蓝星际自动语音平台
外部动态连接库编写及调用规范
Ver 1.8

BlueSpace

www.bluespace.com.cn
zhudn@bluespace.com.cn

0 目 录

1 概述

欢迎使用蓝星际语音平台。本手册将帮助您开发出强大的 DLL 外部扩展。

在下列场合可能需要开发外部 DLL：

1. 当需要集成已经存在的业务系统，而且原来的业务系统通过传统的编程语言已经实现，如金融系统中的与后台系统的交易、查询，可简单地将原来的业务系统封装成 DLL，供脚本调用；
2. 当某些操作需要和底层接口打交道时，如操纵串口，调用平台没有封装的 win32API 等；
3. 当需要访问其它语言才有的机制时；
4. 当某些操作需要更高的安全性或更快的速度时，此外由于 DLL 仅仅是二进制模块，可以达到隐藏某些关键算法的目的。

但当某项功能可以用脚本语言实现时，建议不必用外部 DLL 来实现。原因请参考后面的讨论。

蓝星际语音平台支持外部动态连接库的调用，该技术使系统有很好的扩展性。但库的编写和调用必须遵守本规范。

下面的规范内容皆以 C/C++语言描述。如采用其它语言开发 DLL，方法是类似的，请一并参考该语言的手册。

2 编写规范

所有的 DLL 输出函数原型都必须为：

```
extern "C" DLLEXPORT int _stdcall FuncName(int paraNum,TagBsVar * paras);
```

即必须为 C 名字调用，stdcall 的参数顺序，int 返回值

paraNum 表示参数的个数，paras 是传递进来的参数数组，请参考下面的例子。

定义一个类型用来传递参数变量：

```
struct TagBsVar
{
    int type; // 变量的类型，可以为：TAGVAR_STR, TAGVAR_INT,
              TAGVAR_DOUBLE
    char * s; // 如果是 TAGVAR_STR 类型，变量的值等于 s 的内容
    int i;    // 如果是 TAGVAR_STR 类型，i 表示 s 的长度；如果是 VAR_INT
              类型，变量的值等于 i 的内容
    double d; // 如果是 TAGVAR_DOUBLE 类型，变量的值等于 d 的内容
};
```

注意：编译选项中的数据对齐（Data Alignment）方式要采用“Quad word”

变量的类型定义如下：

```
const int TAGVAR_STR = 1;    // 字符串
const int TAGVAR_INT = 2;    // 整型
const int TAGVAR_DOUBLE = 3; // 浮点型
```

注意： 如果是字符串型变量，调用者(脚本方)负责分配和释放缓冲区，动态库编写者只负责使用，但不可超过这个缓冲区的大小，该大小在 TagBsVar::i 中。所以在动态库端会改变字符串变量内容时，调用者(脚本方)必须分配足够大的内存，动态库编写者也必须在程序中检查传入参数的类型和大小，即 TagBsVar.type 和 TagBsVar.i。脚本语言提供了一个库函数 Repeatstr(v,subStr, n)能生产指定大小的字符串变量，如脚本语句 Repeatstr(v, " ", 1000)，则 v 的值为 1000 个空格组成的字符串。Repeatstr()函数的简单替代是字符串相乘，如：v = " " * 1000;这样更简单和直观。

此外，不能使用脚本语言中的数组类型变量作为参数，当需要返回大量列表数据给调用者时，可以将数据打包成一个格式化的字符串（如逗号分隔的串），脚本语言可以通过系统函数 AnlyStr()分解到数组。返回 XML 格式的字符串也很流行。

如果 DLL 能够存储静态数据，也可以给调用者提供一种检索数据的方法，例如：GetListItem(index, val); // 得到下标 index 的值放置到 val 变量。

