG Para,VS_CHAR *Name,VS_ULONG NameHashValue);

5.15 Whether object and service item is sync or not

5.15.1 Get object sync status-GetSyncStatus

VS_UINT8 GetSyncStatus(void *Object);

5.15.2 Get service item sync status-GetSyncGroupStatus

VS_UINT8 **GetSyncGroupStatus**(VS_UUID *SysRootItemID, VS_SYNCGROUP GroupIndex);
VS_UINT8 **GetSyncGroupStatusEx**(VS_CHAR *SysRootItemName, VS_SYNCGROUP GroupIndex);

5.15.3 Change object syncgroup-SetSyncGroup

void **SetSyncGroup** (void *Object, VS_SYNCGROUP GroupIndex);
For global dynamic object, static object, or client dynamic object, the function is only valid at server. object syncgroup starts from 1.
Object must belong to one service item, or else the function takes no effect.
If parent object syncgroup has been set, then child object syncgroup can not be set , which is to avoid sync problem if they are different.

5.15.4 Get object syncgroup-GetSyncGroup

void **GetSyncGroup** (void *Object, VS_SYNCGROUP *GroupIndex);
If syncgroup is not set, then return its parent's syncgroup, and so on.

5.16 Service related function

5.16.1 Get active service-GetActiveService

void *SRPAPI **GetActiveService**();

5.16.2 Get current serivce-GetService

void *SRPAPI **GetService**();

5.16.3 Get active service path-GetActiveServicePath

VS_INT32 **GetActiveServicePath** (VS_INT8 *Buf,VS_INT32 BufSize);

5.16.4 Get current service name-GetActiveServiceName

VS_CHAR * GetActiveServiceName ();

5.16.5 Get current service path-GetServicePath

VS_INT32 **GetServicePath**(VS_INT8 *Buf,VS_INT32 BufSize);

5.16.6 Get service name-GetServiceName

VS_CHAR *GetServiceName();

5.16.7 Get service frame interval-GetServiceInterval

VS_INT32 GetServiceInterval( ); //--Unit is 10ms

5.16.8 Get ID of current/active service-GetService/ActiveServiceID

void **GetServiceID**(VS_UUID *UuidPtr);
void **GetActiveServiceID**(VS_UUID *UuidPtr);

5.16.9 Load service-StartVSService(The function is reserved)

void **StartVSService**(VS_UUID *ServiceID);

5.16.10 Exit service-ExitVSService

void **ExitVSService**(); Exit current active service.

5.16.11 Save service-SaveVSService

void **SaveService**(VS_CHAR *Path); Path may be set to NULL,in this case, default path is used.

5.16.12 Whether service is changed-IsServiceChange

VS_BOOL IsServiceChange();

5.16.13 Whether service is active-IsServiceActive

VS_BOOL **IsServiceActive**();

5.16.14 Get service statistic-GetServiceInfo

void **GetServiceInfo**(VS_SERVICEINFO *ServiceInfo);

5.16.15 Query first depend-QueryFirstDepend

VS_BOOL SRPAPI **QueryFirstDepend** (VS_QUERYRECORD *QueryRecord,VS_UUID *ServiceID, VS_UUID *RetUuid,VS_CHAR **RetServiceName);
If ServiceID equals NULL, then returns the depended service of this service .

5.16.16 Query next depend-QueryNextDepend

VS_BOOL SRPAPI **QueryNextDepend** (VS_QUERYRECORD *QueryRecord, VS_UUID *RetUuid,VS_CHAR **RetServiceName);

5.16.17 Whether os is support- IsOsSupport

VS_BOOL SRPAPI **IsOsSupport**(VS_UINT16 ProgramRunType,VS_UINT32 OsType)
The condition is the corresponding OS sharelib exists for modules of the service and its depended service.

5.16.18 Get service frame pulse-GetFrameTicket

VS_ULONG SRPAPI **GetFrameTicket**( );

5.16.19 Export module-ExportModule

VS_BOOL SRPAPI **ExportModule**(VS_CHAR *XmlCfgFile, VS_CHAR **ErrorInfo)
Valid at server.

Example of XmlCfgFile format:

```
<?xml version="1.0" standalone="no" encoding="utf-8" ?>
<ExportModuleInfo  ExportModuleDir=D:\Work\VS_NEW\Examples>
   <SRPFSEngineBasicBCEditModule>
      <DriveClass/>
   </SRPFSEngineBasicBCEditModule>
   <SRPFSEngineBCModule>
```

```
    <FileToolClass/>
  </SRPFSEngineBCModule>
  <SRPFSEngineBasicModule>
    <DriveClass/>
    <DirectoryClass/>
    <FileClass/>
  </SRPFSEngineBasicModule>
</ExportModuleInfo>
```

5.17 Get service group ID

VS_ULONG SRPAPI **GetServiceGroupID**( );

5.18 Print function

5.18.1 Print

void **Print**(VS_CHAR * format,…);
void **PrintVar** (VS_CHAR * format ,va_list argList);

5.18.2 PrintLua

void **PrintLua**(VS_CHAR * format,…); Print Lua information
void **PrintLuaVar**(VS_CHAR * format ,va_list argList); Print Lua information

5.18.3 Lua print function-SetPrintToLua

void SRPAPI SetPrintToLua(VS_BOOL PrintFlag);
When call Print, if PrintFlag equals to true, then output to Lua window, if PrintFlag equals to false, then not output to Lua window.

5.18.4 MessageBox

void SRPAPI **MessageBox**(VS_CHAR *Caption, VS_CHAR *format,...);
void SRPAPI **MessageBoxVar**(VS_CHAR *Caption, VS_CHAR *format ,va_list argList);

5.18.5 Register MessageBox function-RegMessageBoxFunction

void SRPAPI **RegMessageBoxFunction**(void *Object,VS_MessageBoxProc MessageBoxProc);
Replace MessageBox function.
typedef void (SRPAPI *VS_MessageBoxProc)(void *Object, VS_CHAR *Caption, VS_CHAR *Info);

5.18.6 UnRegister MessageBox function -UnRegMessageBoxFunction

void SRPAPI **UnRegMessageBoxFunction**(void *Object,VS_MessageBoxProc MessageBoxProc);

5.18.7 Output error information-ProcessError

void **ProcessError**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format,…);
void **ProcessErrorVar**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format ,va_list argList);

void **ProcessLuaError**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format,…);
void **ProcessLuaErrorVar**(VS_INT32 AlarmLevel, VS_CHAR *SourceName,VS_INT32 LineIndex, VS_CHAR * format ,va_list argList);

5.18.8 Whether object is traced-IsBeingTrace[reserved]

VS_BOOL IsBeingTrace(void *Object);
The function is reserved.

5.18.9 Display trace information-Trace[reserved]

void **Trace**(VS_CHAR *SourceName, VS_INT32 BinaryBufSize,VS_INT8 *BinaryBuf, VS_CHAR * format,…);
void **TraceVar**(VS_CHAR *SourceName, VS_INT32 BinaryBufSize,VS_INT8 *BinaryBuf, VS_CHAR * format ,va_list argList);

The function is reserved.

5.18.10 Lua display information capture and release-CaptureLuaDisp/ ReleaseLuaDisp

void SRPAPI **CaptureLuaDisp**(VS_LuaInfoDispProc DispProc,VS_ULONG Para);
void SRPAPI **ReleaseLuaDisp**(VS_LuaInfoDispProc DispProc,VS_ULONG Para);

typedef void (SRPAPI *VS_LuaInfoDispProc)(VS_CHAR *DispInfo,VS_ULONG Para);

5.19 Service object table

InsertToSDT,DelFromSDT,QueryFirstFromSDT,QueryNextFromSDT,QueryFirstInstFromSDT,QueryNextInstFromSDT.

Limited within a service scope.
void **InsertToSDT**(void *Object);
void **DelFromSDT**(void *Object);
void ***QueryFirstFromSDT**(VS_QUERYRECORD *QueryRecord);
void ***QueryNextFromSDT**(VS_QUERYRECORD *QueryRecord);
void ***QueryFirstInstFromSDT**(VS_QUERYRECORD *QueryRecord,VS_UUID *ObjectClassID);
void ***QueryNextInstFromSDT**(VS_QUERYRECORD *QueryRecord,VS_UUID *ObjectClassID);

5.20 Object create,free and change

The pointer queue which is not global attribute, can not create dynamic child object. For global static, global or client object, except ChangeObject function, other functions are invalid at debug. But for local object, they are valid at debug.

When create object, if parent is service item, then attribute index should be set to 0.

5.20.1 Create global object-MallocGlobalObject

void ***MallocGlobalObject**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf,VS_ULONG ClientID);
Create global object, SRPClientID is ignored at client side. At server side, if object is created for server self, SRPClientID should be set to 0.
ObjectClassID may be set to NULL.

5.20.2 Create global object with ID-MallocGlobalObjectEx

void ***MallocGlobalObjectEx**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf,VS_ULONG ClientID);
Create global object, SRPClientID is ignored at client side. At server side, if object is created for server self, SRPClientID should be set to 0.
ObjectClassID may be set to NULL.

5.20.3 Create local object-MallocObject, MallocObjectL

void ***MallocObject**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf);
void ***MallocObjectL**UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf);
MallocObjectL creates object in current service context, if the service is deactivated, then these objects will be freed by CLE.
ObjectClassID may be set to NULL.

5.20.4 Create local object with ID-MallocObjectEx, MallocObjectLEx

void ***MallocObjectEx**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf);
void ***MallocObjectLEx**(VS_UUID *ObjectID, VS_UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf);
MallocObjectEx creates object in current service context, if the service is deactivated, then these objects will be freed by CLE.
ObjectClassID may be set to NULL.

5.20.5 Create client object-MallocClientObject

void ***MallocClientObject**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf,VS_ULONG ClientID);
// SRPClientID is ignored at client side.
ObjectClassID may be set to NULL.

5.20.6 Create client object with ID-MallocClientObjectEx

void ***MallocClientObjectEx**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_INT32 AttachBufSize,void *AttachBuf,VS_ULONG ClientID);
// SRPClientID is ignored at client side.
ObjectClassID may be set to NULL.

5.20.7 Wait object created finish-WaitMalloc

VS_BOOL SRPAPI WaitMalloc(void *Object);

Valid at client side. When object to be created is global or client object, client may wait confirmation from server. If the function returns VS_FALSE, the object is failed to create or connection to server is closed.

5.20.8 Get object operation code-GetOPPermission

VS_ULONG SRPAPI GetOPPermission(); Valid at client.
Return value is the combination of
VSCLIENTOP_CREATE,VSCLIENTOP_DELETE,VSCLIENTOP_CHANGE.

5.20.9 Copy object attribute-CopyObject

VS_BOOL SRPAPI **CopyObject**(void *Object,void *SrcObject)
The function does not copy pointer and local attribute.
The two objects should belong to same class

5.20.10 Free object-FreeObject

void **FreeObject**(void *Object);
For global, static or client object, the function is invalid at debug.
At client side, the object is not freed intermidately because the request will be send to server.

5.20.11 Defer free object-DeferFreeObject

void **DeferFreeObject**(void *Object);
The object will be freed at next Ticket. At client side, the function is the same as FreeObject.
For global,static or client object, the function is invalid at debug side.

5.20.12 Whether object is in free-IsObjectInFree

VS_BOOL SRPAPI **IsObjectInFree**(void *Object);

5.20.13 Local change object attribute-ChangeLocal

void **ChangeLocal**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex, VS_INT8 *NewValue);
The change of attribute will trigger two events OnAttributeBeforeChange and OnAttribute. The change is valid at local, not sync to client.
For VSTRING attribute, input should use normal string pointer, which may be set to NULL.

5.20.14 Global change object attribute-ChangeObject[Valid for global,client and static object]

void **ChangeObject**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex, VS_INT8 *NewValue);
The change of attribute will trigger two events OnAttributeBeforeChange and OnAttribute. The change will be synchronized to client.
For VSTRING attribute, input uses normal string pointer, which may be set to NULL.
Attribute may be changed at client side,in this case, the attribute is changed locally, and then pack into buffer to send to server. The server will broadcast the change to other clients.
Debug side does not support this function.

5.20.15 Change parent object-ChangeParent

void **ChangeParent**(void *Object,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex);
Valid at server and client side.
ParentObject may be service item, in this case AttributeIndex should be set to 0.

Change parentobject may cause object status is reset. If object is active, then the following event will be triggered:

ONDEACTIVATING,ONDEACTIVATE,ONDESTROY,ONCREATE,ONACTIVATING,ONACTIVATE.

5.20.16 Mark object attribute change-MarkChange

void SRPAPI **MarkChange** (void *Object,OBJECTATTRIBUTEINDEX AttributeIndex);
At server side, marks object attribute change, which will be sent to client at next frame pulse.

5.20.17 Object attribute change callback

VS_BOOL SRPAPI **RegBeforeChangeCallBack**(void *Object,VS_ObjectBeforeChangeNotifyProc ObjectBeforeChangeNotifyProc , VS_ULONG Para,VS_BOOL ChildNotify);
VS_BOOL SRPAPI **RegChangeCallBack**(void *Object,VS_ObjectChangeNotifyProc ObjectChangeNotifyProc ,VS_ULONG Para,VS_BOOL ChildNotify);
void SRPAPI **UnRegBeforeChangeCallBack**(void *Object,VS_ObjectBeforeChangeNotifyProc ObjectBeforeChangeNotifyProc ,VS_ULONG Para);
void SRPAPI **UnRegChangeCallBack**(void *Object,VS_ObjectChangeNotifyProc ObjectChangeNotifyProc ,VS_ULONG Para);

ChildNotify = true. If child object attribute changes, then the callback will be called.

Prototype of the callback:
typedef VS_BOOL (SRPAPI *VS_ObjectBeforeChangeNotifyProc)(void *Object, VS_ULONG Para, OBJECTATTRIBUTEINDEX AttributeIndex, VS_INT8 *NewValue,VS_INT32 DebugEditFlag);
//--Returns true, permit change, or else, not permit. For normal change, DebugEditFlag = 0. For debug server change: DebugEditFlag = 1.
typedef void (SRPAPI *VS_ObjectChangeNotifyProc)(void *Object, VS_ULONG Para, OBJECTATTRIBUTEINDEX AttributeIndex ,VS_ATTRIBUTEINDEXMAP *AttributeIndexMap);
AttributeIndex: attribute to be changed
AttributeIndexMap: bit mask of attributes being changed in this change procedure.

5.20.18 Get object alloc type-GetAllocType

VS_ULONG GetAllocType(void *Object);

5.20.19 Callback of object memory free or address change-RegReMallocCallBack

VS_BOOL SRPAPI **RegReMallocCallBack**(void *Object,VS_ObjectReMallocNotifyProc ObjectReMallocNotifyProc,VS_ULONG Para);
void SRPAPI **UnRegReMallocCallBack**(void *Object,VS_ObjectReMallocNotifyProc ObjectReMallocNotifyProc ,VS_ULONG Para);
The prototype of callback:
typedef void (SRPAPI *VS_ObjectReMallocNotifyProc)(void *Object,VS_ULONG Para,void *NewObject);
//---NewObject == NULL means free
If application stores object address, but object address may be changed or freed, so it must register callback function to capture the change.

5.20.20 Alloc queue-AllocQueue

OBJECTATTRIBUTEINDEX SRPAPI **AllocQueue**(void *ParentObject,void *Object);
Object can be in which queue of its parent. If fails, then INVALID_OBJECTATTRIBUTEINDEX is returned.

OBJECTATTRIBUTEINDEX SRPAPI **AllocQueueEx**(void *ParentObject,VS_UUID *ClassID);;

5.21 Register object ID change callback

VS_BOOL RegObjectIDChangeNotify(VS_ObjectIDChangeNotifyProc ChangeNotifyProc,VS_ULONG Para);
void UnRegObjectIDChangeNotify(VS_ObjectIDChangeNotifyProc ChangeNotifyProc,VS_ULONG Para);
Function prototype:
typedef void (SRPAPI *VS_ObjectIDChangeNotifyProc)(void *Object,VS_ULONG Para,VS_UUID *NewObjectID);

5.22 Register object free callback

VS_BOOL SRPAPI **RegObjectFreeNotify**(VS_ObjectFreeNotifyProc FreeNotifyProc,VS_ULONG Para);
void SRPAPI **UnRegObjectFreeNotify**(VS_ObjectFreeNotifyProc FreeNotifyProc,VS_ULONG Para);
Function prototype:
typedef void (SRPAPI *VS_ObjectFreeNotifyProc)(void *Object,VS_ULONG Para);

5.23 Variable length string functions

5.23.1 Duplicate variable length string -DupVString

void SRPAPI **DupVString**(VS_VSTRING *InVString,VS_VSTRING *OutVString)
Copy InVString to OutVString.

5.23.2 Get VSTRING buffer size-GetVStringBufSize

VS_ULONG SRPAPI **GetVStringBufSize**(VS_VSTRING *VString);

5.23.3 Expand VSTRING buf size-ExpandVStringBufSize

void SRPAPI **ExpandVStringBufSize**(VS_VSTRING *VString,VS_ULONG Size);
Buffer lenth is the string length +1.

5.23.4 Set variable string—SetVString

void SRPAPI SetVString(VS_VSTRING *Buf,VS_CHAR *Str);

5.23.5 Change to variable string—ToVString

VS_VSTRING *SRPAPI ToVString(VS_CHAR *Str);
The return pointer should not be freed.

5.24 User management-valid at server side.

5.24.1 Create or change user-CreateUser

VS_BOOL SRPAPI **CreateUser**(VS_CHAR *UserName,VS_CHAR *UserPass,VS_UINT8 ReadWriteOrExecute);
If user does not exist, then create new one, or else change the user information.

ReadWriteOrExecute is the combination of the following values:
READ 0x01
WRITE 0x02
EXECUTE 0x04
EXPORTXML 0x08

5.24.2 Delete user-DeleteUser

void SRPAPI **DeleteUser**(VS_CHAR *UserName);
For calling the function, the interface should be get using root user.

5.24.3 Query first user-QueryFirstUser

VS_CHAR *SRPAPI **QueryFirstUser**(VS_QUERYRECORD *QueryRecord,VS_UINT8 *ReadWriteOrExecute);

5.24.4 Query next user -QueryNextUser

VS_CHAR *SRPAPI **QueryNextUser**(VS_QUERYRECORD *QueryRecord,VS_UINT8 *ReadWriteOrExecute);

5.25 Set and get object app class

5.25.1 SetAppClass

void SRPAPI **SetAppClass**(void *Object ,VS_ULONG ClassLayer,class ClassOfSRPObject *SRPObjectClass);
The function should not be called by application. Which is set by WrapObject function of SRPObjectClass.

5.25.2 GetAppClass

class ClassOfSRPObject *SRPAPI **GetAppClass**(void *Object ,VS_ULONG ClassLayer=0xFFFFFFFF)

5.26 Object attribute function and event function

5.26.1 Get attribute number-GetAttributeNumber,GetAttributeSize, GetAttributeName

VS_INT32 SRPAPI **GetAttributeNumber**(void *Object); Return attribute number defined in this object and its class
VS_INT32 SRPAPI GetAttributeSize (void *Object); Return attribute size in this object and its class.
VS_CHAR *SRPAPI **GetAttributeName**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex); Return attribute name in this object or its class.

5.26.2 Get attribute info-GetAttributeInfo, GetAttributeInfoEx

VS_BOOL SRPAPI **GetAttributeInfo**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex,VS_ATTRIBUTEINFO *AttributeInfo);
AttributeIndex starts from 0.
VS_BOOL SRPAPI **GetAttributeInfoEx**(void *Object, VS_CHAR *AttributeName,VS_ATTRIBUTEINFO *AttributeInfo);
Return attribute information of this object and its class.

5.26.3 Get attach attribute number-GetAttachAttributeNumber,GetAttachAttributeSize

VS_INT32 SRPAPI **GetAttachAttributeNumber**(void *Object);
VS_INT32 SRPAPI **GetAttachAttributeSize**(void *Object);

5.26.4 Get attach attribute info-GetAttachAttributeInfo

VS_BOOL SRPAPI **GetAttachAttributeInfo**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex,VS_ATTRIBUTEINFO *AttributeInfo);
AttributeIndex starts from 0

5.26.5 Fill object attribute combobox item-GetComboBoxItem

VS_BOOL SRPAPI **GetComboBoxItem**(VS_UINT8 *ComboBoxID, VS_COMBOBOXITEM *ComboBoxItemBuf);

5.26.6 Get object function number-GetFunctionNumber

VS_INT32 SRPAPI **GetFunctionNumber**(void *Object); return function number defined in this object.

5.26.7 Get object function info-GetFunctionInfo

VS_BOOL SRPAPI **GetFunctionInfo**(void *Object,VS_INT32 FunctionIndex,VS_FUNCTIONINFO *FunctionInfo);
returns function information defined in this object.

5.26.8 Get output event number -GetOutEventNumber

VS_INT32 SRPAPI **GetOutEventNumber**(void *Object); return number of events defined in this object.

5.26.9 Get output event info-GetOutEventInfo

VS_BOOL SRPAPI **GetOutEventInfo** (void *Object,VS_INT32 OutEventIndex,VS_OUTEVENTINFO *OutEventInfo);
returns event infomation defined in this object.

5.27 Index and search function

5.27.1 create an Indexer(CreateIndex_Nor,CreateIndexCmp_Nor ,CreateIDIndex_Nor,CreateIDIndexEx_Nor)

void ***CreateIndex_Nor**(VS_INT32 KeyNumber, VS_UINT16 HashTableBits); KeyNumber represents number of the key, which supports 1,2,3.
void *SRPAPI **CreateIndexCmp**_Nor(VS_INT32 KeyNumber, VS_UINT16 HashTableBits, VS_IndexCompareProc CompareProc);
void ***CreateIDIndex_Nor**(VS_UINT16 HashTableBits);
void ***CreateIDIndexEx_Nor**(VS_UINT16 HashTableBits); uses UUID + VS_ULONG as index
typedef VS_INT32 (SRPAPI *VS_IndexCompareProc)(void *NodeBuf1,void *NodeBuf2); -1 Buf1 < Buf2, 0 Buf1 == Buf2, 1 Buf1 > Buf2 does not support _F series searching functions.

HashTableBits represents Hash table size,2^(HashTableBits) number, which is used to speed up the search process; it may be set to 0. Hash table will occupy extra memory about 2^(HashTableBits)*4 bytes.

5.27.2 create a indexer(CreateIndex_Dbg,CreateIndexCmp_Dbg ,CreateIDIndex_Dbg,CreateIDIndexEx_Dbg)

void ***CreateIndex_Dbg***(VS_INT32 KeyNumber, VS_UINT16 HashTableBits, VS_CHAR *FileName,VS_INT32 LineNumber); KeyNumber represents number of the key, which supports 1,2,3.
void ***CreateIDIndex_Dbg*** (VS_UINT16 HashTableBits,VS_CHAR *FileName,VS_INT32 LineNumber);
void ***CreateIDIndexEx_Dbg*** (VS_UINT16 HashTableBits,VS_CHAR *FileName,VS_INT32 LineNumber);
uses UUID + VS_ULONG as index
void *SRPAPI ***CreateIndexCmp***_Dbg(VS_INT32 KeyNumber, VS_UINT16 HashTableBits, VS_IndexCompareProc CompareProc, VS_CHAR *FileName,VS_INT32 LineNumber);
typedef VS_INT32 (SRPAPI *VS_IndexCompareProc)(void *NodeBuf1,void *NodeBuf2); -1 Buf1 < Buf2, 0 Buf1 == Buf2, 1 Buf1 > Buf2 does not support _F series searching functions.

HashTableBits represents Hash table size,2^(HashTableBits) number, which is used to speed up the search process; it may be set to 0. Hash table will occupy extra memory about 2^(HashTableBits)*4 bytes.

5.27.3 Add,delete or find an index(one key)(InsertOneKey,FindOneKey,DelOneKey)

void ***InsertOneKey***(void *IndexContext,VS_ULONG MainKey,VS_INT8 *Buf);
VS_INT8 ****FindOneKey***(void *IndexContext,VS_ULONG MainKey);
VS_INT8 ****DelOneKey***(void *IndexContext,VS_ULONG MainKey);
VS_INT8 ****QueryFirstOneKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); Small to large order
VS_INT8 ****QueryNextOneKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); Small to large order
VS_INT8 ****QueryFirstOneKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); large to small order
VS_INT8 ****QueryNextOneKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey); large to small order
Context is Indexer, MainKey is the key, Buf is the buffer of the index points to.

