

AlarmManager

Reference Manual

AlarmManager Reference

Copyright © 2003-2005 Insignis Technologies. All rights reserved worldwide.

Created: 2006-04-05 by: Giuseppe Recalcati
Last modified: 2006-04-06 by: Giuseppe Recalcati Revision: 1

Copyright

This manual is the intellectual property of Insignis Technologies and is protected by copyright. All rights are reserved. No part of this document may be reproduced or transmitted for any purpose, by whatever means, be they electronic or mechanical, without the express written permission of Insignis Technologies.

Note

This manual was compiled with the greatest of care and all information double checked. At the time of printing the description was complete and correct. Because of the further development of products, the content of the manual might change without prior notice. Insignis Technologies will not be liable for damage which is directly or indirectly due to errors, incompleteness, or discrepancies between the manual and the product described.

Trademarks

All names used in this manual for hardware and software could be registered trademarks and must be treated as such.





Table of contents

Overview.....	6
Data Structure.....	7
AlarmManager Messaging Approach	8





Overview

AlarmManager is a client application that is part of the CIEFFE client-side software, the application is responsible of receiving and dispatching server alarms.

Inside this reference manual explained is the structure of the data dispatched by the AlarmManager application to the defined client application and the messaging approach used by the AlarmManager in dispatching this data.

The AlarmManager default settings are available inside the registry key "HKEY_LOCAL_MACHINE\SOFTWARE\Insignis Technologies\Alarm Manager".

Some of the application settings are the following:

- STRING "Default App Executable": the path for the defined application that receives alarms notification from AlarmManager;
- STRING "Default App WorkDir": the working directory for the defined application that receives alarms notification from AlarmManager;
- STRING "Default App Description": Gives a name to the defined application that receives alarms notification from AlarmManager;
- STRING "Automatic App Startup": boolean value (True or False) specifying if the AlarmManager is allowed to startup the defined application when alarms to dispatch have been enqueued;
- STRING "Show Notifications": boolean value (True or False) specifying if the AlarmManager is allowed to display notifications for inability to dispatch alarms to the defined application.

Data Structure

Following is the notification data structure sent by the AlarmManager to the defined client application:

```
struct ClientAlarmData
{
    BYTE          login[10];
    BYTE          passwd[10];
    WORD          port;
    DWORD         cameras;
    DWORD         alarmid;
    BYTE          mainaddress[16];
    DWORD         serial;
    struct ServerAddress
    {
        DWORD     address;
        BYTE      lowbandwidth;
    } ipaddress[6];
    WORD         numaddresses;
    DWORD        extsize;
};
```

where:

- **login**: the alarm configured user login to the server;
- **passwd**: the alarm configured password to use to access the server (if supplied);
- **port**: the server port;
- **cameras**: a bitmask containing the cameras recorded mask for the specified alarm;
- **alarmid**: the 0th based index of the alarm;
- **mainaddress**: the server main ip address (string);
- **serial**: the server serial identifier;
- **ipaddress**: an array of 6 `ServerAddress` structure containing the 32bit ip address and a low bandwidth flag (RAS flag);
- **numaddresses**: the number of addresses configured inside the `ipaddress` array;
- **extsize**: the size of the data following the `ClientAlarmData` structure. This data are implementation dependent and are not explained in this manual.

Messaging Approach

The AlarmManager approach in sending notification messages to the defined client application uses the client application main window messages queue.

When the AlarmManager processes an alarm notification, it packs the notification data into a `ClientAlarmData` structure and sends this data to the defined client application using the following message:

```
#define ALARMMANAGERMSG_ACTIVATE 13900
```

This message is sent by the AlarmManager application using the `SendMessage(...)` function. The client application that receives this message should answer to this message with a `ReplyMessage(1)` to notify that it has processed the message.

As previously discussed, the data sent to the defined client application is a `ClientAlarmData` structure. To send this data structure the AlarmManager application uses a `COPYDATASTRUCT` structure, so the client application should process the event in the following way:

```
INT_PTR CALLBACK WndProc(HWND hwnd,UINT msg,WPARAM wparam,LPARAM lparam)
{
    switch(msg)
    {
        case ...
        {
            ...
        }
        break;
        .....
        case WM_COPYDATA:
        {
            switch(wparam)
            {
                case ALARMMANAGERMSG_ACTIVATE:
                {
                    ReplyMessage(1L);
                    COPYDATASTRUCT* data = (COPYDATASTRUCT*)lparam;
                    ClientAlarmData* alarmdata = (ClientAlarmData*)data->lpData;
                    .....
                }
                break;
            }
        }
        break;
        .....
        default:
        {
            ...
        }
        break;
    }
    return DefWindowProc(hwnd,msg,wparam,lparam);
}
```



Remarks

Experience always drives to the right way in building applications, with the current version of the AlarmManager application and the relative approach in posting notifications to the defined client application, there are obviously some guidelines you should follow, the main one is discussed below:

Since the number of messages processed by the AlarmManager application and dispatched to the defined client application can be up to thousands, the defined client application should take care on managing these notifications. In particular you should avoid notifications management inside the client application Window Procedure since this can slow down the client application and the AlarmManager application too.

Remember that, while a receiving application is processing a `SendMessage(...)` message, the sender application is blocked waiting that the receiving application has finished in managing the message.

To work around this problem the AlarmManager application uses the `SendMessageTimeout(...)` function. Anyway the client application should be able to do its work inside the Window Procedure as fast as it can since taking too much time processing a message can cause the AlarmManager to always reach its `SendMessageTimeout(...)` timeout slowing down its dispatching thread and accumulating messages in the dispatch list. Since the dispatch list has a maximum amount of elements, the result is that not dispatched old notifications are trashed away with lost notifications.