传输代理客户端(TAC)
API 使用说明书
Version 1.0

1 引言

传输代理是一种简单的消息中间件，主要负责应用之间消息的可靠传送与管理。整个传输代理系统分为客户端（以下简称 TAC）和服务器端（TAS）。TAC 和 TAS 都是可配置的，每个 TAS 可以对应多个 TAC，但每个 TAC 必须是 TAS 的一个合法配置客户端。TAC 可以运行于各种 UNIX 系列操作系统和 WINDOWS 系列操作系统，通过 TCP 协议，与 TAS 进行数据交换，以提供标准 API 接口的形式和应用程序集成。

本文主要对 TAC 提供的所有 API 函数进行详细的说明，包括每个函数的功能概括、参数描述、返回值定义和注意事项等。并且以代码的形式举例说明了每个函数的调用过程，对 TAC 的配置文件也进行了详细的配置说明。

2 变量命名

本文档所描述的所有函数变量名称遵循以下命名规则：

Variable_type $1$2$3variable_name

$1	Pointer or not	P|	
$2	Data type	L	Long
		I	Int
		S	Short
		DW	Dword
		C	Char
		R	Record/structures
		D	Double/float double
		F	File
		H	Handle
$3	Scope	G	Golbal,visible anywhere in module.
		L	Local,visible inside current function only.
		P	Parameter,function parameter variable.

例子：

```
char pclName[10];  /*     pointer char local Name  */

char cgString;  /*     char global String    */
```

3 返回值常量定义

本文档函数说明中的返回值常量遵循以下定义规则。应用程序可以根据对应常量值自行定义。

#define TA_SUCCESS 1
#define TA_FAIL 0
#define TA_QUEUE_EMPTY –1
#define TA_QUEUE_FULL –2
#define TA_LINKOK 1
#define TA_LINKERROR 0

4 TAStart

int TAStart( )

函数功能：

读取客户端配置内容，根据配置进行初始化设置，负责主动与服务器端建立网络连接。应用程序在成功调用本函数后，便可进行正常的消息接收和发送工作。

返回值：

TA_SUCCESS	TAC 成功启动
TA_FAIL	启动 TAC 过程中发生错误（tac.log 文件记录错误的详细原因）

在以下几种条件下，**TAStart( )**返回 TA_FAIL：

1**TACStart( )**已经成功启动，并且没有被 Close(调用 **TAClose( )**函数)。

2 在应用程序当前目录下不存在 tac.ini 配置文件。

3 配置文件中的必要配置项目不存在。

4 配置文件中的某项配置内容不正确。

5 与服务器端建立网络连接失败。

6 客户端的 IP 非法。

7 同样的客户端实例已经存在。

备注：

TAStart( )函数必须在其他 **TA* (…)**函数前调用。而且只需要在应用程序和服务器进行通讯前调用一次，并不需要在每次发送和接收消息报文前调用。如希望重新调用 **TAStart( )**，必须先成功调用 **TAClose( )**函数。

关于必要配置项目以及如何正确设置可以查看“配置文件”。

每个 TAC（配置文件中的 Name 为本 TAC 的身份标志）在服务器端配置有合法 IP 集合，这里的非法 IP，是指客户端 IP 不在此集合中。出现此错误，TAC 强制性把整个应用程序关闭。可以通过修改配置文件中的 Name 或修改计算机 IP 解决这个问题。

同样名称的 TAC 在网络中不允许同时存在两个以上的实例。服务器端如发现这种情况，主动向 TAC 发送关闭命令，TAC 便会强制性关闭整个应用程序。

如在使用 TAC 的过程中遇到任何致命错误，TAC 都会强制性关闭应用程序，可以通过查看 LOG 文件获取详细错误信息。

例程：

```
main( )
{
    ini ilReturn;
```

```
    ilReturn=TACStart( );
    if(ilReturn= =TA_SUCCESS){
        /*    TAC started successfully */
        ......
    }
    else{
        /*    TAC started failed   */
        /*    See log file for further details        */
        ......
    }
    ......
}
```

