ICE iPush® Communication Server

Embedded

Remote Management Programming Guide For WinCon-8000

By: ICE Technology Corp., Feb 11, 2004

Ver: 1.1.3

E-Mail \ service@icetechnology.com Tel \ +886-2-23961880 Fax \ +886-2-23961881

Content

Introduction

Remote Management System was made up by four parts : Client Side Application (RAdm.exe), Client Side's Plug-In, Service Side Service (RAdmAgent.dll), Server Side's Plug-in. First, RAdm.exe and RAdmAgent.dll are the key parts of Remote Management System, they are the backbone of message exchange. Based on request/response message exchange pattern, RadmAgent.dll will send response message back to Radm.exe only after received the request message from Radm.exe

RAdm.exe and RAdmAgent.dll will take care the task of message transportation, but they won't resolve the content of message. The format of message and the meaning of message are decided by client side plug-in and server side plug-in, and those 2 plug-in could be implement by developer itself. Client side plug-in will be loaded after RAdm.exe startup, plug-in developer have to implement the presentation of user interface (in other words, plug-in must create a child window), and the specific program interface to cooperate with RAdm.exe in processing response message from RAdmAgent.dll and sending request message . Server side plug-in will be loaded after RAdmAgent.dll received request message, After RAdmAgent.dll received a message, RAdmAgent.dll will pass it to plug-in to process the content of this message.RAdmAgent.dll will send back a response message that contains the result of processing In order to cooperate with RAdmAgent.dll in processing request message and sending response message, Server side plug-in must implement specific interface too.

Following is detail explanation for client side plug-in and server side plug-in.

Client Side Plug-in

Client side plug-in will use 3 callback function as fellow:

1. void __stdcall *PLUGIN_CALLBACK_FUNC(long nResult,
   long UserData,
   const char* Buf,
   long nBufLen)

   This callback function was implemented and provided by plug-in (Function name is just a placeholder). After RAdm.exe received response, it will callthis function. If request message successfully received and processed, the first parameter nResult will be greater then zero and the 3rd , 4th parameter will be response message content and content length. Second parameter is user-defined data, plug-in developer could use it to cooperate with following RAdm.exe provided callback function.

2. void __stdcall *SENDREQUEST(long ContextID,
   long UserData,
   const char* szServerPlug-inID,
   const char* MsgContent,
   long MsgLength,
   PLUGIN_CALLBACK_FUNC PluginCallBack)

   This call back function will be implemented and provided by RAdm.exe. Plug-in could use this function to send request message. The first parameter is Context ID, this parameter will be provided to plug-in by RAdm.exe after plug-in was initialized. Second parameter is UserData, This parameter is provided by plug-in, This value will be the same with PLUGIN_CALLBACK_FUNC function's second parameter when

PLUGIN_CALLBACK_FUNC is callback in plug-in. Third parameter is the name of server side plug-in(the file path must not be included, because those file have to put on system directory for centralized management). The 4th and the 5th is the transfer message content and length. 6th parameter is response callback function address, this parameter have to be provided by plug-in.

3. void __stdcall *TRANSMITFILE(long ContextID,
   long UserData,
   const char* szLocalFileName,
   const char* szRemoteFileName,
   PLUGIN_CALLBACK_FUNC)

   This function will be implemented and provided by RAdm.exe. Plug-in could use this function to transfer file Please note: Remote Management only transfer file to specific server directory. Can NOT transfer file to arbitrary directory on server. Function's 1st and 2nd parameter is same with **SENDREQUEST**, 3rd parameter is client side file name to be transferred, 4th parameter is the file name will save on server (RAdmAgent.dll will automatic append specific path nameto construct complete path name). 5th parameter explanation as **SENDREQUEST**.

Previous 3 function will be used when developer implement client side plug-in. And, plug-in have to implement and export other 3 function as follow:

1. void Initialize(long ContextID,
   SENDREQUEST SendRequest,
   TRANSMITFILE TransmitFile)

   RAdm.exe will using this function to pass **ContextID**, **SENDREQUEST**, **TRANSMITFILE** to plug-in. This function will be called only once after loaded up.

2. bool EnumUI(int* pCount, long* puid)

   RAdm.exe will use this function to query plug-in the number of management windows implements, and the identification of those implement. When puid is NULL, means query the window number. Plug-in should pass window number by pCount number. When puid not NULL, *pCount is LONG type element number, puid is the array start address.

3. HWND ActivateUI(long uid,HWND hParent)

   RAdm.exe will using this function to ask plug-in to generate child windows, and return child windows handle. The first parameter is management window implement ID, this id is get from calling EnumUI. Second one is the parent window handle of child window.

Server Side Plug-in

Server side plug-in have to implement and export 3 functions as fellow.

1. void Initialize(long* major, long* minor)

   This function will be called after plug-in loaded, plug-in can do initialize at this moment, then return plug-in version information by 1st and 2nd parameter.

2. void UnloadPlugIn()

   This function will be called after plug-in unloaded.

3. bool ProcessMsg(PLUGIN_MESSAGE* msg)

   RAdmAgent.dll will use this function send request message to plug-in, and get message respond form PLUGIN_MESSAGE struct. Return False means plug-in can NOT handle that message.

   PLUGIN_MESSAGE struct as fellow:

```
struct PLUGIN_MESSAGE
{
    long ConnID;
    long TotalBytes;
    long StgHandle;
    WRITEDATA   WriteData;
    TRANSMITFILE TransmitFile;
};
```

   First field **ConnID** is **ContextID**, it will be used at 4th and 5th field's callback function. 2nd field **TotalBytes** will point out the message length comes in. 3rd field **StgHandle** is file handle, Because RAdmAgent.dll will temporary save incoming message into files, so plug-in need to use **ReadFile** to read incoming message (Warning, DO

NOT USE **CloseHandle** on this file handle). 4th and 5th is supported callback functions provided by RAdmAgent.dll, plug-in can use these functions to send response to RAdmAgent.dll. These 2 callback function are:

```
bool __stdcall  WriteData(long ContextID,
                          const char* MsgContent,
                          long MsgLength,
                          long* BytesWritten)
```

First field must be **PLUGIN_MESSAGE** 's **ConnID**, 2nd and 3rd parameter is address and length of plug-in message. 4th parameter is this actual data length.

```
bool __stdcall TransmitFile(long ContextID,
                            char* szFileName)
```

First field must be PLUGIN_MESSAGE 's ConnID , 2nd Parameter is File name need to transfer. When the request is looking for specific server side file content, this function could send file content back as response message to client.

Example

ClientSample and **ServerSample** are 2 example program run at Remote Management framework. ServerSample.dll need to be put at WinCon8000 \Compact Flash\IceTechnology\RAdm directory. ClientSample.dll need to be put under same directory of RAdm.exe, and add new file entry “ClientSample.dll” at Plugins.ini.

ServerSample is a server side plug-in. Because these sample don’t need User Interface, so we just need to implement Initialize, UnLoadPlugIn, ProcessMsg function. In the example, We only handle “gettime” message and return system clock on WinCon8000, others return “Command not Support” message.

Client Side plug-in is a little bit harder to implement, because we have to handle some miscellaneous item. But by using MFC, we can quickly build VC application. Although we write a dll plug-in, but we still can generate windows, and ClientSample is made up by Visual C++. Program shows how to generate a child window when implement ActiveUI, ClientSample use modeless dialog box as child window, and using class wizard generated related window messaging handle function.