5.27.4 Add,delete or find an index(two keywords)(InsertTwoKey,FindTwoKey,DelTwoKey)

void ***InsertTwoKey***(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_INT8 *Buf);
VS_INT8 ****FindTwoKey***(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey);
VS_INT8 ****DelTwoKey***(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey);
VS_INT8 ****QueryFirstTwoKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ****QueryNextTwoKey***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ****QueryFirstTwoKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); large to small order
VS_INT8 ****QueryNextTwoKeyA***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey); large to small order
VS_INT8 ****QueryFirstTwoKey_F***(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ****QueryNextTwoKey_F*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); Small to large order
VS_INT8 ****QueryFirstTwoKeyA_F*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); large to small order
VS_INT8 ****QueryNextTwoKeyA_F*** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey); large to small order

5.27.5 Add,delete or find an index(three keywords)(InsertThreeKey,FindThreeKey,DelThreeKey)

void **InsertThreeKey**(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG ThirdKey,VS_INT8 *Buf);
VS_INT8 ***FindThreeKey**(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG ThirdKey);
VS_INT8 ***DelThreeKey**(void *IndexContext,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG ThirdKey);
VS_INT8 ***QueryFirstThreeKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryNextThreeKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryFirstThreeKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryNextThreeKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG *MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryFirstThreeKey_F**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryNextThreeKey_F** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryFirstThreeKeyA_F** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryNextThreeKeyA_F** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG *SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryFirstThreeKey_S**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryNextThreeKey_S** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); Small to large order
VS_INT8 ***QueryFirstThreeKeyA_S** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); large to small order
VS_INT8 ***QueryNextThreeKeyA_S** (void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_ULONG MainKey,VS_ULONG SecondKey,VS_ULONG *ThirdKey); large to small order

5.27.6 Add,delete or find an index(UUID keywords)(InsertIDKey,FindIDKey,DelIDKey,InsertIDKeyEx,FindIDKeyEx,DelIDKeyEx).

void **InsertIDKey**(void *IndexContext,VS_UUID *UuidKey,VS_INT8 *Buf);
VS_INT8 ***FindIDKey**(void *IndexContext,VS_UUID *UuidKey);
VS_INT8 ***DelIDKey**(void *IndexContext,VS_UUID *UuidKey);
VS_INT8 ***QueryFirstIDKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); Small to large order
VS_INT8 ***QueryNextIDKey**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); Small to large order
VS_INT8 ***QueryFirstIDKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); large to small order
VS_INT8 ***QueryNextIDKeyA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey); large to small order

void **InsertIDKeyEx**(void *IndexContext,VS_UUID *UuidKey,VS_ULONG ExKey,VS_INT8 *Buf);
VS_INT8 ***FindIDKeyEx**(void *IndexContext,VS_UUID *UuidKey,VS_ULONG ExKey);
VS_INT8 ***DelIDKeyEx**(void *IndexContext,VS_UUID *UuidKey,VS_ULONG ExKey);

VS_INT8 ***QueryFirstIDKeyEx**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order
VS_INT8 ***QueryNextIDKeyEx**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order
VS_INT8 ***QueryFirstIDKeyEx_F**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order,use special UuidKey value
VS_INT8 ***QueryNextIDKeyEx_F**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); Small to large order,use special UuidKey value
VS_INT8 ***QueryFirstIDKeyExA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); large to small order
VS_INT8 ***QueryNextIDKeyExA**(void *IndexContext,VS_QUERYRECORD *QueryRecord,VS_UUID *UuidKey,VS_ULONG *ExKey); large to small order

5.27.7 get index number-GetKeyNumber

VS_INT32 SRPAPI GetKeyNumber(void *IndexContext)

5.27.8 delete all index-DelAllKey

void **DelAllKey**(void *IndexContext);

5.27.9 delete indexer-DestoryIndex

void **DestoryIndex**(void *IndexContext);

5.27.10 get Hash value-GetHashValue

VS_ULONG SRPAPI GetHashValue(void *Key,VS_ULONG Length,VS_ULONG InitValue);

5.28 Memory manager function

5.28.1 create memory manager-CreateMemory_Nor

void ***CreateMemory_Nor**(VS_INT32 ItemSize);
ItemSize is size of each memory block

5.28.2 create memory manager -CreateMemory_Dbg

void ***CreateMemory_Dbg**(VS_INT32 ItemSize, VS_CHAR *FileName,VS_INT32 LineNumber);
ItemSize is size of each memory block

5.28.3 alloc memory block-GetMemoryPtr_Nor

void ***GetMemoryPtr_Nor**(void *MemoryContext);
MemoryContext is memory manager.

5.28.4 alloc memory block -GetMemoryPtr_Dbg

void ***GetMemoryPtr_Dbg**(void *MemoryContext, VS_CHAR *FileName,VS_INT32 LineNumber);
MemoryContext is memory manager.

5.28.5 query first memory block-QueryFirstMemoryPtr

void ***QueryFirstMemoryPtr**(void *MemoryContext,VS_QUERYRECORD *QueryRecord);

5.28.6 query next memory block -QueryNextMemoryPtr

void ***QueryNextMemoryPtr**(void *MemoryContext,VS_QUERYRECORD *QueryRecord);

5.28.7 free memory block-FreeMemoryPtr

void **FreeMemoryPtr**(void *MemoryContext,void *Ptr);

5.28.8 clear memory block-ClearMemory

void SRPAPI **ClearMemory**(void *MemoryContext);
Free all memory blocks in the memory manager.

5.28.9 delete memory manager-DestoryMemory

void **DestoryMemory**(void *MemoryContext);

5.28.10 alloc memory-Malloc_Nor

void ***Malloc_Nor**(VS_INT32 MemorySize);

5.28.11 alloc memory-Malloc_Dbg

void ***Malloc_Dbg**(VS_INT32 MemorySize VS_CHAR *FileName,VS_INT32 LineNumber);

5.28.12 free memory-Free

void **Free**(void *MemoryPtr);

5.28.13 get memory used-GetMemoryUsed

void SRPAPI **GetMemoryUsed**(VS_ULONG *KernelAllocSize,VS_ULONG *DataAllocSize,VS_ULONG *AppAllocSize, VS_ULONG *ScriptMemoryUsed);
KernelAllocSize: memory of core.
DataAllocSize: memory of static data
AppAllocSize: memory of application allocated using interface function Malloc.
ScriptMemoryUsed: memory of lua script.

5.29 Event processing function

5.29.1 Order of event process

1. event function define in Lua
2. event function defined in the name script
3. event function defined in C

```
function EventName( self, Event )

   Return true;
end

VS_INT32 OnEvent(VS_ULONG FunctionChoice,void *EventPara)
{
   VS_EVENTPARAM *LocalEventParaPtr;

   LocalEventParaPtr = (VS_EVENTPARAM *)EventPara;
}
```

5.29.2 Register system event function-InjectSysEventFunction[reserved for cle]

void **InjectSysEventFunction**(VS_ULONG SysEventID, VSSystemEvent_EventProc ProcessFunctionProc);
SysEventID is system event ID. For example:SysEventID = VSEVENT_SYSTEMEVENT_ONCREATE;
If there is return value, and the event need not continue to be processed, then handler should return VSEVENTMANAGER_STOP.

5.29.3 Unregister system event function -RejectSysEventFunction[reserved for cle]

When module is unloaded, if it has registered the function, it must be unregistered
void **RejectSysEventFunction**(VS_ULONG SysEventID, VSSystemEvent_EventProc ProcessFunctionProc);

5.29.4 Get event response buffer-GetResponseBuf

If the event wants to return value, it should alloc response buffer, fill the buffer, and use *AttachResponseBuf function to pass the buffer to event*
VS_EVENTPARAM_RUNPARAM *GetResponseBuf();

5.29.5 Get event request buffer-GetRequestBuf

VS_EVENTPARAM_RUNPARAM * GetRequestBuf ();

5.29.6 Get system event ID-GetSysEventID

VS_ULONG **GetSysEventID**(VS_EVENTPARAM *EventParam);
//---If returns 0, then the event is not system event

5.29.7 Free event response buffer-FreeResponseBuf

If alloc response buffer, but not attach to event, the handler should call this function to free the response buffer.
void **FreeResponseBuf**(VS_EVENTPARAM_RUNPARAM *ResponseParam);

5.29.8 Free request buffer-FreeRequestBuf

void **FreeRequestBuf** (VS_EVENTPARAM_RUNPARAM * RequestParam);

5.29.9 Attach response buffer to event-AttachResponseBuf

void **AttachResponseBuf**(VS_EVENTPARAM *EventParam,VS_EVENTPARAM_RUNPARAM *ResponseParam);
The response buffer will be freed by CLE.

5.29.10 Process event-ProcessEvent_Nor, ProcessEvent_Dbg

VS_EVENTPARAM_RUNPARAM ***ProcessEvent_Nor**(VS_UUID *EventID,void *SrcObject,void *DesObject,VS_EVENTPARAM_RUNPARAM *RequestParam);
VS_EVENTPARAM_RUNPARAM ***ProcessEvent_Dbg**(VS_UUID *EventID,void *SrcObject,void *DesObject,VS_EVENTPARAM_RUNPARAM *RequestParam);

EventID : Event ID.
SrcObject : Source object which trigger the event
DesObject : should be NULL, reserved.
RequestParam : Event request parameter, which is allocated using GetRequestBuf

5.29.11 Defer process event-PostProcessEvent_Nor, PostProcessEvent_Dbg

VS_EVENTPARAM_RUNPARAM ***PostProcessEvent_Nor**(VS_UUID *EventID,void *SrcObject,void *DesObject,VS_EVENTPARAM_RUNPARAM *RequestParam);
VS_EVENTPARAM_RUNPARAM ***PostProcessEvent_Dbg**(VS_UUID *EventID,void *SrcObject,void *DesObject,VS_EVENTPARAM_RUNPARAM *RequestParam);

5.29.12 Call parent event handler-ProcessParentEvent

VS_EVENTPARAM_RUNPARAM *ProcessParentEvent(VS_EVENTPARAM *EventParam);

5.30 Dynamic event register

5.30.1 Register dynamic event function-RegEventFunction

VS_BOOL **RegEventFunction**(void *SrcObject, VS_UUID *EventID, void *Object, void *FuncAddr,VS_ULONG Para);
Object is as the DesObject of EventPara. SrcObject is as the SrcObjet of EventPara, which may be NULL,in this case, event with EventID created by any object will be sent to the callback .
Event created by SrcObject self and its instance, will be sent to the callback.
Object can not be NULL.

Prototype of function:
typedef VS_INT32 (SRPAPI * VSSystemEvent_EventProc)(VS_ULONG FunctionChoice,void *EventPara);
Object may be NULL. FunctionChoice is set to Para.

5.30.2 Unregister dynamic event function-UnRegEventFunction

void **UnRegEventFunction**(void *SrcObject, VS_UUID *EventID,void *Object, void *FuncAddr); 对象 Object can not be NULL.

5.31 Register system event function

VS_BOOL SRPAPI **RegSysEventFunction**(void *Object,VS_ULONG SysEventID, void *FuncAddr,VS_ULONG Para);
void SRPAPI **UnRegSysEventFunction**(void *Object,VS_ULONG SysEventID, void *FuncAddr,VS_ULONG Para);

Function prototype :
typedef VS_INT32 (SRPAPI * VSSystemEvent_EventProc)(VS_ULONG FunctionChoice,void *EventPara);
FunctionChoice is set to Para.

System event list:
VSEVENT_SYSTEMEVENT_ONATTRIBUTEBEFORECHANGE
VSEVENT_SYSTEMEVENT_ONFIRSTCREATE
VSEVENT_SYSTEMEVENT_ONMALLOC
VSEVENT_SYSTEMEVENT_ONFREE
VSEVENT_SYSTEMEVENT_ONCREATE
VSEVENT_SYSTEMEVENT_ONDESTROY
VSEVENT_SYSTEMEVENT_ONBEFOREFIRSTCREATE
VSEVENT_SYSTEMEVENT_ONCREATECHILD
VSEVENT_SYSTEMEVENT_ONDESTROYCHILD
VSEVENT_SYSTEMEVENT_ONACTIVATING
VSEVENT_SYSTEMEVENT_ONDEACTIVATING
VSEVENT_SYSTEMEVENT_ONACTIVATE
VSEVENT_SYSTEMEVENT_ONDEACTIVATE
VSEVENT_SYSTEMEVENT_ONACTIVATECHILD
VSEVENT_SYSTEMEVENT_ONDEACTIVATECHILD
VSEVENT_SYSTEMEVENT_ONATTRIBUTECHANGE
VSEVENT_SYSTEMEVENT_ONPARENTBEFORECHANGE
VSEVENT_SYSTEMEVENT_ONPARENTCHANGE
VSEVENT_SYSTEMEVENT_ONSTATICCHANGE
VSEVENT_SYSTEMEVENT_ONSCRIPTCHANGE
VSEVENT_SYSTEMEVENT_ONBECOMESYNC
VSEVENT_SYSTEMEVENT_ONSYNCGROUPCHANGE
VSEVENT_SYSTEMEVENT_ONACTIVESETCHANGE
VSEVENT_SYSTEMEVENT_ONCHILDSYNCGROUPCHANGE
VSEVENT_SYSTEMEVENT_ONLOADMASK
VSEVENT_SYSTEMEVENT_ONLOADFINISH

5.32 Wait event

5.32.1 WaitEvent

VS_BOOL SRPAPI WaitEvent(void *SrcObject,VS_UUID *EventID, void *Object, void *FuncAddr,VS_ULONG Para,VS_BOOL AutoDelete);
If AutoDelete is VS_TRUE, then the wait will be deleted after event is triggered.
SrcObject may be NULL.
Callback prototype:

typedef void (SRPAPI *VS_WaitEventCallBackProc)(void *SrcObject, void *EventParam, void *Object,VS_ULONG Para);

5.32.2 UnWaitEvent

void SRPAPI UnWaitEvent(void *SrcObject,VS_UUID *EventID, void *Object, void *FuncAddr,VS_ULONG Para);
SrcObject may be NULL. Callback prototype:
typedef void (SRPAPI *VS_WaitEventCallBackProc)(void *SrcObject, void *EventParam, void *Object,VS_ULONG Para);

VS_INT32 SRPAPI LuaPushEventPara(void *EventParam)
Push event request param to Lua stack, and return parameter number. The function will trigger object's event VSEVENT_SYSTEMEVENT_ONVSTOSCRIPTINPUTPARA.

5.33 Object activate function

5.33.1 Set object active command-ActiveCmd

VS_BOOL **ActiveCmd** (void *Object,VS_UINT8 ActiveCmd);
Valid for local object at client side.
Valid for local and global object at server side.
For VSACTIVE_FOLLOW, if parent is service item, then the object will be activated. Otherwise, If parent object is active, the object will be activated, or else, wait for parent to be activated.

ActiveCmd:
#define VSACTIVE_ALONE 0 //---activate using interface function
#define VSACTIVE_FOLLOW 1 //---activate with parent.
#define VSACTIVE_ACTIVE 2 //---If service is active, then the object will be activated
#define VSACTIVE_DEACTIVE 3 //---If service is active, then the object will be deactivated

5.33.2 Get object active command-GetActiveCmd

VS_UINT8 SRPAPI GetActiveCmd(void *Object);

5.33.3 Active object at client side-ActiveClient

VS_BOOL SRPAPI ActiveClient(VS_ULONG ClientID,void *Object);
If ClientID is 0, then the function activates object at all clients

5.33.4 Deactive object at client side -DeactiveClient

void SRPAPI DeactiveClient(VS_ULONG ClientID,void *Object);
If ClientID is 0, then the function deactivates object at all clients

5.33.5 active object-Active

VS_BOOL Active(void *Object);

5.33.6 deactive object-Deactive

void **Deactive**(void *Object);

5.33.7 Deactive all objects-DeactiveAll

void **DeactiveAll**();

5.33.8 Whether object is active-IsActive

VS_BOOL IsActive(void *Object);

5.33.9 Query first active instance-QueryFirstActiveInst

void ***QueryFirstActiveInst***(VS_QUERYRECORD *QueryRecord,VS_UUID *ObjectClassID);
Include instance of instance
The function will traverse all active instances.

5.33.10 Query next active instance-QueryNextActiveInst

void ***QueryNextActiveInst***(VS_QUERYRECORD *QueryRecord,VS_UUID *ObjectClassID);
In the traverse procedure, if new object is activated or deactivated, then the procedure will be restart..

5.34 Client representative object

5.34.1 Set representative object -SetClientObject

VS_BOOL SRPAPI **SetClientObject**(VS_ULONG ClientID,void *Object);
Called at server side., and object should be global or client dynamic object.

5.34.2 Get representative object -GetClientObject

void *SRPAPI **GetClientObject**();
Called at client side.

5.35 Service item and active set manager

5.35.1 Create service item-CreateSysRootItem

VS_BOOL **CreateSysRootItem**(VS_CHAR *SystemRootItemName , VS_CHAR *DependSysRootItem,VS_UUID *SystemRootItemID, VS_UUID *SystemRootItemIDEx);
The function can only be called at server side. DependSysRootItem may be set to NULL. If SystemRootItemID is valid, then SystemRootItemIDEx should set to a valid value too.

5.35.2 Active all service item-ActiveAllSysRootItem

void **ActiveAllSysRootItem** ( );
Called at server

5.35.3 Active service item-ActiveSysRootItem

void **ActiveSysRootItem**(VS_CHAR *SystemRootItemName);

5.35.4 Deactive service item-DeactiveSysRootItem

void **DeactiveSysRootItem** (VS_CHAR *SystemRootItemName);

5.35.5 Active service item at client-ActiveCSysRootItem Called at server

void **ActiveCSysRootItem**(VS_ULONG ClientID, VS_CHAR *SystemRootItemName);
ClientID is the ID of client state machine. If application knows state machine pointer, the ID may be get with function **GetMachineID.**

5.35.6 Deactive service item at client -DeactiveCSysRootItem Called at server

void **DeactiveCSysRootItem** (VS_ULONG ClientID, VS_CHAR *SystemRootItemName);

5.35.7 Get service item pointer-GetSysRootItem

void ***GetSysRootItem**(VS_CHAR *SystemRootItemName); service item must be activated
void ***GetSysRootItemEx**(VS_UUID *SystemRootItemID); service item must be activated

5.35.8 Get service item name-GetSysRootItemName

VS_CHAR * GetSysRootItemName (void *SystemRootItem);

5.35.9 Get service item active set-GetSysRootItemActiveSet

void **GetSysRootItemActiveSet**(void *SystemRootItem,VS_ACTIVESETITEM *ActiveSetPtr);
SystemRootItem is pointer of service item.
When the function is called at debug, the return value is always {1,0xFFFFFFFF,{}}.

5.35.10 Set service item active set-SetSysRootItemActiveSet

void **SetSysRootItemActiveSet**(void *SystemRootItem,VS_ACTIVESETITEM *ActiveSetItem);

5.35.11 Set service item active set for client at server side-SetCSysRootItemActiveSet

void **SetCSysRootItemActiveSet**(VS_ULONG ClientID,void *SystemRootItem,VS_ACTIVESETITEM *ActiveSetItem);
The function is valid only at server side, and ClientID must be valid value. If the service item is not active at client, it will be activated automaticly.

5.35.12 Get first service item name-QueryFirstSysRootItem

VS_CHAR *QueryFirstSysRootItem( );

5.35.13 Get next service item name-QueryNextSysRootItem

VS_CHAR *QueryNextSysRootItem( );

5.35.14 Get first object in service item sync group-QueryFirstGroupObject

void ***QueryFirstGroupObject**(void *SystemRootItem, VS_SYNCGROUP GroupIndex,VS_ULONG *QueryGroupContext);
GroupIndex should not be set to 0;

5.35.15 Get next object in service item sync group-QueryNextGroupObject

void ***QueryNextGroupObject**(VS_ULONG *QueryGroupContext);

5.35.16 Get first object in service item-QueryFirstSysRootItemChild

void *SRPAPI **QueryFirstSysRootItemChild**(void *SystemRootItem);

5.35.17 Register service item sync callback-RegClientSysRootItemToSyncFunc

VS_BOOL SRPAPI RegClientSysRootItemToSyncFunc(void *SystemRootItem,VS_ClientSysRootItemChangeToSyncProc FuncPtr, VS_ULONG Para);
When client activates service item, and the service item changes to sync status, then callback on server side will be triggered.
Function prototype:
typedef void (SRPAPI *VS_ClientSysRootItemChangeToSyncProc)(void *SysRootItem,VS_ULONG ClientID, VS_ULONG SyncGroupIndex,VS_ULONG Para);

5.35.18 Unregister service item sync callback -UnRegClientSysRootItemToSyncFunc

void SRPAPI UnRegClientSysRootItemToSyncFunc(void *SystemRootItem,VS_ClientSysRootItemChangeToSyncProc FuncPtr, VS_ULONG Para);

5.36 Edit function[reserved for debug server]

Object edit function is valid only at debug side.

5.36.1 Extern start edit object-InitEdit

void **InitEdit**(void *ClassObject,void *Object);

5.36.2 Extern term edit object -TermEdit

void **TermEdit**(void *ClassObject,void *Object);

5.36.3 Get object edit flag-GetEditMode

VS_BOOL SRPAPI **GetEditMode**(void *Object);
Returns true, if object is edited.
If any parent of object is edited, the function returns true.

5.36.4 Set object edit flag-SetEditMode

void SRPAPI **SetEditMode**(void *Object,VS_BOOL EditFlag);

5.36.5 Send edit change to server-EditCommit

void SRPAPI **EditCommit**();
All changes at debug side, will be sync to server.

5.36.6 Extern select object-EditSelect

void **EditSelect**(void *Object);
Object is to be selected.

5.36.7 Extern change object attribute-EditChange

void **EditChange**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex, VS_INT8 *NewValue);
For VSTRING, input is normal string, which may be set to NULL.

At server side, this function is same as ChangeObject

5.36.8 Extern mark object attribute change-EditMarkChange

void SRPAPI **EditMarkChange**(void *Object,OBJECTATTRIBUTEINDEX AttributeIndex);
Valid for global object, indicates object attribute is changed, and then application can use function EditCommit to sync the changes to server.

5.36.9 Set object save flag(EditSetSaveFlag)

void SRPAPI **EditSetSaveFlag**(void *Object,VS_UINT8 SaveFlag);
At server side, this function is same as SetSaveFlag

5.36.10 Change object nameEditSetName)

void **EditSetName**(void *Object, VS_CHAR *Name);
At server side, this function is same as SetName

5.36.11 Extern request to fill object init parameter-FillAttachBuf

void **FillAttachBuf**(VS_UUID *ObjectClassID,VS_INT8 *AttachBuf,VS_INT32 AttachBufSize);

5.36.12 Whether object editor exist -IsEditProcExist

VS_BOOL **IsEditProcExist**(void *Object);
Object and its class has hook editor or not.

5.36.13 Call object editor-TriggerEditProc

void **TriggerEditProc**(void *ClassObject,void *Object,VS_ULONG AppCode,VS_ULONG AppCode1);
Using ClassObject's editor to edit Object; Object is the instance of ClassObject. AppCode, AppCode1 is init parameter of editor, which is defined by editor.

5.36.14 Extern delete object-EditDelete

void **EditDelete**(void *Object);
At server side, call this function is same as FreeObject

5.36.15 Extern create object(static object)-EditCreate[Debug side can not create global dynamic object]

void ***EditCreate**(VS_UUID *ObjectClassID,VS_UUID *ParentObjectID,OBJECTATTRIBUTEINDEX AttributeIndex,VS_ULONG OrderIndex, VS_INT32 AttachBufSize,void *AttachBuf);
AttachBuf is object init parameter, may be set to NULL. The init parameter is defined by attach attribute of object
At server side, the function is same as MallocObject.

5.36.16 Extern create object with ID(static object)-EditCreate[Debug side can not create global dynamic object]

void ***EditCreateEx**(VS_UUID *ObjectClassID,VS_UUID *ObjectID,VS_UUID *ParentObjectID,OBJECTATTRIBUTEINDEX AttributeIndex,VS_ULONG OrderIndex, VS_INT32 AttachBufSize,void *AttachBuf);
AttachBuf is object init parameter, may be set to NULL. The init parameter is defined by attach attribute of object
At server side, the function is same as MallocObjectEx.

5.36.17 Extern chage parent object-EditChangeParent

void **EditChangeParent**(void *Object,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex);
At server side, the function is same as ChangeParent.

5.36.18 Extern change sync group-EditChangeSyncGroup

void **EditChangeSyncGroup**(void *Object, VS_SYNCGROUP GroupIndex);
At server side, the function is same as SetSyncGroup.
It sync group of parent object is set, then the sync group of child object can not be set

5.36.19 Exern get queue attribute by ClassID-EditGetClassID

Void **EditGetClassID**(VS_UUID *ObjectID,OBJECTATTRIBUTEINDEX AttributeIndex, VS_UUID *UuidPtr);

5.36.20 Exern get instance by ClassID-EditGetInstID

Void EditGetInstID(VS_UUID *ObjectClassID, VS_UUID *UuidPtr);
// Select instance of class

5.36.21 Extern set status of editor-EditSetWndStatus

void **EditSetWndStatus**(VS_BOOL Normal);
// ==true normal display; ==false special display, interpreted by editor

5.36.22 Update object name script - EditUpdateObjectScript

VS_BOOL SRPAPI **EditUpdateObjectScript**(void *Object, VS_CHAR *ScriptName, VS_CHAR *ScriptBuf)

5.36.23 Update object name script -EditUpdateObjectScriptEx

VS_BOOL SRPAPI **EditUpdateObjectScriptEx**(void *Object, VS_CHAR *ScriptName, VS_CHAR *FileName)

5.37 Get parapkg interface

5.37.1 Get parapkg interface-GetParaPkgInterface

class ClassOfSRPParaPackageInterface ***GetParaPkgInterface**();

5.38 Service redirect

5.38.1 Redirect

void **Redirect**(VS_ULONG ClientID, VS_CHAR *DesServerInterface,VS_CHAR *DesServerName, VS_UINT16 DesServerPortNumber, class ClassOfSRPParaPackageInterface * ParaPkg, VS_RedirectCallBackProc RedirectCallBackProc,VS_ULONG Para);
The function should be called at server side. ParaPkg is parameter, which should be freed by the caller. ParaPkg may also be set to NULL.
If succeed, server first call the callback function, and then close the connection with the corresponding client DesServerInterface is link-layer interface, which may be set to NULL

5.39 Client operation callback

VS_BOOL **RegClientOpFunction**(VS_ClientOperationCallBackProc ClientOperationCallBackProc ,VS_ULONG Para);

void **UnRegClientOpFunction**(VS_ClientOperationCallBackProc ClientOperationCallBackProc ,VS_ULONG Para);

5.40 State machine function

5.40.1 Get server state machine-GetServiceMachine

void ***GetServiceMachine**();
The function should be called at client side.

5.40.2 Release state machine-DelMachine

void **DelMachine**(void *Machine);
Delete state machine will close the connection to server.

5.40.3 Get machine associated buffer-GetMachineAttachBuf

VS_INT8 *GetMachineAttachBuf(void *Machine);
Associated buffer is defined and used by APP.