5 TAPutMsg

int TAPutMsg(char *pcpDest,char *pcpMsg)

函数功能：

把需发送的数据内容放进 TAC 的发送队列里，并告诉 TAC 本条消息发送的目的地名称。TAC 以先来先服务的原则发送队列里的消息报文。

参数：

char *pcpDest	消息发送的目的地名称。
char *pcpMsg	消息数据内容。

消息发送的目的地名称是一个长度不超过 10 的字符串。

消息数据内容是一个长度不超过 4096 的字符串。

返回值：

TA_SUCCESS	消息成功压队
TA_QUEUE_FULL	发送队列满
TA_FAIL	消息压队过程中发生错误（tac.log 文件记录错误的详细原因）

在以下几种条件下，**TAPutMsg(…)**返回 TA_FAIL：

1 未调用 **TAStart( )**函数或调用返回 TA_FAIL。

2 参数中有 NULL 指针。

3 参数指针所指内容为空。

备注：

无论是在网络正常或存在网络故障的情况下，只要客户端发送队列不处于队列满的状态，应用程序都可以调用 **TAPutMsg(…)**函数把需要发送的消息报文放进发送队列，一切的发送和管理工作由 TAC 负责。

（char *pcpDest）应该指向一个长度不超过 10 的字符串，如果长度超过 10，TAC 自动截取前 10 个字符做为发送的目的地。

TAC 默认每条消息内容的最大字符长度为 4096。如果消息内容超过最大长度，TAC 自动截取前 4096 个字符发送。

发送队列默认的最大长度为 4096，用户可以通过修改 TAC.ini 来修改该配置。如果是由于发送队列满而返回 TA_ QUEUE_FULL，LOG 文件不会有任何相关错误记录。

例程：

```
main( )
{
    ini ilReturn;
```

```
    ......
    /*    TAC already start successfully       */
    ilReturn= TAPutMsg ("TAS"," Message of TAC");
    if(ilReturn= =TA_FAIL){
        /*    Message put failed  */
        /*    See log file for further details        */
        ......
    }
    ......
}
```

6 TAPutMsgExt

int TAPutMsgExt(char *pcpMsg)

函数功能：

把需发送的数据内容放进 TAC 的发送队列里，消息发送目的地为 TAC 默认消息目的地（配置文件中指定的 Dest）。TAC 以先来先服务的原则发送队列里的消息报文。

参数：

char *pcpMsg　　消息数据内容。

消息数据内容是一个长度不超过 4096 的字符串。

返回值：

TA_SUCCESS	消息成功压队
TA_QUEUE_FULL	发送队列满
TA_FAIL	消息压队过程中发生错误（tac.log 文件记录错误的详细原因）

在以下几种条件下，**TAPutMsgExt(…)**返回 TA_FAIL：
1 未调用 **TAStart( )**函数或调用返回 TA_FAIL。
2 参数中有 NULL 指针。
3 参数指针所指内容为空。

备注：

无论是在网络正常或存在网络故障的情况下，只要客户端发送队列不处于队列满的状态，应用程序都可以调用 **TAPutMsgExt(…)**函数把需要发送的消息报文放进发送队列，一切的发送和管理工作由 TAC 负责。

TAC 默认每条消息内容的最大字符长度为 4096。如果消息内容超过最大长度，TAC 自动截取前 4096 个字符发送。

发送队列默认长度为 4096，用户可以通过修改 TAC.ini 来修改该配置。如果是由于发送队列满而返回 TA_ QUEUE_FULL，LOG 文件不会有任何相关错误记录。

例程：

```
main( )
{
    ini ilReturn;

    ......
    /*    TAC already start successfully      */
```

```
        ilReturn= TAPutMsgExt (" Message of TAC");
        if(ilReturn= =TA_FAIL){
            /*    Message put failed  */
            /*    See log file for further details        */
            ......
        }
        ......
}
```