需要注意的是，如果有多条线路加载同一个 DLL，它们将处于同一进程空间内，将共享 DLL 里面的全局变量，在编制 DLL 时需慎用全局变量。

下面举例说明：

动态连接库库名为"bs_test1.dll",里面必须有一个输出函数：

```
// Fun1 的实现函数，具体逻辑在这里实现，注意这个函数不必输出
// 这里可以调用你自己编写的类，甚至调用别的动态库等等
int Fun1Proc(int i1, int i2, char * str1)
{
   // procedue codes
   return(x);
}

extern "C" DLLEXPORT int _stdcall Fun1(int paraNum, TagBsVar * paras)
// 输出函数
{
   if( paraNum != 4 ){
      // 参数错误，Error Report
      return(-1);
   }

   // 取出传入的参数
   int a = paras[0].i;
   int b = paras[1].i;
   char * str = paras[2].s;
   // 实际实现可以增加判断参数有效性(合法性)的代码

   int ret = Fun1Proc(a, b, str); // 在这里调用 Fun1 的实现函数，并传入相应的参数

   // 设置返回参数
   paras[3].i = ret;
   paras[3].type = TAGVAR_INT;    // 指明返回值的类型

   return(ret);  // 这个值将会写入脚本语言(调用者)的系统变量_retVal之中
}
```

调用这个库的脚本例子：

```
hd = -1;
DllLoad("bs_test1.dll", hd);  // 装载这个库
DllFunction(hd, "Fun1", 4);   // 说明其中有一个函数,名字叫"Fun1", 共有四个参数
```

```
...
a = 1;
b = 2;
ret = Fun1(a, b, "test", ret);  // 调用 Fun1 这个函数，传入 4 个参数
if( ret<0 )
{
   // 处理返回结果...
}
...
DllFree(hd);  // 卸载动态库
```

实际上可以认为脚本用户真正调用的是函数 Fun1Proc，而 Fun1 仅仅是个入口.

技巧: 动态库的编写者除了提供动态库本身和说明文件外，可以同时提供函数说明的脚本文件，供用户包含。而且装载函数和 Dll 说明函数可以在函数内调用。

例如: 某个动态库实现了短信网关，共有 5 个函数，库文件：SMS.Dll，同时提供说明脚本文件 SMS.h，内容如下:

```
function LoadSmsDll()
{
  smsDllHd = -1;  // 初始化动态库句柄
  DllLoad("SMS.dll", smsDllHd);          // 装载这个库
  if( smsDllHd < 0 )
      return(-1);   // Error

   DllFunction(smsDllHd, "SMSOpen", 3);   // 打开设备
   DllFunction(smsDllHd, "SMSClose", 1);  // 关闭设备
   DllFunction(smsDllHd, "SMSSend", 2);   // 发送
   DllFunction(smsDllHd, "SMSRecv", 2);   // 接收
   DllFunction(smsDllHd, "SMSTest", 1);   // 测试
   return(0);
}
ret = LoadSmsDll();  // 装载
if( ret<0 )
   return(-1);
```

调用者的脚本可能为:

```
#include "SMS.h"  // 装载调用说明
smsHd = -1;  // 初始化短信句柄
SMSOpen("COM1", "19200", smsHd); // 打开
if( smsHd < 0 )
   return(-2);   // Error

while(true)  // 主循环
```

```
{
   SMSSend(smsHd, "Hello, Koodoo!");  // 发送
   num = 0;
   SMSTest(smsHd, num);  // 看看有没有短消息可收
   for(i=0; i<num; i++)
   {
      buf = " " * 512;  // 分配 512 个字节的空格。预先分配足够的缓冲区
      SMSRecv(smsHd,buf);  // 接收
      // 将接收到的 buf 存到数据库里面...
   }
   Sleep(0.6);
}
OnSysQuit();  // 如果系统退出
SMSClose(smsHd);  // 关闭
DbCloseAll();  // 关闭所有数据库
DllFreeAll();  // 释放所有的动态库
return(0);
```

这样调用者只需增加一条包含语句，其它就如同调用系统函数一样。

讨论:　从脚本设计的角度考虑，动态连接库调用扩展了用户功能，但也增加了用户滥用的机会，比如语音平台本来提供了时间日期的函数，有些用户也非要用动态连接库去实现不可；也增加了系统不稳定性的风险，因为 DLL 加载后和语音平台在同一个进程空间，如果用户编写的 DLL 的不可靠，将会直接导致运行平台的不稳定，当然蓝星际语音平台经过精心设计，当外部 DLL 出现异常时，语音平台将记录该异常到系统日志，并继续正常运行。此外动态连接库调用也增加了系统接口的复杂性。

蓝星际语音平台可以防范动态库出现的异常，从而保持运行的稳定性。