5.40.4 Set machine associated buffer-SetMachineAttachBuf

void **SetMachineAttachBuf**(void *Machine,VS_INT8 *AppBuf);

5.40.5 Get state machine ID-GetMachineID

VS_ULONG GetMachineID(void *Machine); MachineID 也即 ClientID.

5.40.6 Find state machine by ID-FindMachine

void ***FindMachine**(VS_ULONG MachineID);

5.41 Client management function(valid at server side)

5.41.1 Register and unregister client login function-RegClientMachineProcess,UnRegClientMachineProcess

VS_BOOL SRPAPI **RegClientMachineProcess**(void *CallBackPtr, void *Object, VS_ULONG Para);
void SRPAPI **UnRegClientMachineProcess**(void *CallBackPtr, void *Object);
Prototype of callback:
typedef VS_BOOL (SRPAPI *VS_ClientMachineProcessProc)(void *Machine,void *Object,VS_ULONG Para,VS_ULONG uMes,VS_UUID *SrcServiceID, VS_ULONG SrcServiceAdd,VS_UINT16 SrcServicePort,class ClassOfSRPParaPackageInterface *ParaPkg ,VS_CHAR *UserName,VS_CHAR *UserPassword);
Machine: is client machine, Para is the value which is set at register.
SrcServiceID is source service ID, If is not redirect from other server, this parameter is invalid UUID.
ParaPkg is parameter when client setup the connection.
Object may be set to NULL.

uMes meaning:
VS_CLIENT_LOGIN : client login ,SrcServiceID,SrcServiceAdd,SrcServicePort,ParaPkg are valid. In this case, application should call **ClientInitOk** or **ClientInitError to accept client or deny client. Otherwise, client login process will suspend to wait one of the two functions is called.**
If the function returns VS_TRUE, indicates client is processed. If all registered functions returns VS_FALSE, then client is permit to login by default.
VS_CLIENT_LOADSERVICEOK: Client has loaded service
VS_CLIENT_LOGOUT: Client logout.

5.41.2 Client init failed-ClientInitError

void **ClientInitError**(void *Machine);

5.41.3 Client init success-ClientInitOk

void **ClientInitOk**(void *Machine ,VS_BOOL ReSyncFlag, VS_CHAR *TermOldScript, VS_CHAR *InitNewScript,VS_ULONG ClientPrivateTag,VS_ULONG ClientOPPermission,VS_ULONG ClientUploadMaxSize);

If ReSyncFlag == VS_TRUE, then client will be forced to resychronize for the same service, otherwise not.
TermOldScript: is the lua script called before service is loaded, should use carefully. If no active serice exists at client side, then it is ignored. lua global variable "_gService" represents current service.
InitNewScript: is the lua script called before service is loaded; lua global variable "_gService" represents current service.
TermOldScript + InitNewScript: total length should < 32Kbytes.
If is redirected from the same service, then only InitNewScript is valid.

when InitNewScript is executed, the service is not synchronized. Therefore application should not call functions of the object in the service.
ClientOPPermission may be combination of following values:
#define VSCLIENTOP_CREATE ((VS_ULONG)0x00000001)
#define VSCLIENTOP_DELETE ((VS_ULONG)0x00000002)
#define VSCLIENTOP_CHANGE ((VS_ULONG)0x00000004)
ClientUploadMaxSize: maximum size permitted to upload at client side, which unit is bytes. Client side uses SetStaticData to upload data.

5.41.4 Delete client-DeleteClient

void **DeleteClient**(void *Machine);

5.41.5 Get client information and number-GetClientInfo,QueryFirstClientInfo,QueryNextClientInfo,GetClientNumber

void **GetClientInfo**(void *Machine,VS_CLIENTINFO *ClientInfo); //---Valid at server
VS_BOOL **QueryFirstClientInfo**(VS_CLIENTINFO *ClientInfo); //--- Valid at server
VS_BOOL **QueryNextClientInfo**(VS_CLIENTINFO *ClientInfo); //--- Valid at server

5.41.6 VS_INT32 GetClientNumber(); //---Valid at server

5.42 Client Qos management(GetClientQos,SetClientQos,GetServiceQos)

//---Client Qos management
void **GetClientQos**(void *Machine,VS_CLIENTQOS *QosBuf); //--Valid at server and client
void **SetClientQos**(void *Machine,VS_CLIENTQOS *QosBuf); //--Valid at server only
void **GetServiceQos**(VS_CLIENTQOS *QosBuf); //-- Valid at server and client

5.43 File upload and download

File and service in the same directory.

5.43.1 Download file-DownLoad[reserved]

VS_BOOL **DownLoad**(void *AttachObject, VS_CHAR *ServerPath,VS_CHAR *ClientPath, VS_CHAR *FileName, VS_FileUpDownLoadCallBackProc CallBackProc , void *Object, VS_ULONG Para);
//---Called at client and debug side. CallBackProc may be set to NULL, In this case, callback registered with **RegFileCallBack will be called**
Object may be set to NULL
void *AttachObject may be set to NULL
ServerPath, ClientPath may be set to NULL, if it is relative path, then relative to the path of service.

5.43.2 Upload file-UpLoad[reserved]

VS_BOOL **UpLoad**(void *AttachObject, VS_CHAR *ServerPath,VS_CHAR *ClientPath, VS_CHAR *FileName, VS_FileUpDownLoadCallBackProc CallBackProc , void *Object, VS_ULONG Para);
//---Called at client and debug side. CallBackProc may be set to NULL, In this case, callback registered with **RegFileCallBack will be called**
Object may be set to NULL
void *AttachObject may be set to NULL
ServerPath, ClientPath may be set to NULL, if it is relative path, then relative to the path of service.

5.43.3 Get file upload or download info -GetFileInfo

void **GetFileInfo**(VS_UPDOWNFILEINFO *InfoPtr);

5.43.4 Register callback(file and static data)-RegFileCallBack

VS_BOOL **RegFileCallBack**(VS_FileUpDownLoadCallBackProc CallBackProc , void *Object, VS_ULONG Para);
The function is valid at client or debug side. The callback will be called when download or upload happens.
The callback may be registered more than one.
Object may be set to NULL.

5.43.5 Unregister callback -UnRegFileCallBack

void **UnRegFileCallBack**(VS_FileUpDownLoadCallBackProc CallBackProc , void *Object, VS_ULONG Para);
The function is valid at client or debug side.

5.43.6 Get file upload or download status-GetFileStatus

VS_INT32 **GetFileStatus**(VS_CHAR *FileName, VS_UPDOWNFILEMSG *FileInfo);

The function is valid at client or debug side, and returns VSFILE_STATUSDOWN,VSFILE_STATUSUP or VSFILE_STATUSIDLE.

5.43.7 Get static data upload or download status -GetDataStatus

VS_INT32 SRPAPI **GetDataStatus**(void *Object,VS_ULONG UniqueDataUnitID, VS_UPDOWNFILEMSG *FileInfo);
The function is valid at client or debug side,returns VSFILE_STATUSDOWN,VSFILE_STATUSUP or VSFILE_STATUSIDLE.

5.43.8 Get first file or static data being downloaded-QueryFirstDown

VS_BOOL SRPAPI **QueryFirstDown**(VS_QUERYRECORD *QueryRecord, VS_UPDOWNFILEMSG *FileInfo);
The function is valid at client or debug side

5.43.9 Get next file or static data being downloaded -QueryNextDown

VS_BOOL SRPAPI **QueryNextDown**(VS_QUERYRECORD *QueryRecord, VS_UPDOWNFILEMSG *FileInfo);
The function is valid at client or debug side

5.43.10 Get first file or static data being uploaded-QueryFirstDown

VS_BOOL SRPAPI **QueryFirstUp** (VS_QUERYRECORD *QueryRecord, VS_UPDOWNFILEMSG *FileInfo);
The function is valid at client or debug side

5.43.11 Get next file or static data being uploaded -QueryNextDown

VS_BOOL SRPAPI **QueryNextUp** (VS_QUERYRECORD *QueryRecord, VS_UPDOWNFILEMSG *FileInfo);
The function is valid at client or debug side

5.44 Static data management[download/upload callback of static data is same as file]

5.44.1 Get static data-GetStaticData

VS_INT8 ***GetStaticData**(void *Object,VS_ULONG UniqueDataUnitID, VS_STATICID *DataVersion,VS_ULONG *DataSize, VS_BOOL AutoDownLoad);
DataVersion and DataSize may be set to NULL, or else, DataVersion is the version requested for input, and is current version for output. DataSize is ingored for input, and is size of data for output.
AutoDownLoad==true, if data does not exist, then it will be downloaded from server
AutoDownLoad==false, if data does not exist, then not downloaded from server
Note: address of static data is returned, any changes to the address, will cause the change of the staticdata, and such change can not be captured by CLE, so, do not change the content directly.

5.44.2 Set static data-SetStaticData

VS_BOOL SRPAPI CanSetStaticData(void *Object,VS_ULONG DataSize); Called at client side to determine whether upload is permitted.

VS_BOOL **SetStaticData**(void *Object,VS_ULONG UniqueDataUnitID,VS_ULONG DataSize,VS_INT8 *DataBuf, VS_STATICID *RetDataVersion);
Set object static data, if succeed, the function returns version of the data. The function will also change the corresponding attribute automaticly.

5.44.3 Set static data -SetStaticDataEx

VS_BOOL SRPAPI **SetStaticDataEx**(void *Object,VS_ULONG UniqueDataUnitID,VS_ULONG *DataSize, VS_ULONG Offset,VS_CHAR *FileName ,VS_STATICID *RetDataVersion);
If DataSize==0, it will be taken from the file, and Offset is set to 0 automaticaly.
The function change locally. If is called at client or debug side, the data will not be uploaded to server.
The function takes file as static data, which will be read when the data is used.

5.44.4 Get static data code(UniqueDataUnitID)-GetStaticAppCode

VS_ULONG **GetStaticAppCode**(void *Object,OBJECTATTRIBUTEINDEX StaticPersistentAttributeIndex);
If error occurs, the function returns 0.

5.44.5 Wait static data upload/download finish-WaitGetStaticData/ WaitSetStaticData

VS_BOOL SRPAPI **WaitGetStaticData**(void *Object,VS_ULONG UniqueDataUnitID,VS_FileUpDownLoadCallBackProc CallBackProc,VS_ULONG Para ,VS_BOOL WaitFlag);
VS_BOOL SRPAPI **WaitSetStaticData**(void *Object,VS_ULONG UniqueDataUnitID,VS_FileUpDownLoadCallBackProc CallBackProc,VS_ULONG Para ,VS_BOOL WaitFlag);
If returns VS_FALSE, then not upload or download is pending, or error occurs.
If WaitFlag=VS_FALSE, only query whether there is upload or download pending.

callback:VS_FileUpDownLoadCallBackProc
If returns 0, then continue to process. Otherwise, cancel the upload or download process. Return valus is valid at VSFILE_ONDOWNPROGRESS/ VSFILE_ONUPPROGRESS procedure.

5.45 miscellaneous function

5.45.1 Get program type-GetProgramType

VS_UINT16 GetProgramType( );

5.45.2 Whether is default server-IsDefaultServer

Valid at server side.
VS_BOOL **IsDefaultServer**(); //---Is always default server for current version.

5.45.3 Whether mamager window is visible-IsWindowVisible

VS_BOOL IsWindowVisible();

5.45.4 Hide manager window-HideWindow

void **HideWindow**();

5.45.5 Show manager window -ShowWindow

void **ShowWindow**();

5.45.6 Set manager window caption-SetWindowCaption

void **SetWindowCaption**(VS_CHAR *Caption);
When create and load a service, the caption of manager window is set automaticly. So, the function should be called after service is loaded or created.

5.45.7 Exit application-ExitVSSystem

void **ExitVSSystem**(VS_CHAR *ErrorInfo);
ErrorInfo: error info, may be set to NULL

5.45.8 Whether application is active-IsAppActive

VS_BOOL **IsAppActive**();

5.45.9 Whether has message to process- SetIdleActive

void **SetIdleActive**(VS_BOOL true/false);
If input is VS_TRUE, when there is no message to be processed, CLE will continue to trigger IDLE event. Otherwise, cle will enter wait status.

5.45.10 Get starcore version-GetVersion

void **GetVersion**(VS_UINT8 *MainVersion,VS_UINT8 *SubVersion,VS_UINT16 *BuildVersion);

5.45.11 Get starcore version string -GetVersionInfo

void SRPAPI **GetVersionInfo**(VS_CHAR *InfoBuf,VS_INT32 InfoBufSize);

5.45.12 Get manager window handle-GetWindowHandle

VS_HWNDGetWindowHandle( );

5.45.13 Get manager window size-GetWindowSize

void **GetWindowSize**(VS_INT32 *Width, VS_INT32 *Height);

5.45.14 Show or hide manager window menu or status bar-ShowStatusMenu

void SRPAPI **ShowStatusMenu**(VS_BOOL MenuShowFlag, VS_BOOL StatusShowFlag);
== true show == false hide

5.45.15 Set text colot-SetColor

void SRPAPI **SetColor**(VS_COLORText, VS_COLORExplane, VS_COLORObjName, VS_COLOR, VS_COLORNumber, VS_COLORError);

5.45.16 Set text backgroud color-SetBkColor

void SRPAPI SetBkColor(VS_COLORBkColor);

5.46 Client window function(GetClientWndHandle,GetClientWndSize,SetClientWndSize,SetClientWndFocus,KillClientWndFocus)

VS_HWND**GetClientWndHandle**( );
void **GetClientWndSize**(VS_INT32 *Width, VS_INT32 *Height);
void **SetClientWndSize**(VS_INT32 Width, VS_INT32 Height);
void **SetClientWndFocus**(VS_HWND hWnd, VS_BOOL NeedAction);
If hWnd is NULL, then uses the last hWnd. If NeedAction=VS_TRUE, then the focus should be set, or else is only notification
void **KillClientWndFocus**(VS_HWND hWnd, VS_BOOL NeedAction); NeedAction=VS_TRUE, then the focus should be canceled, or else is only notification
void SRPAPI **ClearClientWnd**( )
void SRPAPI **HideClientWnd**( );
void SRPAPI **ShowClientWnd**( );
void SRPAPI **SetClientBkColor**(VS_COLOR BkColor);

5.47 Message hook(SetMessageHook,GetMessageHook)[reserved]

typedef VS_BOOL (***VS_SRPMessageProcessHookProc**)(VS_HWND hWnd,VS_ULONG message, VS_ULONG wParam, VS_ULONG lParam);
//---If the message has been processed, the function should return true; otherwise return false.
void **SetMessageHook**(VS_SRPMessageProcessHookProc HookProc);
VS_SRPMessageProcessHookProc **GetMessageHook**( );

5.48 ShareLibrary functions

VS_CHAR *SRPAPI FisrtShareLib(VS_QUERYRECORD *QueryRecord);
VS_CHAR *SRPAPI NextShareLib(VS_QUERYRECORD *QueryRecord);
VS_HANDLE SRPAPI GetShareLib(VS_CHAR *ShareLibName);
void SRPAPI FreeShareLib(VS_HANDLE ShareLibHandle);

5.49 Object Group Management

1. **VS_ULONG SRPAPI NewGroup()**
   Create an object group
2. **void SRPAPI FreeGroup(VS_ULONG GroupID)**
   Delete an object group
3. **VS_LONG SRPAPI GroupAdd(VS_ULONG GroupID,void *Object)**
   Add an object to group, the return value is RefID;
4. **void *SRPAPI GroupGet(VS_ULONG GroupID,VS_LONG RefID)**
   Get object in the group by RefID;
5. **void SRPAPI GroupRemove(VS_ULONG GroupID,VS_LONG RefID)**
   Remove object from group
6. **void SRPAPI GroupRemoveEx(VS_ULONG GroupID,void *Object)**
   Remove object from group

7. void SRPAPI GroupClear(VS_ULONG GroupID,VS_BOOL FreeObject)
Clear the group, if FreeObject == true, then objects in the group will be freed.

5.50 Dynamic register Lua function

Valid at local.

5.50.1 Register Lua function

VS_BOOL SRPAPI RegLuaFunc(void *Object, VS_CHAR *FuncName, void *FuncAddress, VS_ULONG Para);
Prototype of Lua:
VS_INT32 LuaFunction(void*L)
{
Retrun 0;
}
Parameters in UpValue is: ServiceGroupID, Para, FuncName,…, number should not exceed 8
Parameters in Lua stack is: Object, Para1, Para2….
The Lua function may be called by ExecNameScript.

if FuncName == NULL, then register general Lua Func.

5.50.2 IsRegLuaFunc

VS_BOOL SRPAPI IsRegLuaFunc(void *Object, VS_CHAR *FuncName, void *FuncAddress, VS_ULONG Para);

5.50.3 Unregister Lua funcion

void SRPAPI UnRegLuaFunc(void *Object, VS_CHAR *FuncName, void *FuncAddress, VS_ULONG Para);
void SRPAPI UnRegLuaFuncEx(void *Object, void *FuncAddress, VS_ULONG Para)

5.50.4 Forbid or permit Lua function

void SRPAPI ValidRegLuaFunc(void *Object, VS_CHAR *FuncName, void *FuncAddress, VS_ULONG Para);
void SRPAPI InValidRegLuaFunc(void *Object, VS_CHAR *FuncName, void *FuncAddress, VS_ULONG Para);

5.50.5 Lua function Filter

VS_BOOL SRPAPI RegLuaFuncFilter(void *Object, VS_LuaFuncFilterProc Filter, VS_ULONG Para);
VS_BOOL SRPAPI UnRegLuaFuncFilter(void *Object, VS_LuaFuncFilterProc Filter, VS_ULONG Para);
return VS_FALSE, then the function will be not called

5.51 Dynamic register Lua attribute function

VS_BOOL SRPAPI RegLuaGetValueFunc(void *Object, VS_LuaGetValueProc GetValueProc, VS_ULONG Para);
VS_BOOL SRPAPI RegLuaSetValueFunc(void *Object, VS_LuaSetValueProc SetValueProc, VS_ULONG Para);
In the call back, if the attribute is not defined in the callback , it should return VS_FALSE.

VS_BOOL SRPAPI UnRegLuaGetValueFunc(void *Object, VS_LuaGetValueProc GetValueProc, VS_ULONG Para);
VS_BOOL SRPAPI UnRegLuaSetValueFunc(void *Object, VS_LuaSetValueProc SetValueProc, VS_ULONG Para);
void SRPAPI ValidLuaGetValueFunc(void *Object, VS_LuaGetValueProc GetValueProc , VS_ULONG Para);
void SRPAPI ValidLuaSetValueFunc(void *Object, VS_LuaSetValueProc SetValueProc , VS_ULONG Para);
void SRPAPI InValidLuaGetValueFunc(void *Object, VS_LuaGetValueProc GetValueProc , VS_ULONG Para);
void SRPAPI InValidLuaSetValueFunc(void *Object, VS_LuaSetValueProc SetValueProc , VS_ULONG Para);

5.52 Lua Script funciton

5.52.1 GetLua

void ***GetLua**( );

5.52.2 DoBuffer

VS_BOOL **DoBuffer**(VS_CHAR *ScriptInterface, VS_INT8 *ScriptBuf,VS_INT32 ScriptBufSize, VS_CHAR *Name, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory,VS_BOOL IsUTF8);
Name is name of the script segment, which may be NULL. ErrorInfo returns error information, which is valid when the function returns VS_FALSE, and the returned string should not be freed.
ScriptInterface is NULL,which default is lua. It may be lua,python, or other online script language registered.
ScriptInterfaces length < 16
Name is the name of the segment
ModuleName should not set to “cmd”, case insensitive.

5.52.3 DoFile

Bool **DoFile**(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8);
Bool **DoFileEx**(VS_CHAR *ScriptInterface, VS_CHAR *FileName, VS_CHAR **ErrorInfo, VS_CHAR *WorkDirectory , VS_BOOL IsUTF8,VS_CHAR *ModuleName);
ErrorInfo returns error information, which is valid when the function returns VS_FALSE, and the returned string should not be freed.
ScriptInterface is NULL,which default is lua. It may be lua,python, or other online script language registered.
ScriptInterfaces length < 16
Name is the name of the segment
python,lua does not support parameter IsUTF8.
ModuleName is name of the module,may be “” or NULL. This Parameter is only valid for lua and python and java and csharp. For java and csharp, ModuleName is the init class name. For example, the ModuleName is “com.srplab.www.test”.
ModuleName should not set to “cmd”, case insensitive.
This function can load sharelibrary, in this case, ScriptInterface should be set to “”.

5.52.4 UserData

void SRPAPI LuaNewUserData(VS_INT32 Size);
void SRPAPI LuaSetUserDataGC(VS_LuaUserDataGCProc GCProc);
void *SRPAPI LuaToUserData(VS_INT32 Index);

5.52.5 LuaNewTable

void **LuaNewTable**( );

5.52.6 LuaGetTop

VS_INT32 **LuaGetTop**( );

5.52.7 LuaNext

VS_INT32 **LuaNext**(VS_INT32 Index);

5.52.8 LuaPop

void **LuaPop**(VS_INT32 Index);

5.52.9 LuaPushBool

void **LuaPushBool**(VS_BOOL Value);

5.52.10 LuaPushString/ LuaPushLString

void **LuaPushString**(VS_CHAR *Value);
void SRPAPI **LuaPushLString**(VS_CHAR *Value, VS_ULONG Len);

5.52.11 LuaPushNumber

void **LuaPushNumber**(double Value);

5.52.12 LuaPushInt

void **LuaPushInt**(VS_INT32 Value);

5.52.13 LuaPushNil

void **LuaPushNil**( );

5.52.14 LuaPushObject

VS_BOOL **LuaPushObject**(void *Object);
If returns VS_FALSE, then the input object pointer is error. In this case, nil will be pushed to lua stack. The function can only push normal object except service, service group, and service item.

5.52.15 LuaPushRect

VS_BOOL SRPAPI **LuaPushRect**(VS_RECT *rEct);

5.52.16 LuaPushFont

VS_BOOL SRPAPI **LuaPushFont**(VS_FONT *hFont);

5.52.17 LuaPushTime

VS_BOOL SRPAPI **LuaPushTime**(VS_TIME *hTime);

5.52.18 LuaPushParaPkg

VS_BOOL **LuaPushParaPkg**(class ClassOfSRPParaPackageInterface *ParaPkg , VS_BOOL AutoRelease);
If returns VS_FALSE, then the input object pointer is error. In this case, nil will be pushed to lua stack.If AutoRelease = true, then the function pass the address of ParaPkg to script, which will be freed by lua script other than the caller. ==false,Lua does not respond for freeing the address.

5.52.19 LuaPushQueryRecord

VS_BOOL SRPAPI **LuaPushQueryRecord**(VS_QUERYRECORD *QueryRecord, VS_BOOL AutoRelease);; If returns VS_FALSE, then the input object pointer is error. In this case, nil will be pushed to lua stack.QueryRecord should be allocated using interface function Malloc.
AutoRelease = true, then the function pass the address of QueryRecord to script, then the address will be freed by lua script other than the caller. ==false,Lua does not respond for freeing the address.

5.52.20 LuaPushFunction

void **LuaPushFunction** (void *FunctionAddr);

5.52.21 LuaPushClosure

void SRPAPI **LuaPushClosure**(void *FunctionAddr, VS_INT32 n);

5.52.22 LuaUpValueIndex

VS_INT32 SRPAPI **LuaUpValueIndex**(VS_INT32 Index);

5.52.23 LuaPushValue

void **LuaPushValue**(VS_INT32 Index);

5.52.24 LuaInsert

void **LuaInsert**(VS_INT32 Index);

5.52.25 LuaRemove

void **LuaRemove** (VS_INT32 Index);

5.52.26 LuaPCall

VS_BOOL SRPAPI **LuaPCall**(VS_INT32 nargs, VS_INT32 nresults);
Details explanation refers to lua manual.
If returns VS_FALSE,then there is only one value on stack top, which is error string.
After the call, application should popup the return values from Lua stack

5.52.27 Lua remote call

void SRPAPI **LuaRCall** (VS_ULONG ClientID,void *Object, VS_CHAR *ScriptName, VS_INT32 nArgs);
void SRPAPI **LuaRCallEx** (VS_ULONG ExcludeClientID,void *Object, VS_CHAR *ScriptName, VS_INT32 nArgs);
VS_BOOL SRPAPI **LuaSRCall**(VS_ULONG WaitTime, VS_ULONG ClientID,void *Object, VS_CHAR *ScriptName, VS_INT32 nArgs, VS_INT32 *OutArgs);
If returns VS_TRUE, then OutArgs is the number of return values, which is in order RetCode,return value 1,return value 2…. The caller should pop up the return values to clear the lua stack.
VS_BOOL SRPAPI **LuaARCall**(VS_ULONG WaitTime, VS_ULONG ClientID,void *Object, void *CallBackProc, VS_ULONG Para, VS_CHAR *ScriptName, VS_INT32 nArgs);
CallBackProc is Lua function, which prototype is VS_INT32 CallBackProc(void *L);
the Lua stack is arranged as:Object, RetCode,ServiceGroupID,Para, RetValue.
WaitTime: If equals to 0, then wait forever. Its unit is (ms).

5.52.28 Lua event register and trigger

VS_INT32 SRPAPI **LuaRegEvent**(void *SrcObject,VS_UUID *EventID, void *Object, void *FuncAddr);
void SRPAPI **LuaUnRegEvent**(void *SrcObject,VS_UUID *EventID, void *Object, VS_INT32 FuncRefValue);
FuncAddr is Lua function, which prototype is VS_INT32 FuncAddr (void *L);
The function returns Lua ref, which is used in unregister function.
If the event is triggered, the registered function will be called. In this case, parameters in lua stack is in order:DesObject,Event table,parameter 1,parameter 2…
L may by invalid.

VS_BOOL SRPAPI **LuaProcessEvent**(void *Object,VS_UUID *EventID, VS_INT32 nArgs, VS_INT32 *OutArgs);
VS_BOOL SRPAPI **LuaPostProcessEvent**(void *Object,VS_UUID *EventID, VS_INT32 nArgs, VS_INT32 *OutArgs);
If returns VS_TRUE, then OutArgs is the number of return values. The caller should popup the return values to clear the lua stack.