7 TAGetMsg

int TAGetMsg(char* pcpOrg,char* pcpMsg)

函数功能：

TAC 从接收队列中读取队列头消息报文，把消息的发送者信息记录在 pcpOrg 中，把数据内容记录在 pcpMsg 中。

参数：

char *pcpOrg	消息报文的来源
char *pcpMsg	消息数据内容

消息报文的来源是一个长度不超过 10 的字符串。

消息数据内容是一个长度不超过 4096 的字符串。

返回值：

> 0	消息报文数据区长度
TA_QUEUE_EMPTY	接收队列中无消息数据
TA_FAIL	读取消息失败（tac.log 文件记录错误的详细原因）

在以下几种条件下，**TAGetMsg( )**返回 TA_FAIL：

1 未调用 **TAStart( )**函数或调用返回 TA_FAIL。

2 参数中有 NULL 指针。

备注：

TAC 把所有从服务器发送过来的消息报文放在接收队列中，应用程序可以在任何时候通过成功调用 **TAGetMsg(…)**函数读取消息数据，并得知消息数据的来源。然后，TAC 就会把队列头的消息报文从接收队列中删除。

接收队列默认的最大长度为 4096，在接收队列满的情况下，如再次接收到服务器端发送的消息，TAC 拒绝接收新发送的消息。为了尽量避免接收队列满的情况发生，应用程序应该尽快地读取接收队列的消息。

（char *pcpOrg）应该指向一个长度至少为 11 的字符串空间，如果长度不足，可能发生内存空间访问错误。

（char *pcpMsg）应该指向一个至少为 4097 的字符串空间，如果长度不足，可能发生内存空间访问错误。

如果是由于接收队列空而返回 TA_QUEUE_EMPTY，LOG 文件不会有任何相关错误记录。

例程：

```
main( )
{
    ini ilReturn;
```

```
    char pclOrg[11];
    char pclMsg[4097];

    ......
    /*    TAC already start successfully       */
    ilReturn= TAGetMsg(pclOrg,pclMsg);
    if(ilReturn>0){
        /*    Message get successfully*/
        ......
    }
    else if(ilReturn= = TA_FAIL){
        /*    Message get failed  */
        /*    See log file for further details        */
        ......
    }
    ......
}
```

8 TAGetMsgExt

int TAGetMsgExt(char* pcpMsg)

函数功能：

TAC 从接收队列中读取队列头消息报文，把数据内容记录在 pcpMsg 中。

参数：

char *pcpMsg　　消息数据内容

消息数据内容是一个长度不超过 4096 的字符串。

返回值：

> 0	消息报文数据区长度
TA_QUEUE_EMPTY	接收队列中无消息数据
TA_FAIL	读取消息失败（tac.log 文件记录错误的详细原因）

在以下几种条件下，**TAGetMsgExt(…)**返回 TA_FAIL：
1 未调用 **TAStart( )**函数或调用返回 TA_FAIL。
2 参数中有 NULL 指针。

备注：

TAC 把所有从服务器发送过来的消息报文放在接收队列中，应用程序可以在任何时候通过成功调用 **TAGetMsgExt(…)**函数读取消息数据，并得知消息数据的来源。然后，TAC 就会把队列头的消息报文从接收队列中删除。

接收队列默认的最大长度为 4096，在接收队列满的情况下，如再次接收到服务器端发送的消息，TAC 拒绝接收新发送的消息。为了尽量避免接收队列满的情况发生，应用程序应该尽快地读取接收队列的消息。

（char *pcpMsg）应该指向一个至少为 4097 的字符串空间，如果长度不足，可能发生内存空间访问错误。

如果是由于接收队列空而返回 TA_QUEUE_EMPTY，LOG 文件不会有任何相关错误记录。

例程：

```
main( )
{
    ini ilReturn;
    char pclMsg[4097];

    . . . . . .
    /*    TAC already start successfully      */
    ilReturn= TAGetMsgExt(pclMsg);
```

```
        if(ilReturn>0){
            /*    Message get successfully*/
            ......
        }
        else if(ilReturn= = TA_FAIL){
            /*    Message get failed  */
            /*    See log file for further details        */
            ......
        }
        ......
}
```

9 TAGetLinkStatus

int TAGetLinkStatus()

函数功能：

获取 TAC 与服务器端的网络连接状态。

返回值：

TA_LINKOK　　网络连接正常
TA_LINKERROR　网络连接错误

如果应用程序未调用 **TAStart( )**函数或调用返回 TA_FAIL，调用 **TAGetLinkStatus( )**函数也会返回 TA_LINKERROR。

备注：

TAC 启动后，自动对整个网络状态进行监视，遇到了网络故障，TAC 可以智能地与服务器端进行重新连接。并继续接收和发送队列中的消息数据。

例程：

```
main( )
{
    ini ilReturn;

    ……
    ilReturn= TAGetLinkStatus( );
    if(ilReturn= = TA_LINKOK){
        /*    Link status is OK    */
        ……
    }
    else{
        /*    Link encounter some error     */
        ……
    }
}
```

10 TAClose

int TAClose( )

函数功能：

关闭已启动的 TAC，释放占用资源，关闭相关 socket。

返回值：

TA_SUCCESS	TAC 关闭成功
TA_FAIL	TAC 关闭失败（tac.log 文件记录错误的详细原因）

如果未调用 **TAStart( )**函数或调用返回 TA_FAIL，则 **TAClose( )**函数调用返回 TA_FAIL。

备注：

应用程序被关闭前，系统会自动调用 **TAClose( )**函数。但推荐用户在应用程序中显式调用 **TAClose( )**关闭已启动的 TAC。

如果由于异常错误导致程序中断，请先检查操作系统中是否有未结束的相关进程资源，并且确保在关闭这些进程后才重新启动应用程序。

例程：

```
main( )
{
    ini ilReturn;

    ……
    /*    TAC already start successfully       */
    ilReturn= TAClose( );
    if(ilReturn！=TA_SUCCESS){
        /*    TAC close successfully   */
        ……
    }
    else{
        /*    TAC close failed     */
        /*    See log file for further details        */
        ……
    }
    ……
}
```

11 日志文件

TAC 首次启动时会在应用程序当前目录下创建日志文件(tac.log)，负责记录 TAC 的运行日志，记载报错信息。

12 配置文件

配置文件(tac.ini)必须在应用程序同一目录下，TAC 在调用 **TACStart( )**函数时读取配置文件的内容。修改过的配置文件内容，只有在下次调用 **TACStart( )** 后才能生效。

[GENERAL]

GENERAL 中每个配置项都是必须的。TAC 读取配置文件的过程如果发现必须配置项不存在或配置错误，都会返回操作失败，TAC 无法正确启动。

Name=TAC

客户端名称，不允许为空，也不允许超过 10 个字符。每个客户端的名称都必须和服务端的配置文件中客户端名称对应，客户端不可以随便定义名称，服务器端不处理非法客户端发送的消息。

Dest=TAS

客户端默认的消息发送目的地名称，不允许为空，也不允许超过 10 个字符。和 Name 配置项一样，目的地名称也必须是合法的客户端名称或是服务器端名称。如 Dest 配置的内容不合法，服务器端会视之为非法报文，作抛弃处理。

ServerIP=10.200.200.200

点分十进制数形式的服务器端 IP 地址。

IndexPort=30001

服务器端的公共服务端口，由服务器端配置决定。

[OPTION]