5.52.29 Execute script[reserved]

VS_BOOL SRPAPI **LuaCall**(void *Object, VS_CHAR *ScriptName, VS_INT32 nArgs,VS_INT32 nOutArgs);

If Object == NULL,then the function is global lua function .
The difference between the function and ExecNameScript, in that the function does not process script hook function.

script hook function is set using the following function:
void SRPAPI **LuaRegHook**(void *FuncAddr);
void SRPAPI **LuaUnRegHook**(void *FuncAddr);

5.52.30 LuaType

VS_INT32 **LuaType** (VS_INT32 Index);

```
#define VSLUATYPE_NIL         0
#define VSLUATYPE_NUMBER      1
#define VSLUATYPE_BOOL        2
#define VSLUATYPE_STRING      3
#define VSLUATYPE_FUNCTION    4
#define VSLUATYPE_TABLE       5
#define VSLUATYPE_OBJECT      6
#define VSLUATYPE_PARAPKG     7
#define VSLUATYPE_QUERYRECORD 8
#define VSLUATYPE_TIME        9
#define VSLUATYPE_FONT        10
#define VSLUATYPE_RECT        11
#define VSLUATYPE_UNKNOWN     255
```

5.52.31 LuaToBool

VS_BOOL **LuaToBool**(VS_INT32 Index);

5.52.32 LuaToObject

void ***LuaToObject**(VS_INT32 Index);

5.52.33 LuaToParaPkg

class ClassOfSRPParaPackageInterface ***LuaToParaPkg**(VS_INT32 Index);

5.52.34 LuaToQueryRecord

VS_QUERYRECORD *SRPAPI **LuaToQueryRecord**(VS_INT32 Index);

5.52.35 LuaToString

VS_CHAR ***LuaToString**(VS_INT32 Index);

5.52.36 LuaToNumber

double **LuaToNumber**(VS_INT32 Index);

5.52.37 LuaToInt

VS_INT32 **LuaToInt**(VS_INT32 Index);

5.52.38 LuaToRect

VS_BOOL SRPAPI **LuaToRect**(VS_INT32 Index, VS_RECT *rEct);

5.52.39 LuaToFont

VS_BOOL SRPAPI **LuaToFont**(VS_INT32 Index, VS_FONT *hFont);

5.52.40 LuaToTime

VS_BOOL SRPAPI **LuaToTime**(VS_INT32 Index, VS_TIME *hTime);

5.52.41 LuaIsBool

VS_BOOL **LuaIsBool**(VS_INT32 Index);

5.52.42 LuaIsString

VS_BOOL **LuaIsString**(VS_INT32 Index);

5.52.43 LuaIsNumber

VS_BOOL **LuaIsNumber**(VS_INT32 Index);

5.52.44 LuaIsInt

VS_BOOL **LuaIsInt**(VS_INT32 Index);
If returns VS_TRUE,then function LuaIsNumber returns VS_TRUE too.

5.52.45 LuaIsTable

VS_BOOL **LuaIsTable**(VS_INT32 Index);

5.52.46 LuaIsNil

VS_BOOL **LuaIsNil**(VS_INT32 Index);

5.52.47 LuaIsObject

VS_BOOL **LuaIsObject**(VS_INT32 Index);

5.52.48 LuaIsParaPkg

VS_BOOL **LuaIsParaPkg**(VS_INT32 Index);

5.52.49 LuaIsQueryRecord

VS_BOOL **LuaIsQueryRecord** (VS_INT32 Index);

5.52.50 LuaIsCFunction

VS_BOOL **LuaIsCFunction** (VS_INT32 Index);

5.52.51 LuaIsFunction

VS_BOOL **LuaIsFunction** (VS_INT32 Index);

5.52.52 LuaIsFont

VS_BOOL **LuaIsFont** (VS_INT32 Index);

5.52.53 LuaIsRect

VS_BOOL **LuaIsRect** (VS_INT32 Index);

5.52.54 LuaIsTime

VS_BOOL **LuaIsTime** (VS_INT32 Index);

5.52.55 LuaSetTable

VS_BOOL **LuaSetTable**(VS_INT32 Index); Supports table and object

5.52.56 LuaGetTable

VS_BOOL **LuaGetTable**(VS_INT32 Index); Supports table and object
If return value is not nil, then application may uses function LuaGetDefinedClass to get which class define the value.
void SRPAPI LuaGetDefinedClass(void *Object,VS_UUID *ObjectID);

5.52.57 LuaIsFunctionDefined

void *SRPAPI LuaIsFunctionDefined(void *Object, VS_CHAR *FuncName, VS_BOOL IncludeRawOrDefaultFunction)
If there are lua function or other script function registered with the object, then the function returns true.
IncludeRawOrDefaultFunction: == true, then the function of raw script object or registered using RegLuaFunc with FuncName == null is take into count.
The return value is object or class which defined the function.

5.52.58 LuaSetGlobal

void **LuaSetGlobal** (VS_CHAR *Name);

5.52.59 LuaGetGlobal

void **LuaGetGlobal** (VS_CHAR *Name);

5.52.60 LuaSetRef[reserved]

VS_INT32 SRPAPI **LuaSetRef**(void *Object, VS_INT32 Index);
Set lua ref by Index, which will affect lua gc. If function LuaClearRef is not called, the object will not be garbage collected.
If Index==-1, then pop up top value of lua stack, or else, not pop up.
If returns VS_LUAREFNIL, then error occurs.
Object can not be set to NULL

5.52.61 LuaClearRef[reserved]

void SRPAPI **LuaClearRef**(void *Object, VS_INT32 LuaRefValue);

5.52.62 LuaGetRef[reserved]

Void SRPAPI **LuaGetRef**(void *Object,VS_INT32 LuaRefValue);
Top of lua stack return the value indexed by LuaRefValue.

5.52.63 LuaGetObjectValue

VS_BOOL SRPAPI LuaGetObjectValue(void *Object,VS_CHAR *Name);
Value is returned on top of lua stack.

5.52.64 LuaSetObjectValue

VS_BOOL SRPAPI LuaSetObjectValue(void *Object,VS_CHAR *Name);
Value on top of stack is to be set. The functon pop up the value, and set it to the object.

5.53 Register query function

5.53.1 Read string-GetRegStr

VS_CHAR *SRPAPI GetRegStr(VS_CHAR *SubKey, VS_CHAR *ValueName, VS_CHAR *DefaultValue);
SubKey is key value such as “Software\\SRPLab\\SRPServer"
If not exists, DefaultValue is returned.

5.53.2 Read int-GetRegInt

VS_ULONG SRPAPI GetRegInt(VS_CHAR *SubKey, VS_CHAR *ValueName,VS_ULONG DefaultValue);
SubKey is key value such as “Software\\SRPLab\\SRPServer"
If not exists, DefaultValue is returned.

5.54 Timer function

5.54.1 Setup timer-SetupTimer

VS_ULONG **SetupTimer**(VS_INT32 Ticket, VS_TimerProc FunctionAddr, void *Object, VS_ULONG Para1,VS_ULONG Para2,VS_ULONG Para3,VS_ULONG Para4);
If failes, returns 0. Otherwise the function returns TimerID. Unit of Ticket is 10ms.

typedef void (SRPAPI *VS_TimerProc)(void *Object, VS_ULONG TimerID, VS_ULONG Para1,VS_ULONG Para2,VS_ULONG Para3,VS_ULONG Para4);
Object is not NULL.
TimerID = 0xFFFFFFFF, indicates the timer will be deleted.

5.54.2 Kill timer-KillTimer

void **KillTimer**(VS_ULONG TimerID);

5.54.3 Get current Ticket-GetTickCount

VS_ULONG SRPAPI GetTickCount(); return the number of ms.
If high precision is supported, then the function will return high precision timer.

5.55 Encryption functions

5.55.1 MD5 encryption -GetMD5

VS_INT8 *SRPAPI **GetMD5**(VS_INT8 *Buf,VS_INT32 BufSize);

5.56 Exception handler

5.56.1 exception handler-SetExceptHandler

void SRPAPI **SetExceptHandler**(VS_ExceptHandlerProc ExceptHandlerProc);
VS_ExceptHandlerProc is defined as:
typedef void (SRPAPI *VS_ExceptHandlerProc)(VS_CHAR *ErrorInfo);
ErrorInfo is error information.
The function is called when exception occurs.

5.57 Get basic service interface(ClassOfBasicSRPInterface)

5.57.1 Get basic service interface -GetBasicInterface

class ClassOfBasicSRPInterface ***GetBasicInterface**();
The interface must be freed using its Release function.

5.58 CLE Lock

5.58.1 Lock CLE-SRPLock

void SRPAPI **SRPLock**( )

5.58.2 UnLock CLE-SRPUnLock

void SRPAPI **SRPUnLock**( )

5.59 compression and decompression functions(uses zlib)

5.59.1 Compress

VS_BOOL **Compress**(VS_UINT8 *dest,VS_ULONG *destLen,VS_UINT8 *source,VS_ULONG sourceLen);
destLen is destination buffer size

5.59.2 UnCompress

VS_BOOL **UnCompress**(VS_UINT8 *dest,VS_ULONG *destLen,VS_UINT8 *source,VS_ULONG sourceLen);
destLen is destination buffer size

5.60 Synchronous related functions

5.60.1 Whether service item is sync-IsSysRootItemSync,Valid at client

VS_BOOL SRPAPI IsSysRootItemSync(void *SystemRootItem);

5.60.2 Wait service item to sync-WaitSysRootItemSync, Valid at client

VS_BOOL SRPAPI **WaitSysRootItemSync**(void *SystemRootItem);
The function waits service item to sync forever.

5.60.3 Synchronous remote call-SRemoteCall, Valid at client

VS_ULONG SRPAPI **SRemoteCall**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object,VS_UUID *FunctionID,...);
VS_ULONG SRPAPI **SRemoteCallVar**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object,VS_UUID *FunctionID ,va_list argList);
Type supported:
VS_BOOL,VS_INT32,VS_FLOAT,VS_CHARPTR,VS_PARAPKGPTR,VS_OBJPTR.

For caller:
If input parameter is parapkg or binbuf, then the caller is responsible for freeing.
If output is string, parapkg, or binbuf, then application should change output from VS_ULONG to proper pointer type.
if output is parapkg or binbuf, then the caller is responsible for freeing.
Note: VS_OBJPTR should not be freed by caller.

For called:
If the called function returns parapkg and binbuf, then the called function is responsible for freeing.

```
//---- RetCode –Result of remotecall
#define VSRCALL_OK            0   //---successful
#define VSRCALL_COMMERROR     -1   //---communication error
```

#define VSRCALL_OBJNOTEXIST -2 //---object not exist
#define VSRCALL_FUNCNOTEXIST -3 //--- function not exist
#define VSRCALL_PARAERROR -4 //---parameter error
#define VSRCALL_SYSERROR -5 //---system error
#define VSRCALL_INVALIDUSR -6 //---client illegal
#define VSRCALL_UNKNOWN -7 //---other errors

If WaitTime=0, then wait for ever.

5.60.4 Asynchronous remote call -ARemoteCall

VS_BOOL SRPAPI **ARemoteCall**(VS_ULONG WaitTime, VS_ULONG ClientID,VS_RemoteCallResponseProc CallBackProc,VS_ULONG Para,void *Object,VS_UUID *FunctionID,...);
VS_BOOL SRPAPI **ARemoteCallVar**(VS_ULONG WaitTime, VS_ULONG ClientID,VS_RemoteCallResponseProc CallBackProc,VS_ULONG Para,void *Object,VS_UUID *FunctionID,va_list argList);
Type supported:
VS_BOOL,VS_INT32,VS_FLOAT,VS_CHARPTR,VS_PARAPKGPTR,VS_OBJPTR.

For caller function:
If input parameter is parapkg or binbuf, the caller is responsible for freeing.
For called function:
If the called function returns parapkg or binbuf, then the called function is responsible for freeing.
For response callback function:
If input is parapkg or binbuf, CLE is responsible for freeing.

typedef void (SRPAPI *VS_RemoteCallResponseProc)(void *Object , VS_ULONG Para,VS_ULONG RetCode,VS_ULONG RetValue);
If WaitTime=0, then wait for ever.

5.60.5 Get remotecall ID-GetRemoteCallID

VS_ULONG SRPAPI GetRemoteCallID(void *Object);

5.60.6 Set remotecall response postponed-SetDeferRspFlag

void SRPAPI **SetDeferRspFlag**(void *Object);
The function should be used in the called function.

5.60.7 Set response code-SetRetCode

void SRPAPI **SetRetCode**(void *Object,VS_ULONG RemoteRetCode);
The function should be used in the called function.
#define VSRCALL_OK 0 //---successful
#define VSRCALL_COMMERROR -1 //---communication error
#define VSRCALL_OBJNOTEXIST -2 //---object not exist
#define VSRCALL_FUNCNOTEXIST -3 //--- function not exist
#define VSRCALL_PARAERROR -4 //---parameter error
#define VSRCALL_SYSERROR -5 //---system error
#define VSRCALL_INVALIDUSR -6 //---client illegal
#define VSRCALL_UNKNOWN -7 //---other errors

5.60.8 Set response attach parameter-SetRemoteRspAttach[reserved]

void SRPAPI **SetRemoteRspAttach**(void *Object,void *RemoteAttach);
RemoteAttach: is attached response parameter, which is valid for VSRCALLSRC_WEBSERVICE. It is defined as follows:

```
struct StructOfVSRemoteCallResponseAttach_WebService{
   class ClassOfSRPSXMLInterface *SoapInfo;
   struct{
      VS_ULONG MimeDataSize;
      VS_INT8  *MimeDataBuf;
   }MimeData;
   VS_CHAR  *MimeContentType;
};
```

SoapInfo: may return segment Envelop,Head,Body,Operation. If it exists, it should be complete.

5.60.9 Response remotecall-RemoteCallRsp

void SRPAPI **RemoteCallRsp**(void *Object,VS_ULONG ClientID,VS_ULONG RemoteCallID, VS_CHAR *RemoteCallName,VS_UINT16 RemoteSourceTag,VS_ULONG RetCode,VS_UINT8 RetType,VS_ULONG RetValue, void *RemoteAttach);
If **SetDeferRspFlag** is called. Then after the called function returns, application can use this function to return the response result to client.
ClientID: Obtained by function **GetRemoteID**.
RemoteCallID: Obtained by function **GetRemoteCallID**.
RemoteSourceTag: Obtained by function **GetRemoteSourceTag**.
RetType supported is list as follows:

```
#define VSTYPE_BOOL
#define VSTYPE_INT8
#define VSTYPE_UINT8
#define VSTYPE_INT16
#define VSTYPE_UINT16
#define VSTYPE_INT32
#define VSTYPE_UINT32
#define VSTYPE_FLOAT            RetValue is float, ( (VS_ULONG *)&VS_FLOAT)[0]
#define VSTYPE_LONG
#define VSTYPE_ULONG
#define VSTYPE_LONGHEX
#define VSTYPE_ULONGHEX
#define VSTYPE_CHARPTR
#define VSTYPE_PARAPKGPTR
#define VSTYPE_BINBUFPTR
#define VSTYPE_OBJPTR       0x2A
```

If return values is string, then uses type VSTYPE_CHARPTR.
RetValue: Vary according to type. For string, it is address of the string. For parapkg or binbuf, it is the address of the parapkg or binbuf, which should be freed by the caller.
RemoteAttach: is attached response parameter, which is valid for VSRCALLSRC_WEBSERVICE. It is definedas follows:

```
struct StructOfVSRemoteCallResponseAttach_WebService{
   class ClassOfSRPSXMLInterface *SoapInfo;
   struct{
      VS_ULONG MimeDataSize;
      VS_INT8  *MimeDataBuf;
   }MimeData;
   VS_CHAR  *MimeContentType;
};
```

SoapInfo:may be return segment Envelop,Head,Body,Operation. If exists, must be complete.

*[**RemoteAttach** should be set to NULL]*

5.60.10 Fill soap response header-FillSoapRspHeader

VS_BOOL SRPAPI **FillSoapRspHeader**(class ClassOfSRPSXMLInterface *SXMLInterface);
Format as follows:
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="urn:starcore-XXXXX">
</SOAP-ENV:Envelope>

5.61 Atomic oject function

Atomic object interface, which may be used to create service, objects attributes or functions,etc.
Attribute index used in atomic object, is global index. If it is used in other functions, should be converted to external index.

5.61.1 Get atomic service-GetAtomicService

void *SRPAPI **GetAtomicService**();

5.61.2 Create atomic service item-CreateAtomicSysRootItem[valid at server]

void *SRPAPI **CreateAtomicSysRootItem**(VS_CHAR *SysRootItemName, VS_CHAR *DependSysRootItem ,VS_UUID *SystemRootItemID, VS_UUID *SystemRootItemIDEx)
Input is service item name, and depended service item name.
SystemRootItemID, SystemRootItemIDEx may be set to NULL

5.61.3 Get atomic service item-GetAtomicSysRootItem

void *SRPAPI **GetAtomicSysRootItem**(VS_CHAR *SysRootItemName)

5.61.4 Get atomic object-GetAtomicObject/ GetAtomicObjectEx

void *SRPAPI **GetAtomicObject**(VS_UUID *UuidPtr);
void *SRPAPI **GetAtomicObjectEx**(void *ParentAtomicObject, VS_CHAR *ObjectName);

5.61.5 Get atomic class object-GetAtomicClass

void *SRPAPI **GetAtomicClass**(void *AtomicObject)

5.61.6 Get atomic object ID-GetAtomicID

void SRPAPI **GetAtomicID**(void *AtomicObject,VS_UUID *UuidPtr)

5.61.7 Get atomic object name-GetAtomicName

VS_CHAR *SRPAPI **GetAtomicName**(void *AtomicObject)

5.61.8 Atomic object to normal object-AtomicToObject

void *SRPAPI **AtomicToObject**(void *AtomicObject);

5.61.9 Normal object to atomic object-ObjectToAtomic

void *SRPAPI **ObjectToAtomic**(void *Object);

5.61.10 Create atomic macro-CreateAtomicMacro[valid at server]

void *SRPAPI **CreateAtomicMacro**(VS_CHAR *MacroName , VS_UINT8 MacroType);
Input is name of macro define
MacroType = 0 Integer
MacroType = 1 Float
MacroType = 2 String

5.61.11 Create atomic macro itrm-CreateAtomicMacroItem[valid at server]

VS_BOOL SRPAPI **CreateAtomicMacroItem** (void *MacroObject, VS_CHAR *MacroItemName, VS_CHAR *MacroItemValue);
Input is macro item name and value string

5.61.12 Create atmoic module-CreateAtomicModule[valid at server]

void *SRPAPI **CreateAtomicModule**(VS_CHAR *ModuleName, VS_UINT16 ModuleType,VS_UUID *ModuleID);
Input is module name
ModuleID may be set to NULL

5.61.13 Create atomic edit module-CreateAtomicEditModule[valid at server]

void *SRPAPI **CreateAtomicEditModule**(VS_CHAR *ModuleName ,VS_UUID *ModuleID);
Input is module name
ModuleID may be set to NULL

5.61.14 Create atomic struct-CreateAtomicStruct[valid at server]

void *SRPAPI **CreateAtomicStruct**(VS_CHAR *StructName , VS_CHAR *StructCaption,VS_UUID *StructID);
Input is struct name
StructID and StructCaption may be set to NULL

5.61.15 Create atomic object-CreateAtomicObject[valid at server]

void *SRPAPI **CreateAtomicObject**(void *AtomicObject, OBJECTATTRIBUTEINDEX AtomicAttributeIndex, void *AtomicClassObject, VS_CHAR *ObjectName, VS_UUID *ObjectID);
AtomicObject : atomic parent object, may be atomic object or atomic service item.
AtomicAttributeIndex : Attribute global index of queue of atomic parent, which can be obtained from VS_ATTRIBUTEINFO. For atomic service item, the paramter should be set to 0.
AtomicClassObject: atomic class object, may be set to NULL
ObjectName: atomic object name.

ObjectID: may be set to NULL

5.61.16 Create atomic attach attribute-CreateAtomicAttachAttribute[reserved valid at server]

void *SRPAPI **CreateAtomicAttachAttribute**(void *AtomicObject, VS_CHAR *AttributeName, VS_CHAR *Caption, VS_UINT8 Type, VS_ULONG StaticID, VS_UINT8 SyncFlag,VS_UINT8 CreateFlag,VS_UINT8 NotifyFlag,VS_UINT8 EditType,VS_UINT8 EditControl,VS_UINT8 EditReadOnly, VS_CHAR *Default,VS_CHAR *Desc);
AtomicObject: Atomic parent object
AttributeName: attribute name
Type: attribute type.
Caption, Default, Desc may be set to NULL
SyncFlag = 1, for global attribute, otherwise, is local attribute.

Types supported in attach attribute:
VSTYPE_BOOL :
VSTYPE_INT8 :
VSTYPE_UINT8 :
VSTYPE_INT16 :
VSTYPE_UINT16 :
VSTYPE_INT32 :
VSTYPE_UINT32 :
VSTYPE_FLOAT :
VSTYPE_LONG :
VSTYPE_ULONG :
VSTYPE_CHAR :
VSTYPE_COLOR :
VSTYPE_RECT :
VSTYPE_FONT :
VSTYPE_TIME :
VSTYPE_STRUCT :

a. “SyncFlag”,VSTYPE_UINT8,default is 0; note:0 global attribute; 1 local attribute,
b. “CreateFlag”,VSTYPE_UINT8,default is 0; note:0 not need; 1 need
c. “NotifyFlag”,VSTYPE_UINT8,default is 0; note:1 notify before change; 2 notify when change; or combination of the two value.
d. “StaticID”,VSTYPE_ULONG,default is 0
e. “EditType”, VSTYPE_UINT8,default is 0; note:0 normal edit; 1 combobox; 3 button; 4 check box; 5 can not edit; 6 mask edit.
f. “EditControl”, VSTYPE_UINT8,default is 0; note:0 none 1 display callback
g. “EditReadOnly”, VSTYPE_UINT8,default is 0; note:0 edit 1 readonly
h. “ComboBox”,VS_CHAR,default is ““, or else is macro name

5.61.17 Create atomic attribute-CreateAtomicAttribute[valid at server]

void *SRPAPI **CreateAtomicAttribute** (void *AtomicObject, VS_CHAR *AttributeName, VS_CHAR *Caption, VS_UINT8 Type, VS_ULONG StaticID, VS_UINT8 SyncFlag,VS_UINT8 CreateFlag,VS_UINT8 NotifyFlag,VS_UINT8 EditType,VS_UINT8 EditControl,VS_UINT8 EditReadOnly, VS_CHAR *Default,VS_CHAR *Desc);
AtomicObject: Atomic parent object
AttributeName: attribute name
Type: attribute type.
Caption, Default, Desc may be set to NULL
SyncFlag = 1, for global attribute, otherwise, is local attribute.

type supported in attribute:

VSTYPE_BOOL :
VSTYPE_INT8 :
VSTYPE_UINT8 :
VSTYPE_INT16 :
VSTYPE_UINT16 :
VSTYPE_INT32 :
VSTYPE_UINT32 :
VSTYPE_FLOAT :
VSTYPE_LONG :
VSTYPE_ULONG :
VSTYPE_LONGHEX :
VSTYPE_ULONGHEX :
VSTYPE_VSTRING :
VSTYPE_PTR :
VSTYPE_STRUCT :
VSTYPE_CHAR :
VSTYPE_COLOR :
VSTYPE_RECT :
VSTYPE_FONT :
VSTYPE_TIME :
VSTYPE_UUID :
VSTYPE_STATICID :

5.61.18 Create atomic function return value-CreateAtomicFuncRetAttribute[valid at server]

void *SRPAPI **CreateAtomicFuncRetAttribute** (void *AtomicObject, VS_UINT8 Type,VS_CHAR *Desc);
AtomicObject: atomic parent function
Type: attribute type

type supported in function return value:

VSTYPE_BOOL :
VSTYPE_INT8 :
VSTYPE_UINT8 :
VSTYPE_INT16 :
VSTYPE_UINT16 :
VSTYPE_INT32 :
VSTYPE_UINT32 :
VSTYPE_FLOAT :
VSTYPE_LONG :
VSTYPE_ULONG :

VSTYPE_CHARPTR :
VSTYPE_PARAPKGPTR :
VSTYPE_BINBUFPTR
VSTYPE_PTR :
VSTYPE_VOID :

VSTYPE_INT8PTR :
VSTYPE_UINT8PTR :
VSTYPE_INT16PTR :
VSTYPE_UINT16PTR :
VSTYPE_INT32PTR :
VSTYPE_UINT32PTR :

VSTYPE_FLOATPTR :
VSTYPE_LONGPTR :
VSTYPE_ULONGPTR :
VSTYPE_STRUCTPTR :
VSTYPE_COLORPTR :
VSTYPE_RECTPTR :
VSTYPE_FONTPTR :
VSTYPE_TIMEPTR :
VSTYPE_UUIDPTR :
VSTYPE_OBJPTR :

5.61.19 Create atomic function parameter-CreateAtomicFuncParaAttribute[valid at server]

void *SRPAPI **CreateAtomicFuncParaAttribute** (void *AtomicObject, VS_CHAR *AttributeName, VS_CHAR *AttributeCaption,VS_UINT8 Type,VS_CHAR *Desc);
AtomicObject: atomic parent function
AttributeName: attribute name
Type: attribute type
AttributeCaption: may be set to NULL
type supported in function parameter value:

VSTYPE_BOOL :
VSTYPE_INT8 :
VSTYPE_UINT8 :
VSTYPE_INT16 :
VSTYPE_UINT16 :
VSTYPE_INT32 :
VSTYPE_UINT32 :
VSTYPE_FLOAT :
VSTYPE_LONG :
VSTYPE_ULONG :
VSTYPE_COLOR :
VSTYPE_RECT :
VSTYPE_FONT :
VSTYPE_TIME :
VSTYPE_UUID :
VSTYPE_PARAPKGPTR :
VSTYPE_BINBUFPTR

VSTYPE_PTR :
VSTYPE_VOID :
VSTYPE_CHARPTR :

VSTYPE_INT8PTR :
VSTYPE_UINT8PTR :
VSTYPE_INT16PTR :
VSTYPE_UINT16PTR :
VSTYPE_INT32PTR :
VSTYPE_UINT32PTR :
VSTYPE_FLOATPTR :
VSTYPE_LONGPTR :
VSTYPE_ULONGPTR :
VSTYPE_STRUCTPTR :
VSTYPE_COLORPTR :
VSTYPE_RECTPTR :

VSTYPE_FONTPTR :
VSTYPE_TIMEPTR :
VSTYPE_UUIDPTR :
VSTYPE_OBJPTR :

5.61.20 Create atomic struct attribute-CreateAtomicStructAttribute[valid at server]

void *SRPAPI **CreateAtomicStructAttribute** (void *AtomicObject, VS_CHAR *AttributeName, VS_CHAR *Caption, VS_UINT8 Type, VS_CHAR *Desc);
AtomicObject: atomic parent struct
AttributeName: attribute name
Type: attribute type
Caption, Desc may be set to NULL

type supported in struct attribute:
VSTYPE_BOOL :
VSTYPE_INT8 :
VSTYPE_UINT8 :
VSTYPE_INT16 :
VSTYPE_UINT16 :
VSTYPE_INT32 :
VSTYPE_UINT32 :
VSTYPE_FLOAT :
VSTYPE_LONG :
VSTYPE_ULONG :
VSTYPE_CHAR :
VSTYPE_COLOR :
VSTYPE_RECT :
VSTYPE_FONT :
VSTYPE_TIME :
VSTYPE_UUID :
VSTYPE_MEMORY :(reserved)

5.61.21 Set atomic attribute length-SetAtomicAttributeLength[valid at server], GetAtomicAttributeLength

VS_BOOL SRPAPI **SetAtomicAttributeLength**(void *AtomicObject, VS_INT32 Length);
VS_BOOL SRPAPI **GetAtomicAttributeLength**(void *AtomicObject, VS_INT32 *Length)
AtomicObject: atomic attribute

5.61.22 Set atomic attribute struct-SetAtomicAttributeStruct[valid at server], GetAtomicAttributeStruct

VS_BOOL SRPAPI **SetAtomicAttributeStruct**(void *AtomicObject, void *AtomicStruct);
void *SRPAPI **GetAtomicAttributeStruct**(void *AtomicObject)
AtomicObject: atomic attribute
AtomicStruct: atomic struct
If the type of atomic attribute is VSTYPE_PTR, then AtomicStruct is atomic class, indicates the queue corresponds to which atomic class

5.61.23 Set atomic attribute combobox-SetAtomicAttributeCombobox[valid at server], GetAtomicAttributeCombobox

VS_BOOL SRPAPI **SetAtomicAttributeCombobox** (void *AtomicObject, void *MarcoName);

void *SRPAPI **GetAtomicAttributeCombobox** (void *AtomicObject) return atomic macro
AtomicObject: Atomic attribute
Combobox is only used when object is edited, which is atomic macro.

5.61.24 Set atomic attribute synchronous type-SetAtomicAttributeSyncFlag[valid at server], GetAtomicAttributeSyncFlag

VS_BOOL SRPAPI **SetAtomicAttributeSyncFlag**(void *AtomicObject, VS_UINT8 SyncFlag)
VS_UINT8 SRPAPI **GetAtomicAttributeSyncFlag**(void *AtomicObject);
SyncFlag =0 global attribute; == 1 local attribute;

5.61.25 Attribute index convert-ToAttributeIndex,

OBJECTATTRIBUTEINDEX SRPAPI **ToAtomicAttributeIndex**(void *AtomicObject,OBJECTATTRIBUTEINDEX AttributeIndex);
Global attribute index to extern attribute index. If fails, the function returns INVALID_OBJECTATTRIBUTEINDEX.
For atomic struct, its attribute index does not need to convert

OBJECTATTRIBUTEINDEX SRPAPI **ToAttributeIndex**(void * AtomicObject,OBJECTATTRIBUTEINDEX AtomicAttributeIndex);
Extern attribute index to global attribute index. If fails, the function returns INVALID_OBJECTATTRIBUTEINDEX.
For atomic struct, its attribute index does not need to convert

5.61.26 Get atomic struct attribute number and info -GetAtomicStructAttributeNumber, GetAtomicStructAttributeSize, GetAtomicStructAttributeInfo, GetAtomicStructAttributeInfoEx

VS_INT32 SRPAPI **GetAtomicStructAttributeNumber**(void *AtomicObject); AtomicObject is atomic struct
VS_INT32 SRPAPI **GetAtomicStructAttributeSize**void *AtomicObject); AtomicObject is atomic struct
VS_BOOL SRPAPI **GetAtomicStructAttributeInfo**(void *AtomicObject,VS_CHAR *AttributeName,VS_ATTRIBUTEINFO *AttributeInfo);
VS_BOOL SRPAPI **GetAtomicStructAttributeInfoEx**(void *AtomicObject,OBJECTATTRIBUTEINDEX ThisAtomicAttributeIndex,VS_ATTRIBUTEINFO *AttributeInfo);

5.61.27 Get atomic function attribute number-GetAtomicFuncRetAttributeNumber, GetAtomicFuncParaAttributeNumber

VS_INT32 SRPAPI **GetAtomicFuncRetAttributeNumber**(void *AtomicObject); AtomicObject is atomic function
VS_INT32 SRPAPI **GetAtomicFuncParaAttributeNumber**(void *AtomicObject); AtomicObject is atomic function

5.61.28 Create atomic script-CreateAtomicScript[valid at server]

void *SRPAPI **CreateAtomicScript**(void *AtomicObject, VS_CHAR *ScriptName, VS_UUID *ScriptID, VS_CHAR *Desc, VS_UINT8 *ScriptBuf);
AtomicObject: atomic parent object
ScriptName: script name
ScriptBuf: script content
ScriptID may be set to NULL

Desc may be set to NULL

5.61.29 Create atomic function-CreateAtomicFunction[valid at server]

void *SRPAPI **CreateAtomicFunction**(void *AtomicObject, VS_CHAR *FunctionName , VS_UUID *FunctionID, VS_CHAR *Desc, VS_BOOL CantOvl, VS_BOOL CallBack, VS_BOOL StdCallFlag,VS_BOOL GlobalFunctionFlag);
void *SRPAPI **CreateAtomicFunctionEx**(void *AtomicObject, VS_CHAR *FunctionName , VS_UUID *FunctionID, VS_CHAR *Desc, VS_BOOL CantOvl, VS_BOOL CallBack, VS_CHAR *Type,VS_CHAR **ErrorInfo, VS_BOOL StdCallFlag,VS_BOOL GlobalFunctionFlag);
AtomicObject: atomic parent object
FunctionName: function name
FunctionID may be set to NULL
Desc may be set to NULL
Type is function prototype, such as: "VS_CHAR *GetBackImg(VS_INT32 IndexX,VS_INT32 IndexY)"

If StdCallFlag equals to VS_FALSE, then use calling conversion __cdecl, valid for win32
If GlobalFunctionFlag equals VS_TRUE, then when called, Object is not attached as the first parameter.

5.61.30 Create atomic Lua function-CreateAtomicLuaFunction[valid at server]

void *SRPAPI **CreateAtomicLuaFunction**(void *AtomicObject, VS_CHAR *LuaFunctionName , VS_UUID *LuaFunctionID, VS_CHAR *Desc);
AtomicObject: atomic parent object
LuaFunctionName: function name
LuaFunctionID may be set to NULL
Desc may be set to NULL

5.61.31 Create atomic overloading function-CreateAtomicOvlFunction[valid at server]

void *SRPAPI **CreateAtomicOvlFunction**(void *AtomicObject, VS_CHAR *FunctionName, VS_CHAR *OriginFunctionName , VS_UUID *OvlFunctionID, VS_CHAR *Desc , VS_BOOL CantOvl);
AtomicObject: atomic parent object
FunctionName: function name
OriginFunctionName: function to be overloaded.
OvlFunctionID may be set to NULL
Desc may be set to NULL

5.61.32 Create atomic input event-CreateAtomicInEvent[valid at server]

void *SRPAPI **CreateAtomicInEvent**(void *AtomicObject, VS_CHAR *InEventName , VS_CHAR *OutEventName);
AtomicObject:atomic parent object
InEventName: input event name

5.61.33 Createatomic output event-CreateAtomicOutEvent[valid at server]

void *SRPAPI **CreateAtomicOutEvent**(void *AtomicObject, VS_CHAR *OutEventName , VS_UUID *OutEventID, VS_CHAR *Desc, VS_BOOL DynamicFlag);
AtomicObject:atomic parent object
OutEventName: output event name
OutEventID may be set to NULL

Desc may be set to NULL

5.61.34 Simple interface to create atomic object,attribute,function

//------Simple create function, attribute is seperated by ";".
void *SRPAPI **CreateAtomicObjectSimple**(VS_CHAR *SysRootItemName,VS_CHAR *ObjectName,VS_CHAR *Attribute, VS_UUID *ObjectID, VS_CHAR **ErrorInfo)
If SysRootItemName does not exist, then the function creates new one.
Attribute is string to define attribute, such as "VS_CHAR aaa;VS_INT32 bbb;", which may be set to NULL
Local attribute starts with prefix "local", for example"local VS_CHAR aaa;VS_INT32 bbb;".
void *SRPAPI **CreateAtomicStructSimple**(VS_CHAR *StructName,VS_CHAR *Attribute, VS_UUID *ObjectID, VS_CHAR **ErrorInfo);
Attribute is string to define attributes,such as "VS_CHAR aaa;VS_INT32 bbb;"

void *SRPAPI **CreateAtomicObjectAttributeSimple**(void *AtomicObject,VS_CHAR *Attribute, VS_CHAR **ErrorInfo);
Attribute is string to define attributes, such as "VS_CHAR aaa;VS_INT32 bbb;" Local attribute starts with prefix local, for example"local VS_CHAR aaa;VS_INT32 bbb;".
The function returns the input AtomicObject;

void *SRPAPI **CreateAtomicFunctionSimple**(void *AtomicObject, VS_CHAR *FunctionName,VS_CHAR *Attribute, VS_UUID *ObjectID,VS_CHAR **ErrorInfo , VS_BOOL StdCallFlag);
Attribute is function prototype, such as: "VS_CHAR *GetBackImg(VS_INT32 IndexX,VS_INT32 IndexY)"
If StdCallFlag equals to VS_FALSE, then use calling conversion __cdecl, valid for win32
If GlobalFunctionFlag equals VS_TRUE, then when called, Object should not be attached as the first parameter.

Attribute may be signature string, which type string mapping refers to ScriptCall function.
for example:
Atribute: "VS_INT32;VS_FLOAT;" may be use string "if"
Atribute: "VS_INT32 Func(VS_FLOAT)" may be use string "(f)i"

void ***CreateAtomicFunctionSimpleEx**(void *AtomicObject, VS_CHAR *FunctionName,VS_CHAR *Attribute, void *FuncAddress, VS_CHAR **ErrorInfo)
This function is combination of two functions :
CreateAtomicFunctionSimple(AtomicObject, FunctionName, Attribute, NULL, ErrorInfo, false,false);
*SetAtomicFunction(AtomicFunction,(void *)FuncAddress);*

5.61.35 Set function address

void SRPAPI **SetAtomicFunction**(void *AtomicFunction,void *FuncAddress);
VS_BOOL SRPAPI **AtomicAttach**(void *AtomicObject,VS_CHAR *ShareLibName)

AtomicAttach will load sharelib and set address based on function definition. These functions should be global function.

5.61.36 Get atomic function, script, and output event

These functions search input atomic object and its class.
void *SRPAPI **GetAtomicFunction**(VS_UUID *FunctionID);
void *SRPAPI **GetAtomicFunctionEx**(void *AtomicObject,VS_UUID *FunctionID, VS_CHAR **ErrorInfo); //---may be defined in class, and overloaded in this object.
void *SRPAPI **GetAtomicFunctionByName**(void *AtomicObject, VS_CHAR *FunctionName);

void *SRPAPI **GetAtomicScript**(void *AtomicObject, VS_CHAR *ScriptName);
void *SRPAPI **GetAtomicOutEvent**(void *AtomicObject, VS_CHAR *OutEventName);

More information about the returned atomic value may be obtained by function GetAtomicInfo.
VS_BOOL SRPAPI **GetAtomicInfo**(void *Atomic, VS_ULONG *AtomicType, VS_ULONG *Para1, VS_ULONG *Para2, VS_ULONG *Para3, VS_ULONG *Para4, VS_ULONG *Para5 , VS_ULONG *Para6 , VS_ULONG *Para7);
Atomic: may be atomic attribute, function,script, or output event.
For attribute:SRPATOMICQUERYTYPE_ATTRIBUTE
Para1returns Name,Para2 returns Caption,Para3 returns length,Para4 returns type,Para5 returns default value string,Para6 returns address of struct ID,Para7 returns Description
For function:SRPATOMICQUERYTYPE_FUNCTION
Para1 returns FunctionName,Para2 returns 0, can be overloaded; 1,can not be overloaded,Para3 returns 0, normal function; 1 Lua function,Para4 returns 0, normal function; 1, callback function,Para5 returns atomic function being overloaded,Para6 returns address of function pointer,Para7 returns Description.
For script:SRPATOMICQUERYTYPE_SCRIPT
Para1 returns ScriptName,Para2 returns Description,Para3 returns ScriptBuf.
For output event:SRPATOMICQUERYTYPE_OUTEVENT
Para1 returns OutEventName,Para2 returns Description,Para3 returns 0, normal event; 1, dynamic event.
For service item:SRPATOMICQUERYTYPE_SYSROOTITEM
//--Para1 returns SysRootItem name,Para2 returns DependSysRootItem name,Para3 returns SystemRootItemNameID address, Para4 returns SystemRootItemIDaddress
For object:SRPATOMICQUERYTYPE_OBJECT
//-- Para3 returns atomic class object,Para4 returns ObjectName

5.61.37 Get atomic attribute info by name-GetAtomicAttributeInfo

VS_BOOL SRPAPI **GetAtomicAttributeInfo**(void *AtomicObject, VS_INT32 AttributeIndexNumber,OBJECTATTRIBUTEINDEX *AttributeIndex,VS_CHAR *AttributeName,VS_ATTRIBUTEINFO *AttributeInfo);
Get attributes defined in atomic class, also include attributes define by cle.
AttributeName:attribute name
VS_ATTRIBUTEINFO:attribute info
If get attribute defined in the object, then uses AttributeIndexNumber=0, AttributeIndex=NULL.
If get attribute of struct attribute in the object, then uses AttributeIndexNumber=1, AttributeIndex[0]=the struct attribute index in object attributes.

Attribute and type include in different kinds of atomic object:
2. attribute of atomic attribute:
   a. “Caption”,VSTYPE_CHAR,default is “Caption”
   b. “SyncFlag”,VSTYPE_UINT8,default is 0 note:0 global attribute 1 local attribute
   c. “CreateFlag”,VSTYPE_UINT8,default is 0 note:0 not need 1 need
   d. “NotifyFlag”,VSTYPE_UINT8,default is 0 note:1 notify before change 2 notify when change, or sum of them.
   e. “Default”, VSTYPE_CHAR,default is “0”
   f. “Desc”, VSTYPE_CHAR,default is ““
   g. “StaticID”,VSTYPE_ULONG,default is 0
   h. “EditType”, VSTYPE_UINT8,default is 0 note:0 normal edit 1 combobox 3 button 4 check box 5 can not edit 6 mask edit
   i. “EditControl”, VSTYPE_UINT8,default is 0 note:0 none 1 display callback
   j. “EditReadOnly”, VSTYPE_UINT8,default is 0 note:0can edit 1 readonly
   k. “ComboBox”,VS_CHAR,default is ““, otherwise is macro name
3. attribute of other atomic object
   a. “Caption”,VSTYPE_ CHAR,default is “Caption”
   b. “Desc”, VSTYPE_ CHAR,default is ““
4. atomic script
   a. “Desc”,VSTYPE_ CHAR,default is ““
5. atomic macro

a. “Type”, VSTYPE_UINT8,default is 0 note:0 integer 1 float 2 string

6. atomic function
   a. “Desc”,VSTYPE_ CHAR,default is "function description"
   b. “CantOvl”, VSTYPE_UINT8,default is 1, note:0 can be overloaded 1 can not be overloaded
   c. “CallBack”, VSTYPE_BOOL,default is false
   d. “StdCall”, VSTYPE_BOOL,default is false
   e. “Global”, VSTYPE_BOOL,default is false
7. Atomic output event
   a. “Desc”,VSTYPE_ CHAR,default is "event description"
   b. “Dynamic”, VSTYPE_BOOL,default is false
8. Atomic module or edit module
   a. “Type”, VSTYPE_UINT16,default is 0
9. Atomic scruct
   a. “Caption”, VSTYPE_ CHAR,default is "Struct"
10. Atomic service
    a. “FrameTicket”, VSTYPE_INT32,default is 5
    b. “NetPkgSize”, VSTYPE_INT32,default is 10240
    c. “UpLoadPkgSize”, VSTYPE_INT32,default is 2048
    d. “DownLoadPkgSize”, VSTYPE_INT32,default is 2048
    e. “DataUpPkgSize”, VSTYPE_INT32,default is 2048
    f. “DataDownPkgSize”, VS_INT32,default is 2048
11. Atomic object
    a. “SysEvent”, VSTYPE_BOOL,default is false
    b. “SpecialEvent”, VSTYPE_UINT8,default is 0

```
#define VSSYSEVENT_PROCESS_TICKET          0x0001
#define VSSYSEVENT_PROCESS_FRAMETICKET     0x0002
#define VSSYSEVENT_PROCESS_IDLE            0x0004
#define VSSYSEVENT_PROCESS_APPACTIVE       0x0008
#define VSSYSEVENT_PROCESS_APPDEACTIVE     0x0010
#define VSSYSEVENT_PROCESS_SERVICEACTIVE   0x0020
#define VSSYSEVENT_PROCESS_SERVICEDEACTIVE 0x0040
```

    c. “ActiveCmd”, VSTYPE_UINT8,default is 0

```
#define VSACTIVE_ALONE     0
#define VSACTIVE_FOLLOW    1
```

    d. “SaveFlag”, VSTYPE_UINT8,default is

```
#define VSSTATIC_SAVE      0
#define VSSTATIC_CLIENTSAVE  1
#define VSSTATIC_NONE      2
```

    e. “SyncGroup”, VS_ULONG,default is 0

5.61.38 Get atomic attribute info by index-GetAtomicAttributeInfoEx

VS_BOOL SRPAPI **GetAtomicAttributeInfoEx** (void *AtomicObject, VS_INT32 AttributeIndexNumber,OBJECTATTRIBUTEINDEX *AttributeIndex, OBJECTATTRIBUTEINDEX This AtomicAttributeIndex,VS_ATTRIBUTEINFO *AttributeInfo);
ThisAtomicAttributeIndex: attribute index
VS_ATTRIBUTEINFO:attribute info
If get attribute defined in the object, then uses AttributeIndexNumber=0, AttributeIndex=NULL.
If get attribute of struct attribute in the object, then uses AttributeIndexNumber=1, AttributeIndex[0]=the struct attribute index in object attributes.

5.61.39 Get atomic attach attribute number and infomation-GetAtomicAttachAttributeInfoEx

VS_INT32 SRPAPI **GetAtomicAttachAttributeNumber**(void *AtomicObject);
VS_INT32 SRPAPI **GetAtomicAttachAttributeSize**(void *AtomicObject);

VS_BOOL SRPAPI **GetAtomicAttachAttributeInfoEx**(void *AtomicObject,OBJECTATTRIBUTEINDEX AttachAttributeIndex,VS_ATTRIBUTEINFO *AttributeInfo);
Get attach attribute info defined in object or its class. If returns VS_FALSE, then attribute does not exist.

5.61.40 Get atomic macro

void *SRPAPI **QueryFirstAtomicMacro**(VS_ULONG *QueryContext, VS_UUID *ServiceID, VS_CHAR **MacroName, VS_UINT8 *Type);
void *SRPAPI **QueryNextAtomicMacro**(VS_ULONG *QueryContext, VS_UUID *ServiceID, VS_CHAR **MacroName, VS_UINT8 *Type);
if ServiceID is not NULL, then query macro of the corresponding service .
After get the atomic macro, then can get atomic macro item using function **QueryFirstAtomicInfo and QueryNextAtomicInfo.**

5.61.41 Query first and next atomic object-QueryFirstAtomicInfo,QueryNextAtomicInfo

void *SRPAPI **QueryFirstAtomicInfo**(VS_ULONG *QueryContext, VS_UINT8 AtomicType, VS_ULONG *Para1, VS_ULONG *Para2, VS_ULONG *Para3, VS_ULONG *Para4, VS_ULONG *Para5);
AtomicType defines as follows:

```
#define SRPATOMICQUERYTYPE_SYSROOTITEM   0     //--Para1 returns SysRootItem name,Para2 returns DependSysRootItem name,Para3 returns SystemRootItemNameID address.
#define SRPATOMICQUERYTYPE_MACRO         1     //--Para1 returns MacroName,Para2 returns Type.
#define SRPATOMICQUERYTYPE_MACROITEM     2     //--Para1 is atomic macro,Para2 returns MacroItemName,Para3 returns MacroItemValue.
#define SRPATOMICQUERYTYPE_MODULE        3     //--Para1 returns ModuleName.
#define SRPATOMICQUERYTYPE_EDITMODULE    4     //--Para1 returns ModuleName.
#define SRPATOMICQUERYTYPE_STRUCT        5     //--Para1 returns StructName,Para2 returns StructCaption,Para3 is input,if(*Para3) == NULL, indicates this service, else is service ID
#define SRPATOMICQUERYTYPE_OBJECT        6     //--Para1 is atomic parent object,Para2 is attribute index.,Para 3 returns atomic class object,Para4 returns ObjectName.
#define SRPATOMICQUERYTYPE_ATTACHATTRIBUTE  7  //--Para1is atomic parent object,Para2 returns AttributeName,Para3 returnstype,Para4 returns length,Para5 returns atomic struct (if exist)
#define SRPATOMICQUERYTYPE_ATTRIBUTE     8     //--Para1is atomic parent object,Para2 returns AttributeName,Para3 returns type,Para4 returns length,Para5 returns atomic struct
#define SRPATOMICQUERYTYPE_FUNCRETATTRIBUTE  9     //--Para1 is atomic function,Para2 returns AttributeName,Para3 returns type,Para4 returns length,Para5 returns atomic atruct (if exist)
#define SRPATOMICQUERYTYPE_FUNCPARAATTRIBUTE  10     //--Para1 is atomic function,Para2 returns AttributeName,Para3 returns type,Para4 returns length,Para5 returns atomic struct (if exist)
#define SRPATOMICQUERYTYPE_STRUCTATTRIBUTE  11     //--Para1 is atomic struct,Para2 returns AttributeName,Para3 returns type,Para4 returns length,Para5 returns Offset
#define SRPATOMICQUERYTYPE_SCRIPT        12     //--Para1is atomic parent object,Para2 returns ScriptName,Para3 returns ScriptBuf
#define SRPATOMICQUERYTYPE_FUNCTION      13     //--Para1is atomic parent object,Para2 returns FunctionName,Para3 returns 0, for normal function, 1for Lua function,Para4 returns atomic function being overloaded
#define SRPATOMICQUERYTYPE_INEVENT       14     //--Para1is atomic parent object,Para2 returns InEventName,Para3 returns OutEventName
#define SRPATOMICQUERYTYPE_OUTEVENT      15     //--Para1is atomic parent object,Para2 returns OutEventName,Para3 returns Description,Para4 returns 0 normal event; 1  dynamic event.
```

5.61.42 Whether object can be output-AtomicObjectCanOutput

VS_BOOL SRPAPI **AtomicObjectCanOutput**(void *AtomicObject, VS_INT32 HasOutputNumber, void **HasOutputAtomicObject)

In cle, object is also class. Therefore threr are relations between object and class. When parse, should first parse class, and then object.
The function is determine object can be output, Input parameter is list of object has been output.

5.61.43 Whether pointer queue can be output -AtomicObjectAttributeCanOutput

VS_BOOL SRPAPI **AtomicObjectAttributeCanOutput**(void *AtomicObject, OBJECTATTRIBUTEINDEX AtomicAttributeIndex,VS_INT32 HasOutputNumber, void **HasOutputAtomicObject)

5.61.44 Set atomic attribute value-SetAtomicAttribute[valid at server]

VS_BOOL SRPAPI **SetAtomicAttribute**(void *AtomicObject, VS_INT32 AttributeIndexNumber,OBJECTATTRIBUTEINDEX *AttributeIndex,OBJECTATTRIBUTEINDEX This AtomicAttributeIndex, VS_INT8 *NewValue);;
ThisAtomicAttributeIndex: global attribute index
NewValue: attribute new value
If attribute defined in the object, then uses AttributeIndexNumber=0, AttributeIndex=NULL.
If attribute defined in struct attribute of the object, then uses AttributeIndexNumber=1, AttributeIndex[0]=the struct attribute index in object attributes, and so on.

5.61.45 Get atomic attribute value-GetAtomicAttribute

void *SRPAPI **GetAtomicAttribute** (void *AtomicObject, VS_INT32 AttributeIndexNumber,OBJECTATTRIBUTEINDEX *AttributeIndex,OBJECTATTRIBUTEINDEX This AtomicAttributeIndex);
ThisAtomicsAttributeIndex global attribute index
If attribute defined in the object, then uses AttributeIndexNumber=0, AttributeIndex=NULL.
If attribute defined in struct attribute of the object, then uses AttributeIndexNumber=1, AttributeIndex[0]=the struct attribute index in object attributes, and so on.