OPTION 中每个配置项都是可选的。每个配置项 TAC 都有一个默认值，如果把某项从配置文件中删除或设置内容不在允许范围内，则 TAC 把此项设为默认值。

TcpTimeout=5000

TCP 事件的等待时间，单位毫秒(ms)。最小值 5000 毫秒， 最大值 300，000 毫秒。TAC 默认为 5000 毫秒。

MaxCnnRetries=3

在调用 **TACStart( )**过程中，TAC 尝试建立与服务器端的网络连接。如果连接失败次数超过 MaxCnnRetries，则 TAC 放弃再次连接。MaxCnnRetries 最小值为 1，最大值为 1024。TAC 默认为 3。

13 错误代码描述

如在运行过程中发生系统级错误，TAC 在 LOG 中记录 WINDOWS 的错误代码。通过下面的错误代码描述可以查看 SOCKET 相关的详细错误原因。其他类型的错误代码描述可以在 MSDN Library 中查看。

Windows Sockets code	Error	Description
WSAEINTR	10004	Interrupted system call.
WSAEBADF	10009	Bad file number.
WSEACCES	10013	Permission denied.
WSAEFAULT	10014	Bad address.
WSAEINVAL	10022	Invalid argument.
WSAEMFILE	10024	Too many open files.
WSAEWOULDBLOCK	10035	Operation would block.
WSAEINPROGRESS	10036	Operation now in progress. This error is returned if any Windows Sockets API function is called while a blocking function is in progress.
WSAEALREADY	10037	Operation already in progress.
WSAENOTSOCK	10038	Socket operation on nonsocket.
WSAEDESTADDRREQ	10039	Destination address required.
WSAEMSGSIZE	10040	Message too long.
WSAEPROTOTYPE	10041	Protocol wrong type for socket.
WSAENOPROTOOPT	10042	Protocol not available.
WSAEPROTONOSUPPORT	10043	Protocol not supported.
WSAESOCKTNOSUPPORT	10044	Socket type not supported.
WSAEOPNOTSUPP	10045	Operation not supported on socket.
WSAEPFNOSUPPORT	10046	Protocol family not supported.
WSAEAFNOSUPPORT	10047	Address family not supported by protocol family.
WSAEADDRIN	10048	Address already in use.
WSAEADDRNOTAVAIL	10049	Cannot assign requested address.

WSAENETDOWN	10050	Network is down. This error may be reported at any time if the Windows Sockets implementation detects an underlying failure.
WSAENETUNREACH	10051	Network is unreachable.
WSAENETRESET	10052	Network dropped connection on reset.
WSAECONNABORTED	10053	Software caused connection abort.
WSAECONNRESET	10054	Connection reset by peer.
WSAENOBUFS	10055	No buffer space available.
WSAEISCONN	10056	Socket is already connected.
WSAENOTCONN	10057	Socket is not connected.
WSAESHUTDOWN	10058	Cannot send after socket shutdown.
WSAETOOMANYREFS	10059	Too many references: cannot splice.
WSAETIMEDOUT	10060	Connection timed out.
WSAECONNREFUSED	10061	Connection refused.
WSAELOOP	10062	Too many levels of symbolic links.
WSAENAMETOOLONG	10063	File name too long.
WSAEHOSTDOWN	10064	Host is down.
WSAEHOSTUNREACH	10065	No route to host.
WSASYSNOTREADY	10091	Returned by WSAStartup(), indicating that the network subsystem is unusable.
WSAVERNOTSUPPORTED	10092	Returned by WSAStartup(), indicating that the Windows Sockets DLL cannot support this application.
WSANOTINITIALISED	10093	Winsock not initialized. This message is returned by any function except WSAStartup(), indicating That a successful WSAStartup() has not yet been performed.
WSAEDISCON	10101	Disconnect.
WSAHOST_NOT_FOUND	11001	Host not found. This message indicates that the key (name, address,

		and so on)was not found.
WSATRY_AGAIN	11002	Nonauthoritative host is not found This error may suggest that the name service itself is not functioning.
WSANO_RECOVERY	11003	Nonrecoverable error. This error may suggest that the name service itself is not functioning.
WSANO_DATA	11004	Valid name, no data record of requested type. This error indicates that the key (name, address, and so on) was not found.