5.61.46 Get atomic attribute default value-GetAtomicAttributeDefault

void *SRPAPI **GetAtomicAttributeDefault** (void *AtomicObject, VS_INT32 AttributeIndexNumber,OBJECTATTRIBUTEINDEX *AttributeIndex,OBJECTATTRIBUTEINDEX This AtomicAttributeIndex);
ThisAtomicsAttributeIndex global attribute index
If attribute defined in the object, then uses AttributeIndexNumber=0, AttributeIndex=NULL.
If attribute defined in struct attribute of the object, then uses AttributeIndexNumber=1, AttributeIndex[0]=the struct attribute index in object attributes, and so on.

5.61.47 Get and set atomic object syncgroup-SetAtomicObjectSyncGroup[valid at server], GetAtomicObjectSyncGroup

VS_SYNCGROUP SRPAPI **GetAtomicObjectSyncGroup**(void *AtomicObject);
VS_BOOL SRPAPI **SetAtomicObjectSyncGroup**(void *AtomicObject, VS_SYNCGROUP SyncGroup);

5.61.48 Get and set atomic object attribute-SetAtomicObjectAttribute[valid at server], GetAtomicObjectAttribute

VS_BOOL SRPAPI GetAtomicObjectAttribute(void *AtomicObject,VS_BOOL *SysEvent,VS_UINT8 *SpecialEvent,VS_UINT8 *ActiveCmd,VS_UINT8 *SaveFlag);
VS_BOOL SRPAPI SetAtomicObjectAttribute(void *AtomicObject,VS_BOOL SysEvent,VS_UINT8 SpecialEvent,VS_UINT8 ActiveCmd,VS_UINT8 SaveFlag);

5.62 Whether the interface is valid-IsValid

VS_BOOL SRPAPI **IsValid**();
If returns VS_TRUE, then the interface is valid, or else the service has been unloaded.

5.63 Other interface

5.63.1 Get Lock Interface-GetSRPLockInterface

class ClassOfSRPLockInterface *SRPAPI **GetSRPLockInterface**(void);

5.63.2 MD5 string to UUID-MD5ToUuid, UuidToMD5

VS_BOOL SRPAPI MD5ToUuid(VS_INT8 *String,VS_UUID *Uuid);
VS_INT8 *SRPAPI UuidToMD5(VS_UUID *Uuid);

5.64 BinBuf interface

5.64.1 Get binbuf interface-GetSRPBinBufInterface

class ClassOfSRPBinBufInterface *SRPAPI **GetSRPBinBufInterface**(void);

5.64.2 Push binbuf interface to lua stack-LuaPushBinBuf

VS_BOOL SRPAPI **LuaPushBinBuf**(class ClassOfSRPBinBufInterface *BinBuf , VS_BOOL AutoRelease);
AutoRelease = true, the function passes address of BinBuf to lua script, and lua is responsible for releasing the pointer. If ==false,Lua is not responsible for releasing the interface.

5.64.3 Get binbuf from lua stack-LuaToBinBuf

class ClassOfSRPBinBufInterface *SRPAPI **LuaToBinBuf**(VS_INT32 Index);

5.64.4 Whether value in lua stack is binbuf -LuaIsBinBuf

VS_BOOL SRPAPI LuaIsBinBuf(VS_INT32 Index);;

5.65 SXml,FunctionPara,CommInterface interface

VS_BOOL SRPAPI LuaPushSXml(class ClassOfSRPSXMLInterface *SXml, VS_BOOL AutoRelease);
class ClassOfSRPSXMLInterface *SRPAPI LuaToSXml(VS_INT32 Index);
VS_BOOL SRPAPI LuaIsSXml(VS_INT32 Index);

VS_BOOL SRPAPI LuaPushFunctionPara(class ClassOfSRPFunctionParaInterface *FunctionPara, VS_BOOL AutoRelease);
class ClassOfSRPFunctionParaInterface *SRPAPI LuaToFunctionPara(VS_INT32 Index);
VS_BOOL SRPAPI LuaIsFunctionPara(VS_INT32 Index);

VS_BOOL SRPAPI LuaPushCommInterface(class ClassOfSRPCommInterface *CommInterface, VS_BOOL AutoRelease);
class ClassOfSRPCommInterface *SRPAPI LuaToCommInterface(VS_INT32 Index);
VS_BOOL SRPAPI LuaIsCommInterface(VS_INT32 Index);

5.66 Whether Lua function is defined and create lua script of object

5.66.1 Whether lua function is defined-DefLuaFunction

VS_BOOL SRPAPI DefLuaFunction(void *Object, VS_CHAR *ScriptName);

5.66.2 Create lua script-SaveToLuaFunc

VS_BOOL SRPAPI SaveToLuaFunc(void *Object, VS_CHAR *LuaFileName, VS_CHAR *FuncName);
Generate lua script of object attribute, name value, and name script.

5.67 Generate or execute object init script

Sentences are seperated by ';'
for each sentence,
If the first char is'$', the the following '$O' will be replaced by Object before executed.
For example:"$$O:_Active();" is translated to Object:_Active() .
If not, then the following rule takes effect.
If contains '=', the translate to Object. XXXX
otherwise translate to Object: XXXX
$O is not case sensitive.
For example:
"attr1=value; attr2=value2; func();"

5.67.1 Execute object init script-LuaInitObject

void SRPAPI LuaInitObject(void *Object,VS_CHAR *InitScript);

5.67.2 Generate object init script-GetAttributeLuaString

VS_CHAR *SRPAPI GetAttributeLuaString(void *Object);

5.68 Create UUID and temporary directory

5.68.1 Get UUID-CreateUuid

void SRPAPI CreateUuid(VS_UUID *UuidPtr);

5.68.2 Get temporary directory-GetSRPTempPath

void SRPAPI GetSRPTempPath(VS_ULONG BufSize,VS_CHAR *Buf);

5.68.3 Get config directory-GetSRPConfigurePath

void SRPAPI GetSRPConfigurePath(VS_ULONG BufSize,VS_CHAR *Buf);

5.69 Register DLL callback

5.69.1 Register DLL callback-RegDllCallBack

void SRPAPI RegDllCallBack(VS_MsgCallBackProc MsgCallBackProc, VS_ULONG MsgCallBackPara)
callback prototype :
typedef VS_ULONG (SRPAPI *VS_MsgCallBackProc)(VS_ULONG ServiceGroupID, VS_ULONG uMsg, VS_ULONG wParam, VS_ULONG lParam, VS_BOOL &IsProcessed, VS_ULONG Para);

5.69.2 Unregister DLL callback-UnRegDllCallBack

void SRPAPI UnRegDllCallBack(VS_MsgCallBackProc MsgCallBackProc, VS_ULONG MsgCallBackPara)
callback prototype :
typedef VS_ULONG (SRPAPI *VS_MsgCallBackProc)(VS_ULONG ServiceGroupID, VS_ULONG uMsg, VS_ULONG wParam, VS_ULONG lParam, VS_BOOL &IsProcessed, VS_ULONG Para);

5.70 Insert and remove element from Lua table

5.70.1 Insert element to lua table-LuaInsertTable

void SRPAPI LuaInsertTable(VS_INT32 TableIndex,VS_INT32 Pos);
Insert element on top of lua statck to lua table.
If Pos<=0, then indicates insert to tail, or else insert to Pos.

5.70.2 Remove element from lua table-LuaRemoveTable

void SRPAPI LuaRemoveTable(VS_INT32 TableIndex,VS_INT32 Pos);
If Pos<=0, then indicates remove from tail, or else remove element at the Pos.

5.70.3 Get Lua object length -LuaObjLen

VS_INT32 SRPAPI LuaObjLen(VS_INT32 TableIndex);

5.70.4 Get element from Lua table-LuaGetTablei

void SRPAPI LuaGetTablei(VS_INT32 TableIndex,VS_INT32 Pos);
If Pos<=0, then indicates get element from tail.

5.70.5 Set element to lua table-LuaSetTablei

void SRPAPI LuaSetTablei(VS_INT32 TableIndex,VS_INT32 Pos);
the function pop element from lua stack.
If Pos<=0, then indicates set element at tail.

5.71 Restart interface

5.71.1 Restart application

VS_BOOL SRPAPI ProgramRestart();
The function is valid only when cle is started by manager program.

5.72 http/ftp download-HttpDownLoad

VS_BOOL SRPAPI HttpDownLoad(VS_UUID *AttachObjectID VS_CHAR *ServerUrl,VS_CHAR *ClientPath,VS_CHAR *FileName, VS_FileUpDownLoadCallBackProc CallBackProc, VS_UUID *ObjectID, VS_ULONG Para, VS_BOOL SaveFileFlag)
ObjectID may be set to NULL
VS_UUID *AttachObjectID may be set to NULL
SaveFileFlag:==true, save to file； or else, not save
HttpDownLoad(NULL,"http://www.srplab.com/Files","e:","srirrlicht_index.htm",NULL,NULL,0,VS_TRUE);

void SRPAPI HttpDownLoadAbort( );
Cancel all Http/FTPdownload.

5.73 Get static data

VS_INT8 *SRPAPI GetStaticDataEx(void *Object,VS_ULONG UniqueDataUnitID,VS_STATICID *DataVersion,VS_ULONG *DataSize, VS_BOOL AutoDownLoad,VS_CHAR *Token);

5.74 Memory file manager

5.74.1 Set environment memory file-SetEnvMemoryFile

void SRPAPI SetEnvMemoryFile(class ClassOfSRPMemoryFileInterface *MemoryFile);
MemoryFile will be freed by cle

5.75 LockGC/UnLockGC

There is a counter for each object. when object is created by script, the counter will be decreased by 1 after objectis garbage collected, and object will be freed if counter less than 0.

5.75.1 Increate object counter-LockGC

VS_BOOL SRPAPI LockGC(void *Object);
Max value is 255

5.75.2 Decreate object counter-UnLockGC

VS_BOOL SRPAPI UnLockGC(void *Object);

5.76 Object Reference Count

```
void SRPAPI AddRefEx(void *Object);
void SRPAPI DelRefEx(void *Object);
VS_INT32 SRPAPI GetRefEx(void *Object);
```

AddRefEx is same as LockGC
DelRefEx is same as UnLockGC

VS_CHAR *SRPAPI GetRefInfo(void *Object)
object is referenced by scripts, return value is script name string, separated by “,”
if object is SLockGC by the script, then there has a “*” prefix before the script name.
for example:
“lua,*python”

VS_BOOL SRPAPI ReleaseOwnerEx(void *Object);
This function should be used when cle object is create by one language and will not used in this language and output to other languages
For c/c++:
This function decreases reference count of object,but not cause object to be freed.

VS_BOOL SRPAPI ReleaseOwnerExForScript(VS_CHAR *ScriptInterface,void *Object)
void SRPAPI CaptureOwnerExForScript(VS_CHAR *ScriptInterface,void *Object)
The above two function should be used in script bridge module. When _ ReleaseOwnerEx function is called, the bridge call ReleaseOwnerExForScript with interface name.
If _SUnLockGC is called, script bridge should call CaptureOwnerExForScript to clear record before.

void SUnLockGC()
This function is provided for check all cle object’s with ReleaseOwnerEx record and trigger _SUnLockGC function of the corresponding script interface.

5.77 Get IP address

5.77.1 Get IP address-GetPeerIP

VS_BOOL SRPAPI GetPeerIP(VS_ULONG ClientID,VSSOCKADDR_IN *ClientIP);
At client, ClientID is ignored

5.78 Get server ID at client

5.78.1 Get server ID-GetServerID

VS_ULONG SRPAPI GetServerID();
Valid for client, the ID may be used for communicating between client and server.

5.79 Communication between client and server

5.79.1 Send message-RemoteSend

VS_BOOL RemoteSend(void *Object, VS_ULONG ClientID, class ClassOfSRPParaPackageInterface *ParaPkg)
At server, If ClientID equals to 0, then server sends message to all clients. In this case, the object must be global object.If is not 0, the server sends message to specific client.
At client, ClientID is ignored.
At receiver side, the object event VSEVENT_SYSTEMEVENT_ONREMOTESEND will be generated, application can get the message in the event handler.

5.80 Force to save service static data(valid at server)

5.80.1 Force to save static data-ForceToSaveStatic

void SRPAPI ForceToSaveStatic()
When service exits, if the function has been called, then static data of all objets will be saved.

5.81 Clear expired static data

5.81.1 clear expired static data-ClearStatic

void SRPAPI ClearStatic (VS_UINT16 BeforeDays);

5.82 Get static data version

5.82.1 Get static data version-GetStaticVersion

void SRPAPI GetStaticVersion(VS_ULONG DataSize,VS_INT8 *DataBuf,VS_STATICID *RetDataVersion);

5.83 Monitor Http download at server

5.83.1 monitor Http download-RegWebDownFunction/ UnRegWebDownFunction

void SRPAPI RegWebDownFunction(VS_WebDownInfoProc CallBackProc,VS_ULONG Para)
void SRPAPI UnRegWebDownFunction(VS_WebDownInfoProc CallBackProc,VS_ULONG Para)
typedef void (SRPAPI * VS_WebDownInfoProc)(VS_ULONG Para, VS_ULONG uMes, VS_CHAR *FileName, VS_ULONG MaxSize, VS_ULONG CurSize);
uMes:

```
#define VSFILE_ONDOWNSTART      0    //---start download
#define VSFILE_ONDOWNPROGRESS   1    //---download progress
#define VSFILE_ONDOWNFINISH     2    //---finish
#define VSFILE_ONDOWNERROR      3    //---error
```

5.83.2 Display download information-WebDownPrint

void SRPAPI WebDownPrint(VS_ULONG uMes, VS_CHAR *FileName, VS_ULONG MaxLength, VS_ULONG CurLength)
Information will be printed in above callback function

5.84 Get object from Lua string

5.84.1 Get object from Lua string-GetObjectFromLua

void *SRPAPI GetObjectFromLua(VS_CHAR *String);
Format of string is such as “Service.DriveClass”

5.85 System Doc object

Doc object is instance of VSSYSDOC_CLASSID

5.85.1 Get Doc class-GetSysDocClass

void *SRPAPI GetSysDocClass();

5.85.2 Get first registered Doc object-FirstDoc

void *SRPAPI FirstDoc(VS_QUERYRECORD *QueryRecord,VS_CHAR **DocName);

5.85.3 Get next registered Doc object -NextDoc

void *SRPAPI NextDoc(VS_QUERYRECORD *QueryRecord,VS_CHAR **DocName);

5.85.4 register Doc object-RegisterDoc

void SRPAPI RegisterDoc(void *DocObject,VS_CHAR *DocName);

5.85.5 Unregister Doc object-UnRegisterDoc

void SRPAPI UnRegisterDoc(void *DocObject,VS_CHAR *DocName);

5.85.6 Trigger Doc event-ProcessSysDocEvent

VS_EVENTPARAM_RUNPARAM *SRPAPI ProcessSysDocEvent(VS_UUID *DocObjectID,VS_UUID *EventID,VS_EVENTPARAM_RUNPARAM *RequestParam);
VSSYSDOC_ONGETTEXT or VSSYSDOC_ONSETTEXT

5.85.7 Register or unregister Doc event-RegDocEventFunction/ UnRegDocEventFunction

VS_BOOL SRPAPI RegDocEventFunction(VS_UUID *DocObjectID,VS_UUID *EventID, void *FuncAddr,VS_ULONG Para);
void SRPAPI UnRegDocEventFunction(VS_UUID *DocObjectID,VS_UUID *EventID, void *FuncAddr,VS_ULONG Para);

5.86 Pack static data file

5.86.1 Pack static data file-PackStaticData

void SRPAPI PackStaticData()
Discard redundancy space

5.87 Insert Lua table

5.87.1 Insert Lua table-LuaInsertTable2/ LuaRemoveTable2

VS_INT32 SRPAPI LuaInsertTable2(VS_INT32 TableIndex);
void SRPAPI LuaRemoveTable2(VS_INT32 TableIndex,VS_INT32 Pos);

5.88 Garbage collect

5.88.1 garbage collect-GCCollect

void SRPAPI GCCollect();

5.89 Object temporary table and gc lock

5.89.1 object temporary table-LuaObjectNewTempTable

VS_BOOL SRPAPI LuaObjectNewTempTable(void *Object, VS_CHAR *Name);

5.89.2 Lock lua garbage collect-LuaObjectIsLock/ LuaObjectLock/ LuaObjectUnLock

VS_BOOL SRPAPI LuaObjectIsLock(void *Object);
void SRPAPI LuaObjectLock(void *Object);
void SRPAPI LuaObjectUnLock(void *Object);

If service defines lua attributes and functions of object, the object is locked automaticly.

5.90 Get Lua variable

5.90.1 Get Lua variable-GetValueFromLua

VS_BOOL SRPAPI GetValueFromLua(VS_CHAR *String);
Format of string is such as “a.b.c.d ”
After the function is called, value on top of lua stack is the return value, which should be popped up by the caller.

5.91 Set attach class[reserved]

5.91.1 register attach class-RegisterAttachClass

void SRPAPI RegisterAttachClass(void *OriginClass,void *AttachClass)

5.91.2 unregister attach class -UnRegisterAttachClass

void SRPAPI UnRegisterAttachClass(void *OriginClass,void *AttachClass)

Used for extend function of origin class.
The attach class does not support overload and remotecall.

5.91.3 Garbage collect-GCCollect

void SRPAPI GCCollect();

5.92 ClipperBoard[Windows]

5.92.1 Copy string to clipboard-ToClipBoard

void SRPAPI ToClipBoard(VS_CHAR *Info);

5.92.2 copy string from clipboard-FromClipBoard

VS_CHAR *SRPAPI FromClipBoard();

5.93 Windowless mode function [reserved]

5.93.1 Windowless mode function

void SRPAPI Windowless_Redraw(VS_BOOL fErase); Require to refresh.
VS_BOOL SRPAPI IsWindowlessTransparent( ) ;

5.93.2 Capture and release DC

void SRPAPI Windowless_GetDC(void **hDC,VS_RECT *rEct);
void SRPAPI Windowless_ReleaseDC(void *hDC);

5.94 Predefined object ID

VS_UUID *SRPAPI GetVSObjectID(VS_INT32 Which);

Which takes value from :
#define VSSYSID_VSSYSOBJ_OBJID 0
#define VSSYSID_VSSYSOBJ_WNDADJUST 1
#define VSSYSID_VSSYSOBJ_WNDCANBERESIZE 2
#define VSSYSID_VSSYSOBJ_WNDRESIZE 3
#define VSSYSID_VSSYSOBJ_EDITSELECT 4
#define VSSYSID_VSSYSOBJ_SETFOCUS 5
#define VSSYSID_VSSYSOBJ_WNDMSG 6

#define VSSYSID_VSSYSDOC_CLASSID 7
#define VSSYSID_VSSYSDOC_ONGETTEXT 8
#define VSSYSID_VSSYSDOC_ONSETTEXT 9
#define VSSYSID_VSSYSDOC_LUA_GETTEXT 10
#define VSSYSID_VSSYSDOC_LUA_SETTEXT 11
#define VSSYSID_VSSYSDOC_ONTEXTCHANGE 12
#define VSSYSID_VSSYSDOC_ONTEXTSELECT 13

5.95 Get interface

5.95.1 Get SXML interface

class ClassOfSRPSXMLInterface *GetSXMLInterface();

5.95.2 Get FunctionPara interface

class ClassOfSRPFunctionParaInterface *SRPAPI GetFunctionParaInterface()

5.95.3 Get communication interface

class ClassOfSRPCommInterface *SRPAPI GetCommInterface()

5.96 Temporary file register

VS_BOOL SRPAPI RegTempFile(VS_CHAR *TempFileName,VS_CHAR *OriFileName); /*clear when process not exist*/
OriFileName: may be NULL
TempFileName: Is unique among multiple processes. TempFileName should be full path name.

VS_CHAR *SRPAPI GetRegTempFile(VS_CHAR *OriFileName ,VS_CHAR *Buf,VS_INT32 BufSize);
Obtain temporary file name registered by other process.
If there is, the filename is automatically registered for the calling process.

void SRPAPI UnRegTempFile(VS_CHAR * TempFileName); /*clear when process not exist*/
Unregister the temporary file. The file will be deleted by CLE.

5.97 Interact with environment[reserved]

void SRPAPI SetRunEnv_FromChildCallBack(void *Object,VS_RunEnvCallBackProc CallBack,VS_ULONG Para);
void SRPAPI SetRunEnv_FromParentCallBack(void *Object,VS_RunEnvCallBackProc CallBack,VS_ULONG Para);
//---object capture runenv event
void SRPAPI RegRunEnv_FromParentCallBack(void *Object, void *ParentObject, VS_RunEnvCallBackProc CallBack,VS_ULONG Para);
void SRPAPI UnRegRunEnv_FromParentCallBack(void *Object, void *ParentObject, VS_RunEnvCallBackProc CallBack,VS_ULONG Para);
//---real function
VS_BOOL SRPAPI RunEnvToChild(void *Object, void *DesObject, struct StructOfVSRunEnv *RunEnvInfo);
VS_BOOL SRPAPI RunEnvToParent(void *Object, struct StructOfVSRunEnv *RunEnvInfo);

5.98 Lock lua table

VS_BOOL SRPAPI LockLuaTable( );
VS_BOOL SRPAPI UnLockLuaTable( );
After lock, lua table only can be read. Table includes global table and object.

5.99 Is root service

VS_BOOL SRPAPI IsRootService();

The service is dynamic service which is loaded by import function, the return value is False,orelse returns true.

5.100 Get environment memory file-GetEnvMemoryFile

class ClassOfSRPMemoryFileInterface *SRPAPI GetEnvMemoryFile();
Returned pointer should not be released.

5.101 Get service interface of object

class ClassOfSRPInterface *SRPAPI GetSRPInterface(void *Object);
class ClassOfSRPInterface *SRPAPI GetSRPInterfaceEx(VS_UUID *ObjectID);

5.102 Object EditLog/CheckPoint[reserved]

With the checkpoint mechanism to provide high reliability.

1. **void SRPAPI SetLog(void *Object,VS_BOOL Flag)**
Set the object or service item to start to log, There after, any changes of object attribute, child object created or deleted, child object attribute changed will be recorded in log file, which can be recover by ApplyLog.
2. **void SRPAPI SetLogFile(VS_CHAR *FileName)**
Set Log file name, default is in service directory.
3. **VS_CHAR *SRPAPI GetLogFile()**
Get log file name.
4. **void SRPAPI ClearLog()**
Clear Log content.
5. **VS_BOOL SRPAPI ApplyLog()**
Recover.

5.103 Object attribute getset and function call general interface

Support attributes and functions defined in c/c++ or script. Here provides a general interface to get or set value, or call function.

VS_ULONG SRPAPI **ScriptCall**(void *Object, VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,...);
VS_ULONG SRPAPI **ScriptCallVar**(void *Object, VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,va_list argList);
VS_FLOAT SRPAPI **ScriptFCall**(void *Object, VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,...);
VS_FLOAT SRPAPI **ScriptFCallVar**(void *Object, VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,va_list argList);

VS_BOOL SRPAPI **ScriptRCall**(VS_ULONG ClientID,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,...);
VS_BOOL SRPAPI **ScriptRCallVar**(VS_ULONG ClientID,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,va_list argList);
VS_BOOL SRPAPI **ScriptRCallEx**(VS_ULONG ExcludeClientID,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,...);
VS_BOOL SRPAPI **ScriptRCallExVar**(VS_ULONG ExcludeClientID,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,va_list argList);
VS_ULONG SRPAPI **ScriptSRCall**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,...);

VS_ULONG SRPAPI **ScriptSRCallVar**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,va_list argList);
VS_FLOAT SRPAPI **ScriptFSRCall**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,...);
VS_FLOAT SRPAPI **ScriptFSRCallVar**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,va_list argList);

//In the following four function, return type of RetType is ignored, and the type is returned in RetType
VS_ULONG SRPAPI **ScriptCall2**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,VS_UINT8 *RetType,...);
VS_ULONG SRPAPI **ScriptCallVar2**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,VS_UINT8 *RetType,va_list argList);
VS_ULONG SRPAPI **ScriptSRCall2**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,VS_UINT8 *RetType,...);
VS_ULONG SRPAPI **ScriptSRCallVar2**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,VS_UINT8 *RetType,va_list argList);

VS_INT64 SRPAPI **ScriptCallInt64**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,...);
VS_INT64 SRPAPI **ScriptCallInt64Var**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,va_list argList);
VS_DOUBLE SRPAPI **ScriptCallDouble**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,...);
VS_DOUBLE SRPAPI **ScriptCallDoubleVar**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,va_list argList);

VS_INT64 SRPAPI **ScriptSRCallInt64**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,...);
VS_INT64 SRPAPI **ScriptSRCallInt64Var**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,va_list argList);
VS_DOUBLE SRPAPI **ScriptSRCallDouble**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,...);
VS_DOUBLE SRPAPI **ScriptSRCallDoubleVar**(VS_ULONG WaitTime,VS_ULONG ClientID,VS_ULONG *RetCode,void *Object, VS_CHAR *ScriptName,VS_CHAR *TypeSet,va_list argList);

TypeSet map :

z	VSTYPE_BOOL
b	VSTYPE_INT8
B	VSTYPE_UINT8
c	VSTYPE_INT8
h	VSTYPE_INT16
H	VSTYPE_UINT16
i	VSTYPE_INT32
I	VSTYPE_UINT32
l	VSTYPE_LONG
L	VSTYPE_ULONG
j/J	VSTYPE_INT64
f/F	VSTYPE_FLOAT
d/D	VSTYPE_DOUBLE
s/S	VSTYPE_CHARPTR
p/P	VSTYPE_PARAPKGPTR
r/R	VSTYPE_BINBUFPTR
o/O	VSTYPE_OBJPTR
v/V	VSTYPE_VOID

Such as:TypeStr : "(iis)s", "()s", "()v", "()"

```
   //----get object attributes
      VS_BOOL SRPAPI ScriptSetBool(void *Object,VS_CHAR *AttributeName,VS_BOOL Value);
      VS_BOOL SRPAPI ScriptSetInt(void *Object,VS_CHAR *AttributeName,VS_INT32 Value);
      VS_BOOL SRPAPI ScriptSetNumber(void *Object,VS_CHAR *AttributeName,VS_FLOAT Value);
      VS_BOOL SRPAPI ScriptSetStr(void *Object,VS_CHAR *AttributeName,VS_CHAR *Value);
      VS_BOOL SRPAPI ScriptSetObject(void *Object,VS_CHAR *AttributeName,VS_UINT8
Type,VS_ULONG Value);

      VS_BOOL SRPAPI ScriptGetBool(void *Object,VS_CHAR *AttributeName);
      VS_INT32 SRPAPI ScriptGetInt(void *Object,VS_CHAR *AttributeName);
      VS_FLOAT SRPAPI ScriptGetNumber(void *Object,VS_CHAR *AttributeName);
      VS_CHAR *SRPAPI ScriptGetStr(void *Object,VS_CHAR *AttributeName);
   VS_ULONG SRPAPI ScriptGetObject(void *Object,VS_CHAR *AttributeName,VS_UINT8 *RetType);

      VS_BOOL SRPAPI ScriptSetInt64(void *Object,VS_CHAR *AttributeName,VS_INT64 Value);
      VS_BOOL SRPAPI ScriptSetDouble(void *Object,VS_CHAR *AttributeName,VS_DOUBLE Value);

      VS_INT64 SRPAPI ScriptGetInt64(void *Object,VS_CHAR *AttributeName);
      VS_DOUBLE SRPAPI ScriptGetDouble(void *Object,VS_CHAR *AttributeName);

      VS_BOOL SRPAPI ScriptSetInt64Index(void *Object,VS_INT32 Index,VS_INT64 Value);
      VS_BOOL SRPAPI ScriptSetDoubleIndex(void *Object,VS_INT32 Index,VS_DOUBLE Value);

      VS_INT64 SRPAPI ScriptGetInt64Index(void *Object,VS_INT32 Index);
   VS_DOUBLE SRPAPI ScriptGetDoubleIndex(void *Object,VS_INT32 Index);

for ScriptSetObject, Type supported:
#define VSTYPE_BOOL
#define VSTYPE_INT32
#define VSTYPE_VSTRING    input type is VS_CHAR*
#define VSTYPE_RECT
#define VSTYPE_FONT
#define VSTYPE_TIME
#define VSTYPE_OBJPTR
#define VSTYPE_PARAPKGPTR   // for set, uses LuaPushParaPkg(parapkg,VS_FALSE)
#define VSTYPE_BINBUFPTR      // for set, uses LuaPushBinBuf(binbuf,VS_FALSE)

for VSTYPE_FLOAT, ScriptSetNumber should be used

for ScriptGetObject, Type supported:

#define VSTYPE_BOOL
#define VSTYPE_INT32
#define VSTYPE_FLOAT
#define VSTYPE_VSTRING    output value is VS_CHAR*
#define VSTYPE_RECT
#define VSTYPE_FONT
#define VSTYPE_TIME
#define VSTYPE_OBJPTR
#define VSTYPE_PARAPKGPTR   // output value is VS_PARAPKGPTR
#define VSTYPE_BINBUFPTR      // output value is VS_BINBUFPTR

you can also push values onto lua stack, and call LuaSetObjectValue.
call LuaGetObjectValue to get object's value

//-----------------------------------------------------
The following function should be used for raw object,
```

```
   VS_BOOL SRPAPI ScriptSetBoolIndex(void *Object,VS_INT32 Index,VS_BOOL Value);
   VS_BOOL SRPAPI ScriptSetIntIndex(void *Object,VS_INT32 Index,VS_INT32 Value);
   VS_BOOL SRPAPI ScriptSetNumberIndex(void *Object,VS_INT32 Index,VS_FLOAT Value);
   VS_BOOL SRPAPI ScriptSetStrIndex(void *Object,VS_INT32 Index,VS_CHAR *Value);
   VS_BOOL SRPAPI ScriptSetObjectIndex(void *Object,VS_INT32 Index,VS_UINT8 Type,VS_ULONG
Value);

   VS_BOOL SRPAPI ScriptGetBoolIndex(void *Object,VS_INT32 Index);
   VS_INT32 SRPAPI ScriptGetIntIndex(void *Object,VS_INT32 Index);
   VS_FLOAT SRPAPI ScriptGetNumberIndex(void *Object,VS_INT32 Index);
   VS_CHAR *SRPAPI ScriptGetStrIndex(void *Object,VS_INT32 Index);
   VS_ULONG SRPAPI ScriptGetObjectIndex(void *Object,VS_INT32 Index,VS_UINT8 *RetType);
//-----------------------------------------------------
The following function should be used for raw object,
   VS_ULONG SRPAPI ScriptGetRawObject(void *Object,VS_CHAR *AttributeName,VS_UINT8
*RetType);
   VS_ULONG SRPAPI ScriptGetRawObjectIndex(void *Object,VS_INT32 Index,VS_UINT8 *RetType);

//-----------------------------------------------------
   void SRPAPI SetReturnRawFlag(void *Object,VS_BOOL ReturnRawFlag);
   VS_BOOL SRPAPI GetReturnRawFlag(void *Object);
```

ReturnRawFlag is valid for raw object which wrap lua and python object. In normal case, lua table and python tuple is tried to convert to parapkg. But is ReturnRawFlag == true, then, lua table and python tuple will be wrapped with starcore object.

5.104 Authorize

```
   VS_BOOL SRPAPI IsRegistered();
   whether service is registered. Do not call the function frequently.

   void SRPAPI CheckPassword(VS_BOOL CheckFlag);
     If check password is set to false, then when _GetService of ServiceGroupObject is called, the cle will not
check user password.
```

5.105 Get/Set Object Source Script Interface

```
   void SRPAPI SetSourceScript(void *Object,VS_INT32 SourceScript);
   VS_INT32 SRPAPI GetSourceScript(void *Object);
```

5.106 GetObject CallBack

```
   void SRPAPI RegGetObjectCallBack(VS_GetObjectCallBackProc CallBackProc,VS_ULONG Para);
   void SRPAPI UnRegGetObjectCallBack(VS_GetObjectCallBackProc CallBackProc,VS_ULONG Para);
   void SRPAPI RegGetObjectExCallBack(VS_GetObjectExCallBackProc CallBackProc,VS_ULONG Para);
   void SRPAPI UnRegGetObjectExCallBack(VS_GetObjectExCallBackProc CallBackProc,VS_ULONG
Para);
```

5.107 Object's call super or base

```
   VS_BOOL SRPAPI SetCallSuper(void *Object);
   VS_BOOL SRPAPI SetCallBase(void *Object,void *Class);
```

```
VS_BOOL SRPAPI PushCallBase(void *Object,void *Class);
VS_BOOL SRPAPI PopCallBase(void *Object,void *Class);
```

5.108 lua functions

```
void SRPAPI LuaReplace(VS_INT32 index);
void SRPAPI LuaCheckStack(VS_INT32 sz);
VS_BOOL SRPAPI LuaIsUserData(VS_INT32 index);
VS_BOOL SRPAPI LuaIsLightUserData (VS_INT32 index);
VS_INT32 SRPAPI LuaRawEqual (VS_INT32 index1, VS_INT32 index2);
VS_INT32 SRPAPI LuaCompare (VS_INT32 index1, VS_INT32 index2, VS_INT32 op);
void SRPAPI LuaConcat (VS_INT32 n);
void SRPAPI LuaCopy (VS_INT32 fromidx, VS_INT32 toidx);
void *SRPAPI LuaToPointer (VS_INT32 index);
void *SRPAPI LuaToCFunction (VS_INT32 index);
void SRPAPI LuaPushLightUserData (void *p);
void SRPAPI LuaRawGet (VS_INT32 index);
void SRPAPI LuaRawGeti (VS_INT32 index, VS_INT32 n);
void SRPAPI LuaCreateTable (VS_INT32 narr, VS_INT32 nrec);
VS_INT32 SRPAPI LuaGetMetatable (VS_INT32 index);
void SRPAPI LuaSetField (VS_INT32 index, VS_CHAR *k);
void SRPAPI LuaRawSet (VS_INT32 index);
void SRPAPI LuaRawSeti (VS_INT32 index, VS_INT32 n);
void SRPAPI LuaSetMetatable (VS_INT32 index);
VS_INT32 SRPAPI Lua_PCall (VS_INT32 nargs, VS_INT32 nresults, VS_INT32 msgh);
void SRPAPI Lua_Call (VS_INT32 nargs, VS_INT32 nresults);
void *SRPAPI LuaAtPanic (void *panicf);
VS_INT32 SRPAPI LuaGC (VS_INT32 what, VS_INT32 data);
VS_INT32 SRPAPI LuaError ();
   VS_INT32 SRPAPI LuaIsNoneOrNil (VS_INT32 index);
   VS_CHAR *SRPAPI LuaTypeName(VS_INT32 tp);

   //--luaL functions
   void SRPAPI LuaL_CheckAny (VS_INT32 narg);
   VS_INT32 SRPAPI LuaL_CheckInt (VS_INT32 narg);
   VS_LONG SRPAPI LuaL_CheckLong (VS_INT32 narg);
   const VS_CHAR *SRPAPI LuaL_CheckLString (VS_INT32 narg, VS_INT32 *l);
   VS_DOUBLE SRPAPI LuaL_CheckNumber (VS_INT32 narg);
   VS_INT32 SRPAPI LuaL_CheckOption (VS_INT32 narg,const VS_CHAR *def,const VS_CHAR *const
lst[]);
   void SRPAPI LuaL_CheckStack (VS_INT32 sz, const VS_CHAR *msg);
   const VS_CHAR *SRPAPI LuaL_CheckString (VS_INT32 narg);
   void SRPAPI LuaL_CheckType (VS_INT32 narg, VS_INT32 t);
   void *SRPAPI LuaL_CheckUData (VS_INT32 narg, const VS_CHAR *tname);
   void SRPAPI LuaL_CheckVersion ();
   VS_INT32 SRPAPI LuaL_DoFile (const VS_CHAR *filename);
   VS_INT32 SRPAPI LuaL_DoString (const VS_CHAR *str);
   VS_INT32 SRPAPI LuaL_Error (const VS_CHAR *info);
   void SRPAPI LuaL_GetMetatable (const VS_CHAR *tname);
   VS_INT32 SRPAPI LuaL_GetSubTable (VS_INT32 idx, const VS_CHAR *fname);
   VS_INT32 SRPAPI LuaL_Len (VS_INT32 index);
   VS_INT32 SRPAPI LuaL_LoadBuffer (const VS_CHAR *buff,VS_INT32 sz,const VS_CHAR *name);
   VS_INT32 SRPAPI LuaL_LoadBufferx (const VS_CHAR *buff,VS_INT32 sz,const VS_CHAR
*name,const VS_CHAR *mode);
   VS_INT32 SRPAPI LuaL_LoadFile (const VS_CHAR *filename);
   VS_INT32 SRPAPI LuaL_LoadFilex (const VS_CHAR *filename,const VS_CHAR *mode);
   VS_INT32 SRPAPI LuaL_LoadString (const VS_CHAR *s);
```

```
void SRPAPI LuaL_NewLib (const VSLuaL_Reg *l);
VS_INT32 SRPAPI LuaL_NewMetatable (const VS_CHAR *tname);
VS_INT32 SRPAPI LuaL_Ref (VS_INT32 t);
void SRPAPI LuaL_Requiref (const VS_CHAR *modname,void *openf, VS_INT32 glb);
void SRPAPI LuaL_SetFuncs (const VSLuaL_Reg *l, VS_INT32 nup);
void SRPAPI LuaL_SetMetatable (const VS_CHAR *tname);
void *SRPAPI LuaL_TestUData (VS_INT32 narg, const VS_CHAR *tname);
const VS_CHAR *SRPAPI LuaL_ToLString (VS_INT32 idx, VS_INT32 *len);
const VS_CHAR *SRPAPI LuaL_TypeName (VS_INT32 index);
void SRPAPI LuaL_UnRef (VS_INT32 t, VS_INT32 ref);
void SRPAPI LuaL_Where (VS_INT32 lvl);
```

5.109 Get Control Service

class ClassOfSRPInterface *GetControlService();
may be return NULL

5.110 new function call back

VS_BOOL SRPAPI RegNewFunctionCallBack(void *Object, VS_NewFunctionCallBackProc callback, VS_ULONG Para);
VS_BOOL SRPAPI UnRegNewFunctionCallBack(void *Object, VS_NewFunctionCallBackProc callback, VS_ULONG Para);
/* 1. when RegLuaFunc is called
2. when __newindex is called
3. the function is not system event
4. when CreateOVLFunction is called
5. when CreateAtomicScript/CreateAtomicFunction/CreateAtomicFunctionEx/CreateAtomicLuaFunction/CreateAtomicOvlFunction/CreateAtomicFunctionSimple is called
*/

5.111 Reference Count

void SRPAPI AddRef();
Increase reference count;

VS_INT32 SRPAPI GetRef();
Get current reference count

void SRPAPI ReleaseOwner();
Release owner, this function will decrease reference count, but not free the instance.

5.112 Malloc Object with parameter

void *SRPAPI **IMallocStaticObject**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf);
void *SRPAPI **IMallocStaticObjectEx**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf)
void *SRPAPI **IMallocGlobalObject**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf,VS_ULONG ClientID);

void *SRPAPI **IMallocGlobalObjectEx**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf,VS_ULONG ClientID);
void *SRPAPI **IMallocObject**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf);
void *SRPAPI **IMallocObjectEx**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf);
void *SRPAPI **IMallocObjectL**(VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf);
void *SRPAPI **IMallocObjectLEx**(VS_UUID *ObjectID,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf);
void *SRPAPI **IMallocClientObject**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf,VS_ULONG ClientID);
void *SRPAPI **IMallocClientObjectEx**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_PARAPKGPTR InitBuf,VS_ULONG ClientID);

VS_PARAPKGPTR SRPAPI **GetInitPara**(void *Object); // valid in ONCREATE event

5.113 Malloc Object with typeset and parameter

void *SRPAPI **IMallocStaticObject2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocStaticObjectEx2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocGlobalObject2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocGlobalObjectEx2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocObject2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocObjectEx2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocObjectL2**(VS_UUID *ObjectClassID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocObjectLEx2**(VS_UUID *ObjectID,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocClientObject2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,...);
void *SRPAPI **IMallocClientObjectEx2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,...);

void *SRPAPI **IMallocStaticObjectVar2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocStaticObjectExVar2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocGlobalObjectVar2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocGlobalObjectExVar2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocObjectVar2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,va_list argList);

void *SRPAPI **IMallocObjectExVar2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocObjectLVar2**(VS_UUID *ObjectClassID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocObjectLExVar2**(VS_UUID *ObjectID,VS_UUID *ObjectClassID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocClientObjectVar2**(void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,va_list argList);
void *SRPAPI **IMallocClientObjectExVar2**(VS_UUID *ObjectID,void *ParentObject,OBJECTATTRIBUTEINDEX AttributeIndex,VS_UUID *ObjectClassID,VS_ULONG ClientID,VS_CHAR *TypeSet,va_list argList);

For these functions, TypeSet supports '(' and ')' symbols, which means sub-parapkg.
for example,
"i(ii)i" corresponding to integer, parapkg, integer.

5.114 Script Raw Function

VS_BOOL SRPAPI **LoadRawModule**(VS_CHAR *ScriptInterface,VS_CHAR *ModuleName,VS_CHAR *FileOrString,VS_BOOL IsString,VS_CHAR **ErrorInfo)

Load Raw Module, which may be lua module, python module, java class library, csharp class library. for version 1.90.1, the above four scripts are supported.
ModuleName : if has been loaded, the the function does nothing. Module name may be input NULL for global name space.
RunString : may be file or string buf, which will be executed.
IsString : true for string, false for file.
ErrorInfo : may be null;

for lua :
if RunString == true,then load and execute lua code.
if RunString == false,then load and execute lua file.

for python :

if IsString == false, and FileOrString is valid, then load python file as module.
if IsString == true/false, and FileOrString is invalid, using python import function to load module
if IsString == true, and FileOrString is valid, then load python code as module.

for java :
IsString is ignored.
if FileOrString is valid, then take FileOrString as class library file. Or else, take ModuleName as class library file.

for csharp :
IsString is ignored.
if ModuleName is valid and FileOrString is invalid,then load assembly by name.
if FileOrString is valid, then load Assembly by file.

VS_BOOL SRPAPI **LoadRawModuleEx**(VS_CHAR *ScriptInterface,VS_CHAR *ModuleName,void *Object,VS_CHAR **ErrorInfo)
This function is valid for csharp, object should be raw assembly object.

VS_BOOL SRPAPI **AttachRawContext**(void *Object,VS_CHAR *ScriptInterface, VS_CHAR *ContextName, VS_BOOL IsClass, VS_CHAR *ContextInfo)

Associate an object with a script class or object.

for c/c++ :
ContextName is share library name, IsClass and ContextInfo is ignored.

for lua :
ContextName is table or userdata.
if ContextName == "{}", then create a new lua table.
if IsClass == true, then the lua value can be called to create instance.
ContextInfo is ignored.

for python :
ContextName is module, dict, tuple, list, class.
If ContextName is invalid, then the object attach python global name space.
if IsClass == true, then the python value should be a class.
ContextInfo is ignored.

for java :
ContextName must be valid.
ContextName is class name.
ContextInfo is ignored.

for csharp :
ContextName must be valid.
ContextName is class name.
ContextInfo is ignored.

void SRPAPI **DetachRawContext**(void *Object,VS_BOOL CallUnLockGC)

if CallUnLockGC == true, then UnLockGC will be called autotically.

VS_CHAR *SRPAPI **GetRawContextType**(void *Object,VS_CHAR **ScriptInterface);
The function is provided for script interface.

VS_BOOL SRPAPI **CreateRawContextBuf**(void *Object,VS_CHAR *ScriptInterface,VS_INT8 *ContextBuf,VS_INT32 ContextBufSize);
The function is provided for script interface.

void *SRPAPI **GetRawContextBuf**(void *Object,VS_CHAR *ScriptInterface);
The function is provided for script interface.

VS_BOOL SRPAPI **HasRawContext**(void *Object);
VS_BOOL SRPAPI **RawContextEquals**(void *Object1,void *Object2);

void *SRPAPI **AssignRawObject**(void *Object,void *RawObject)
copy raw object info from RawObject, and return self

void ***NewRawProxy**(VS_CHAR *ScriptInterface,void *AttachObject,VS_CHAR *AttachFunction,VS_CHAR *ProyInfo,VS_INT32 ProxyType);
AttachObject: is cle object, which contains functions callback from the proxy.
AttachFunction: is callback function name
ProyInfo : proxy info, varies with script interface
ProxyType : 1, the callback function run in main thread, which inits CLE. Others are reserved, set to 0;

for c/c++ :
The function is invalid.

for lua :
The function returns a callable lua userdata. which can be called from lua script.
ProyInfo: is ignored

for python :
The function returns a callable python object. which can be called from python script.
ProyInfo: is ignored

for java :
The function returns a proxy of java interface.
ProyInfo: java interfaces, separated with ‘,’
AttachFunction is ignored

for csharp :
The function returns a proxy of c# delegate.
ProyInfo: c# delegate name
note : for wp8, only delegates : void XXX(object arg) and void XXX(object arg1,object arg2) are supported

void *SRPAPI **NewRawProxyEx**(void *Object, VS_CHAR *ScriptInterface,VS_CHAR *AttachFunction,VS_CHAR *ProyInfo)
if ScriptInterface == NULL, the input Object should be raw object.

VS_CHAR *SRPAPI **CreateRawProxyCode**(VS_CHAR *ScriptInterface,VS_CHAR *NewPackageName,VS_CHAR *Imports,VS_CHAR *NewClassName,VS_CHAR *BaseClass, VS_CHAR *Methods, VS_CHAR *Interface);
This function is used to create code of extend class, valid for java and c#.
for java :
NewPackageName : is package name
Imports : is libraries to be import, separated by “,” , may be NULL
NewClassName : new class name to be created
BaseClass : name of base class, , may be NULL
Methods : public methods to be override, may be NULL
Interface : name of interfaces, separated by “,”, may be NULL

for example:
code =
Service:_CreateRawProxyCode("java","","testextend.*","ExtendBaseClass","testextend/BaseClass","getstr,getmyclass,getstres,getmyclasses,getmyobjectes,getmyobject","testextend/ICallBack");

for csharp :
NewPackageName : is namespace
Imports : is libraries to be import, separated by “,” , may be NULL
NewClassName : new class name to be created
BaseClass : name of base class, , may be NULL
Methods : public methods to be override, may be NULL
Interface : name of interfaces, separated by “,”, may be NULL

for example:
code =
Service:_CreateRawProxyCode("csharp","","testextend","ExtendBaseClass","testextend.BaseClass","getstr,getmyclass,getstres,getmyclasses,getmyobjectes,getmyobject","testextend.ICallBack");

for python :
NewPackageName : is ignored
Imports : is libraries to be import, separated by “,” , may be NULL
NewClassName : new class name to be created

BaseClass : name of base class,
Methods : methods to be override,should use fullname with parameter and seperatedby ';'
Interface : is ignored

for example:
code =
service._CreateRawProxyCode("python","","","ExtendBaseClass","BaseClass","__init__(self);getstr(self,val)",
"");

void ***ImportRawContext**(VS_CHAR *ScriptInterface,VS_CHAR *ContextName, VS_BOOL IsClass, VS_CHAR *ContextInfo);
This function first malloc new object,and then call AttachRawContext for the new object

void* SRPAPI **NewScriptRawType**(VS_INT32 RawType, VS_BOOL *IsParaPkg);
VS_BOOL SRPAPI **SetScriptRawType**(void *Object,VS_INT32 RawType);
VS_INT32 SRPAPI **GetScriptRawType**(void *Object);

VS_BOOL SRPAPI **RegRawLuaSetValueFunc**(void *Object, VS_LuaSetValueProc SetValueProc, VS_ULONG Para);
void *SRPAPI **GetRawLuaSetValueFunc**(void *Object, VS_CHAR *ValueName,VS_ULONG *Para);
if return is not null, then the Value can be set to raw object
VS_BOOL SRPAPI **UnRegRawLuaSetValueFunc**(void *Object, VS_LuaSetValueProc SetValueProc, VS_ULONG Para);

5.115 Sync Call Function

The following functions run in main thread, which inits CLE.
VS_BOOL SRPAPI **LuaSyncCall**(void *Object, VS_CHAR *ScriptName, VS_INT32 nArgs,VS_INT32 nOutArgs);
VS_UWORD SRPAPI **ScriptSyncCall**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,...);
VS_UWORD SRPAPI **ScriptSyncCallVar**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,va_list argList);
VS_FLOAT SRPAPI **ScriptSyncFCall**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,...);
VS_FLOAT SRPAPI **ScriptSyncFCallVar**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,va_list argList);
VS_UWORD SRPAPI **ScriptSyncCall2**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,VS_UINT8 *RetType,...);
VS_UWORD SRPAPI **ScriptSyncCallVar2**(void *Object,VS_ULONG *RetCode,VS_CHAR *FunctionName,VS_CHAR *TypeSet,VS_UINT8 *RetType,va_list argList);

5.116 Debug of index or memory leak

As follows:

```
#ifndef _VSOPENRESDBG
  #define CreateIndex(X)    CreateIndex_Nor(X)
  #define CreateIndexCmp(X,Y)    CreateIndexCmp_Nor(X,Y)
  #define CreateIDIndex()   CreateIDIndex_Nor()
  #define CreateIDIndexEx() CreateIDIndexEx_Nor()
  #define CreateMemory(X)   CreateMemory_Nor(X)
  #define GetMemoryPtr(X)   GetMemoryPtr_Nor(X)
  #define Malloc(X)         Malloc_Nor(X)
  #define ProcessEvent      ProcessEvent_Nor
```

```
   #define PostProcessEvent  PostProcessEvent_Nor
#else
   #define CreateIndex(X)    CreateIndex_Dbg(X,__FILE__,__LINE__)
   #define CreateIndexCmp(X,Y)    CreateIndexCmp_Dbg(X,Y,__FILE__,__LINE__)
   #define CreateIDIndex()   CreateIDIndex_Dbg(__FILE__,__LINE__)
   #define CreateIDIndexEx() CreateIDIndexEx_Dbg(__FILE__,__LINE__)
   #define CreateMemory(X)   CreateMemory_Dbg(X,__FILE__,__LINE__)
   #define GetMemoryPtr(X)   GetMemoryPtr_Dbg(X,__FILE__,__LINE__)
   #define Malloc(X)         Malloc_Dbg(X,__FILE__,__LINE__)
   #define ProcessEvent      ProcessEvent_Dbg
   #define PostProcessEvent  PostProcessEvent_Dbg
#endif
```

5.117 Function call macro

To simplify function call, CLE provides the following macros:

```
#define SRPCALL_RET1( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj)
#define SRPCALL_RET2( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj,SRP_Arg2)
#define SRPCALL_RET3( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj,SRP_Arg2,SRP_Arg3)
…..
#define SRPCALL_RET20( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj,SRP_Arg2,SRP_Arg3,…)

#define SRPCALL1( SRP_Func, SRP_Obj)
#define SRPCALL2( SRP_Func, SRP_Obj1,SRP_Arg2)
…
#define SRPCALL20( SRP_Func, SRP_Obj,SRP_Arg2,…)

#define SRPCALL_RETEX1( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj)
#define SRPCALL_RETEX2( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj,SRP_Arg2)
#define SRPCALL_RETEX3( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj,SRP_Arg2,SRP_Arg3)
…..
#define SRPCALL_RETEX20( SRP_Func,SRP_Ret,SRP_Default, SRP_Obj,SRP_Arg2,SRP_Arg3,…)

#define SRPCALLEX1( SRP_Func, SRP_Obj)
#define SRPCALLEX2( SRP_Func, SRP_Obj,SRP_Arg2)
…
#define SRPCALLEX20( SRP_Func, SRP_Obj,SRP_Arg2,…)
```

5.118 Attribte change macro

```
#define SRPCHANGE_BOOL( SRP_Interface, Object, SRP_VarIndex, SRP_Value )
#define SRPCHANGE_SHORT( SRP_Interface, Object, SRP_VarIndex, SRP_Value )
#define SRPCHANGE_INT( SRP_Interface, Object, SRP_VarIndex, SRP_Value )
#define SRPCHANGE_FLOAT( SRP_Interface, Object, SRP_VarIndex, SRP_Value ) \

//---global attribute change
#define SRPCHANGEEX_BOOL( SRP_Interface, Object, SRP_VarIndex, SRP_Value )
#define SRPCHANGEEX_SHORT( SRP_Interface, Object, SRP_VarIndex, SRP_Value )
#define SRPCHANGEEX_INT( SRP_Interface, Object, SRP_VarIndex, SRP_Value )
#define SRPCHANGEEX_FLOAT( SRP_Interface, Object, SRP_VarIndex, SRP_Value ) \
```

6 SRPParaPackageInterface

Suppport:object、integer、double、binbuf、time, and string.
Class define: **class ClassOfSRPParaPackageInterface**
Index starts from 0.

6.1 Basic function

6.1.1 Get parameter number -GetNumber

VS_INT32 **GetNumber**();

6.1.2 Clear -Clear

void **Clear** ();

6.1.3 Build -Build

class ClassOfSRPParaPackageInterface *SRPAPI Build(VS_CHAR *TypeSet,...)
for explain of TypeSet, please refer to ScriptCall
For Build and BuildVar function, TypeSet supports '(' and ')' symbols, which means sub-parapkg.
for example,
"i(ii)i" corresponding to integer, parapkg, integer.

6.1.4 BuildVar -BuildVar

class ClassOfSRPParaPackageInterface *SRPAPI BuildVar(VS_CHAR *TypeSet,va_list argList)

6.2 Inert parameter

6.2.1 Inert Bool -InsertBool

VS_BOOL **InsertBool**(VS_INT32 Index,VS_BOOL Para);

6.2.2 Inert integer -InsertInt

VS_BOOL **InsertInt**(VS_INT32 Index,VS_INT32 Para);

6.2.3 Inert float-InsertFloat

VS_BOOL **InsertFloat**(VS_INT32 Index, VS_DOUBLE Para);

6.2.4 Inert string-InsertStr

VS_BOOL **InsertStr**(VS_INT32 Index, VS_CHAR *Para);

6.2.5 Inert binbuf-InsertBin

VS_BOOL **InsertBin**(VS_INT32 Index,VS_INT8 *Para,VS_INT32 Size);

6.2.6 Inert time-InsertTime

VS_BOOL **InsertTime**(VS_INT32 Index, VS_TIME *hTime);

6.2.7 Inert empty-InsertEmpty

VS_BOOL **InsertEmpty** (VS_INT32 Index);

6.2.8 Insert Object —InsertObject

VS_BOOL SRPAPI InsertObject(VS_INT32 Index,void *Object);
Only valid locally

6.2.9 Insert ParaPkg —InsertParaPackage

VS_BOOL SRPAPI InsertParaPackage(VS_INT32 Index,class ClassOfSRPParaPackageInterface *ParaPkg);

6.3 Get parameter

6.3.1 Get parameter type -GetType

VS_INT32 **GetType**(VS_INT32 Index);
Type of parameter defined as follows:

```
#define SRPPARATYPE_INVALID   0    //--invalid
#define SRPPARATYPE_INT       1    //--integer
#define SRPPARATYPE_FLOAT     2    //--double
#define SRPPARATYPE_BIN       3
#define SRPPARATYPE_CHARPTR   4    //--string
#define SRPPARATYPE_TIME  5    //--time
#define SRPPARATYPE_BOOL      6    //--bool
#define SRPPARATYPE_OBJECT    7
#define SRPPARATYPE_PARAPKG   8
#define SRPPARATYPE_INT64     9
```

6.3.2 Get bool -GetBool

VS_BOOL **GetInt**(VS_INT32 Index);

6.3.3 Get integer -GetInt

VS_INT32 **GetInt**(VS_INT32 Index);

6.3.4 Get float-GetFloat

VS_DOUBLE **GetFloat**(VS_INT32 Index);

6.3.5 Get string-GetStr

VS_CHAR ***GetStr**(VS_INT32 Index);

6.3.6 Get binbuf-GetBin

VS_INT8 ***GetBin**(VS_INT32 Index,VS_INT32 *Length); Length may be set to NULL.

6.3.7 Get time -GetTime

VS_BOOL SRPAPI **GetTime**(VS_INT32 Index, VS_TIME *hTime);
Index starts from 0;

6.3.8 Get Object —GetObject

void *SRPAPI GetObject(VS_INT32 Index)
Only valid locally

6.3.9 Get ParaPackage —GetParaPackage

class ClassOfSRPParaPackageInterface *SRPAPI GetParaPackage(VS_INT32 Index);

6.4 Parameter exchange or delete

6.4.1 Parameter exchange -ExChange

VS_BOOL SRPAPI **ExChange**(VS_INT32 DesIndex,VS_INT32 SrcIndex);

6.4.2 Parameter delete -Del

void SRPAPI **Del**(VS_INT32 Index);

6.5 Parapkg delete

6.5.1 Parapkg duplicate -Dup

class ClassOfSRPParaPackageInterface *SRPAPI **Dup**(); Create a new parapkg

6.5.2 Parapkg append -AppendFrom

VS_BOOL **AppendFrom**(class ClassOfSRPParaPackageInterface *ParaPkg);

6.6 Pack interface

May set the changed fileds, and only pack the changed fields

6.6.1 Set fields change flag-SetChangeFlag

void SRPAPI SetChangeFlag(VS_INT32 Index);

6.6.2 Set all fields change flag-SetChangeFlagEx

void SRPAPI ClearChangeFlag(VS_INT32 Index);

6.6.3 Clear fields change flag - ClearChangeFlag

void SRPAPI ClearChangeFlag(VS_INT32 Index);

6.6.4 Clear all fields change flag -ClearChangeFlagEx

void SRPAPI ClearChangeFlagEx();

6.6.5 Whether fields is changed -IsChangeFlag

VS_BOOL SRPAPI IsChangeFlag(VS_INT32 Index);

6.6.6 Whether fields is changed -IsChangeFlagEx

VS_BOOL SRPAPI IsChangeFlagEx();

6.6.7 Pack changed fields to buffer-SaveChangeToBuf

VS_INT8 *SRPAPI SaveChangeToBuf(VS_BOOL CompressFlag,VS_INT32 *RetSize)
Returns buffer pointer of the data, and its size.
The buffer should be freed by interface function FreeBuf.

6.6.8 Pack all fields to buffer -SaveChangeToBufEx

VS_INT8 *SRPAPI SaveChangeToBufEx(VS_BOOL CompressFlag,VS_INT32 *RetSize)
Returns buffer pointer of the data, and its size.
The buffer should be freed by interface function FreeBuf.

6.6.9 Restore fields from buffer-LoadChangeToBuf

VS_BOOL SRPAPI LoadChangeToBuf(VS_INT32 BufSize, VS_INT8 *Buf);

6.7 Free Buf

void SRPAPI FreeBuf(VS_INT8 *Buf);

6.8 Reference Count

void SRPAPI AddRef();
Increase reference count;

VS_INT32 SRPAPI GetRef();
Get current reference count

void SRPAPI ReleaseOwner();
Release owner, this function will decrease reference count, but not free the instance.

6.9 Raw Type

void SRPAPI **SetScriptRawType**(VS_INT32 RawType)
VS_INT32 SRPAPI **GetScriptRawType**();

6.10 Dict and JSON functions

```
class ClassOfSRPParaPackageInterface *SRPAPI AsDict(VS_BOOL IsDict);
/* IsDict == true, the parapkg is act as dict, or else is normal dict */

VS_BOOL SRPAPI IsDict();
/* the parapkg is act as dict, or not */

VS_INT32 SRPAPI FindDict(VS_CHAR *Key);
/*return is the value index, -1 means not find*/

VS_INT32 SRPAPI FindDictEx(VS_INT32 Key);
/*return is the value index, -1 means not find*/

/*---cjson, the even index is key, the odd index is value*/
VS_BOOL SRPAPI FromJSon(VS_CHAR *Buf);
/* if the return value is not null, FreeBuf should be called to free the memory*/

VS_CHAR *SRPAPI ToJSon();
/*the even index is key, must be string; the odd index is value, may be
SRPPARATYPE_INT,SRPPARATYPE_FLOAT,SRPPARATYPE_CHARPTR,SRPPARATYPE_BOOL,SRP
PARATYPE_PARAPKG */
```

6.11 Int64 functions

```
VS_BOOL SRPAPI InsertInt64(VS_INT32 Index,VS_INT64 Para);
VS_INT64 SRPAPI GetInt64(VS_INT32 Index);    /*---From 0*/
```

7 ClassOfSRPLockInterface

Class define: **class ClassOfSRPLockInterface**

7.1 Basic function

7.1.1 Free

void SRPAPI Release();

7.1.2 Lock

void SRPAPI Lock();

7.1.3 UnLock

void SRPAPI UnLock();

8 ClassOfSRPBinBufInterface

Class define: **class ClassOfSRPBinBufInterface**

8.1 Basic function

8.1.1 Free

void SRPAPI Release();

8.1.2 Init -Init

void SRPAPI Init(VS_UINT32 BufSize), If BufSize is not 0, the alloc buf according the size.

8.1.3 Get buffer size -GetSize

VS_UINT32 SRPAPI GetSize()

8.1.4 Get valid data offset -GetOffset

VS_UINT32 SRPAPI GetOffset()

8.1.5 Get buffer address -GetBuf

VS_INT8 *SRPAPI GetBuf()

8.1.6 Clear buffer content-Clear

void SRPAPI Clear()

8.1.7 Cleat content from buffer-ClearEx

void SRPAPI ClearEx(VS_UINT32 Offset, VS_UINT32 Length)

8.1.8 Set data-Set

VS_BOOL SRPAPI Set(VS_UINT32 Offset,VS_UINT32 Length,VS_INT8 *Buf)

8.1.9 Get data-Get

VS_BOOL SRPAPI Get(VS_UINT32 Offset,VS_UINT32 Length,VS_INT8 *Buf)

8.1.10 Set buffer address-GetBufPtr

VS_INT8 *SRPAPI GetBufPtr (VS_UINT32 Offset)

8.1.11 Get buffer address-GetBufPtrEx

VS_INT8 *SRPAPI GetBufPtrEx(VS_UINT32 Offset,VS_UINT32 Length);
If Offset+Length is greater than valid data length, then the function returns NULL.

8.1.12 Set valid data size -SetOffset

VS_BOOL SRPAPI SetOffset(VS_UINT32 Offset)

If Offset is greater than current buffer size, then the function expands the buffer

8.1.13 Fill buffer-Fill

VS_BOOL SRPAPI Fill(VS_UINT32 Offset,VS_UINT32 Length,VS_UINT8 Value);

8.1.14 Expand space-Expand

VS_BOOL SRPAPI Expand(VS_INT32 NewBufSize);
If NewBufSize is less than current buffer size, then the function takes no effect.

8.1.15 Duplicate -Dup

class ClassOfSRPBinBufInterface *Dup();

8.1.16 Save and restore object

VS_BOOL SRPAPI PackObject(void *Object);
VS_BOOL SRPAPI UnPackObject(void *Object);

8.1.17 Local and UTF8 coding convert

VS_BOOL SRPAPI ToUTF8(void);
VS_BOOL SRPAPI ToAnsi(void);

The buffer should be a string. If the string is ended with 0, then the converted string is ended with 0, otherwise, without 0.

8.1.18 Local and UNICODE coding convert

VS_BOOL SRPAPI AnsiToUnicode(VS_CHAR *Code,VS_INT32 BytesPerChar);
VS_BOOL SRPAPI UnicodeToAnsi(VS_CHAR *Code,VS_INT32 BytesPerChar);
Code is ignored on windows, which may take value from:UCS2 UCS4 UTF-16 UTF-32 UTF-16BE UTF-16LE UTF-32BE UTF-32LE
BytesPerChar should be set to 2 on windows.

8.1.19 Print,Insert and string function

void SRPAPI Print(VS_UINT32 Offset,VS_CHAR *format,...); //insert MaxLength is 10240
void SRPAPI PrintVar(VS_UINT32 Offset,VS_CHAR *format,va_list argList); //insert MaxLength is 10240
void SRPAPI Insert(VS_UINT32 Offset,VS_UINT32 Length,VS_INT8 *Buf);
VS_INT32 SRPAPI FindStr(VS_UINT32 Offset,VS_CHAR *Str); //return the offset from input offset <0 represent not found
VS_INT32 SRPAPI FindStri(VS_UINT32 Offset,VS_CHAR *Str); //return the offset from input offset <0 represent not found

8.1.20 Light buffer

VS_BOOL SRPAPI SetLightBuf(VS_UINT32 Length,VS_INT8 *Buf);
VS_BOOL SRPAPI IsLightBuf();
Light buffer supports read, and not support operations about write.

8.1.21 Increate ref

```
void SRPAPI AddRef();
VS_ULONG SRPAPI GetRef();
```

8.1.22 Load from memory file

VS_ULONG SRPAPI WriteFromMemoryFile (class ClassOfSRPInterface *SRPInterface,VS_UINT32 Offset,VS_CHAR *FileName)

8.1.23 Compute MD5 or Hash value

VS_INT8 *SRPAPI GetMD5();
VS_ULONG SRPAPI GetHashValue();

8.1.24 Reference Count

void SRPAPI AddRef();
Increase reference count;

VS_INT32 SRPAPI GetRef();
Get current reference count

void SRPAPI ReleaseOwner();
Release owner, this function will decrease reference count, but not free the instance.

9 ClassOfSRPSXMLInterface

UTF-8 format.

9.1 Basic function

9.1.1 Load and save function

VS_BOOL SRPAPI LoadFromFile(VS_CHAR *FileName , VS_CHAR **ErrorInfo)
VS_BOOL SRPAPI LoadFromBuf(VS_INT8 *Buf , VS_CHAR **ErrorInfo);
VS_BOOL SRPAPI SaveToFile(VS_CHAR *FileName);
VS_BOOL SRPAPI SaveToBuf(class ClassOfSRPBinBufInterface *BinBuf);
VS_BOOL SRPAPI LoadFromAnsiBuf(VS_INT8 *Buf,VS_CHAR **ErrorInfo);
VS_BOOL SRPAPI SaveToAnsiBuf(class ClassOfSRPBinBufInterface *BinBuf);

9.1.2 Read function

```
VS_CHAR *SRPAPI GetStandalone();
VS_CHAR *SRPAPI GetVersion();
VS_CHAR *SRPAPI GetEncoding();

void *SRPAPI FindElement(VS_CHAR *Value);
void *SRPAPI FirstElement(void *ParentElement);
void *SRPAPI NextElement(void *Element);
void *SRPAPI ParentElement(void *Element)

VS_CHAR *SRPAPI GetElement(void *Element);
```

```
   void SRPAPI GetElementEx(void *Element,VS_CHAR *Buf,VS_INT32 BufSize); //returns format as
XXX.XX.XX
   //--xmlns:XXX=XXX
   VS_BOOL GetNs(void *Element,VS_CHAR *nsName,VS_INT32 nsNameBufSize,VS_CHAR **nsValue);
   VS_CHAR *SRPAPI GetNsValue(void *Element,VS_CHAR *nsName);
   void SRPAPI SetNs(void *Element,VS_CHAR *nsName,VS_CHAR *nsValue);

   void *SRPAPI FindAttribute(void *Element,VS_CHAR *Name);
   void *SRPAPI FirstAttribute(void *Element);
   void *SRPAPI NextAttribute(void *Attribute);
   VS_CHAR *SRPAPI GetAttributeName(void *Attribute);
   VS_CHAR *SRPAPI GetAttributeValue(void *Attribute);

   VS_CHAR *SRPAPI GetSingleText(void *Element);
   void *SRPAPI FirstText(void *Element);
   void *SRPAPI NextText(void *Text);
   VS_CHAR *SRPAPI GetText(void *Text);
```

9.1.3 Change function

```
   void SRPAPI SetDeclaration(VS_CHAR *Version,VS_CHAR *Encoding,VS_CHAR *Standalone);

   void *SRPAPI InsertElementBefore(void *ParentElement,void *Element,VS_CHAR *Value);
   If Element==NULL,then insert at first
   void *SRPAPI InsertElementAfter(void *ParentElement,void *Element,VS_CHAR *Value);
   If Element==NULL,then insert at last
   void SRPAPI RemoveElement(void *Element);
   void SRPAPI SetElement(void *Element,VS_CHAR *Value);
   If ParentElement is NULL, then insert to root element

   void *SRPAPI InsertTextBefore(void *ParentElement,void *Text,VS_CHAR *Value,VS_BOOL
CDataFlag);
   If Text ==NULL,then insert at first
   void *SRPAPI InsertTextBeforeEx(void *ParentElement,void *Text,VS_BOOL CDataFlag,VS_CHAR
*Format,...); Length should less then 1024
   void *SRPAPI InsertTextAfter(void *ParentElement,void *Text,VS_CHAR *Value,VS_BOOL CDataFlag);
   If Text ==NULL,then insert at last
   void *SRPAPI InsertTextAfterEx(void *ParentElement,void *Text,VS_BOOL CDataFlag,VS_CHAR
*Format,...); Length should less then 1024
   void SRPAPI RemoveText(void *Text);
   void SRPAPI SetText(void *Text,VS_CHAR *Value,VS_BOOL CDataFlag);
   void SRPAPI SetTextEx(void *Text,VS_BOOL CDataFlag,VS_CHAR *Format,...) Length should less then
1024
   If ParentElement is NULL, then insert to root element；

   void *SRPAPI InsertCommentBefore(void *ParentElement,void *Comment,VS_CHAR *Value);
   if Comment ==NULL,then insert at first
   void *SRPAPI InsertCommentBeforeEx(void *ParentElement,void *Comment,VS_CHAR *Format,...);
Length should less then 1024
   void *SRPAPI InsertCommentAfter(void *ParentElement,void *Comment,VS_CHAR *Value);
   if Comment ==NULL,then insert at last
   void *SRPAPI InsertCommentAfterEx(void *ParentElement,void *Comment,VS_CHAR *Format,...);
Length should less then 1024
   void SRPAPI RemoveComment(void *Comment);
   void SRPAPI SetComment(void *Comment,VS_CHAR *Value);
   void SRPAPI SetCommentEx(void *Comment,VS_CHAR *Format,...) Length should less then 1024
If ParentElement is NULL, then insert to root element；
```

void SRPAPI SetAttribute(void *Element,VS_CHAR *Name,VS_CHAR *Value);
void SRPAPI SetAttributeEx(void *Element,VS_CHAR *Name,VS_CHAR *Format,...) Length should less then 1024
void SRPAPI RemoveAttribute(void *Element,VS_CHAR *Name);

9.1.4 Duplicate function

class ClassOfSRPSXMLInterface *SRPAPI Dup();
VS_BOOL SRPAPI Copy(class ClassOfSRPSXMLInterface *SrcSXML)
void *SRPAPI CopyElementBefore(void *ParentElement,void *Element,void *SrcElement);
void *SRPAPI CopyElementAfter(void *ParentElement,void *Element,void *SrcElement);
VS_BOOL SRPAPI CopyChild(void *DesElement,void *SrcElement);

9.1.5 Reference Count

void SRPAPI AddRef();
Increase reference count;

VS_INT32 SRPAPI GetRef();
Get current reference count

void SRPAPI ReleaseOwner();
Release owner, this function will decrease reference count, but not free the instance.

10 ClassOfSRPFunctionParaInterface

Max para number is 64
Each parameter is 4 bytes (32bit),for parapkg and object, application should pass their pointer
Type supported:VS_BOOL,VS_INT8, VS_UINT8, VS_INT16,
VS_UINT16,VS_INT32,VS_UINT32,VS_LONG,VS_ULONG,
VS_FLOAT,VSTYPE_CHARPTR,VSTYPE_PARAPKGPTR,VSTYPE_BINBUFPTR,VS_OBJPTR
Type of return value: VS_BOOL,VS_INT8, VS_UINT8, VS_INT16,
VS_UINT16,VS_INT32,VS_UINT32,VS_LONG,VS_ULONG,
VS_FLOAT,VS_CHARPTR,VS_PARAPKGPTR,VS_OBJPTR

10.1 Basic function

void SRPAPI Clear();
VS_INT32 SRPAPI GetNumber();
VS_UINT8 SRPAPI GetType(VS_INT32 Index); //---Start from 0
VS_ULONG SRPAPI GetValue(VS_INT32 Index); //--- Start from 0
VS_BOOL SRPAPI SetValue(VS_INT32 Index,VS_UINT8 In_Type,VS_ULONG In_Para); //--- Start from 0
VS_BOOL SRPAPI Call(void *Object,VS_UUID *FunctionID,VS_ULONG *RetValue,VS_UINT8 *RetType);

class ClassOfSRPFunctionParaInterface *SRPAPI Dup()

10.2 Reference Count

void SRPAPI AddRef();
Increase reference count;

VS_INT32 SRPAPI GetRef();
Get current reference count

void SRPAPI ReleaseOwner();
Release owner, this function will decrease reference count, but not free the instance.

11 ClassOfSRPCommInterface

Function defined in this interface, may be run in seperated thread

12 Mapping between VS type and XML type

VSTYPE_BOOL	:	xsd:boolean
VSTYPE_INT8	:	xsd:byte
VSTYPE_UINT8	:	xsd:unsignedByte
VS_INT16	:	xsd:short
VSTYPE_UINT16	:	xsd:unsignedShort
VSTYPE_INT32	:	xsd:int
VSTYPE_UINT32	:	xsd:unsignedInt
VSTYPE_FLOAT	:	xsd:float
VSTYPE_LONG	:	xsd:long
VSTYPE_ULONG	:	xsd:unsignedLong
VSTYPE_LONGHEX	:	xsd:long
VSTYPE_ULONGHEX	:	xsd:unsignedLong
VSTYPE_VSTRING	:	xsd:string
VSTYPE_COLOR	:	xsd:unsignedLong
VSTYPE_RECT	:	xsd:string “left,top,right,bottom”
VSTYPE_FONT	:	xsd:string “height,size,charset,style,name”
VSTYPE_TIME	:	xsd:dateTime
VSTYPE_CHAR	:	xsd:string
VSTYPE_UUID	:	xsd:string
VSTYPE_STATICID	:	xsd:unsignedLong
VSTYPE_CHARPTR	:	xsd:string

13 Object encapsulation of C++

When creates object skeleton code, the C++ encapsulation of object will be generated:XXX_VSClass.cpp and XXX_VSClass.h. For each object, there is a common basic class ClassOfSRPObject, in which includes object attribute and function. Using the class, operations of SRP object is same as normal C++ object.

Public attribute and functions include in class ClassOfSRPObject:

Attributes:

1. void *ThisSRPObject; Wrapped SRP Object, can set using function WrapObject, and get directly.
2. class ClassOfSRPInterface *ThisSRPInterface; Operation interface, can set using function WrapObject, and get directly.
3. VS_BOOL AutoReleaseObject; If is True,then when the instance is deleted, the SRP object will be freed too. The value can read and write directly, default is false.
4. VS_BOOL AutoReleaseThis; If is True,then when SRP object is freed, the class instance will be delete too. The value can read and write directly, default is false.

Function:

1. virtual void Release();

2. void SRPAPI WrapObject(class ClassOfSRPInterface *In_SRPInterface,VS_BOOL In_AutoReleaseObject,VS_BOOL In_AutoReleaseThis,void *In_Object); Encapsulate SRP object.
3. void SRPAPI UnWrapObject(); Unwrap SRPObject, In this case, if AutoReleaseObject == true, the SRPObject will be freed.

For example, DirectoryClass is an encapsulation class.

```
class ClassOfDirectoryClass:public ClassOfSRPObject{
public:
   ClassOfDirectoryClass();  //--Not Create a Class Object, Use WrapObject() to attach
   ClassOfDirectoryClass(class ClassOfSRPInterface *In_SRPInterface);  //--Create a Class Object
   ClassOfDirectoryClass(class ClassOfSRPInterface *In_SRPInterface,void *SRPObject);  //--Create a Class
Object and wrap SRPObject
   virtual VS_CHAR *SRPAPI GetSelfName();
   static class ClassOfDirectoryClass *SRPAPI GetSRPWrap( class ClassOfSRPInterface
*In_SRPInterface,void *SRPObject,VS_ULONG In_ClassLayer = 0xFFFFFFFF);
public:
   //--Attribute Get/Put Function Define
   VS_VSTRING Get_Name();
   void Put_Name(VS_VSTRING In_Value);

   VS_TIME Get_FDate();
   void Put_FDate(VS_TIME In_Value);


#if( VS_OS_TYPE == VS_OS_WINDOW )
public:
   //--Attribute Property Define
   __declspec(property(get=Get_Name, put=Put_Name)) VS_VSTRING Name;
   __declspec(property(get=Get_FDate, put=Put_FDate)) VS_TIME FDate;
#endif

};
```

```
class ClassOfDirectoryClass *pWrapObject;

pWrapObject = new class ClassOfDirectoryClass()
pWrapObject -> WrapObject(SRPInterface, false, false, SRPObject )
pWrapObject -> Name = name string;
pWrapObject -> FDate = time.
```