

Screen Scraper Technology

This Paper was written by Sudharsan one of our former Senior Consultant. It was the result of research in connection with a project for Living Questions AB in Sweden. The document is an introduction to how screen scraper technology works

Author:

Sudharshan

Screen Scraper Technology

This Paper was written by Sudharsan one of our former Senior Consultant. It was the result of research in connection with a project for Living Questions AB in Sweden.

Abstract

If you are looking for a technology to interact with your existing software by means of extracting or posting data into its user interface without adding a single line of code in your existing application, you are at the right place. Using this technology it is possible to interact with an application, which was developed in any development tool under windows environment. However, there are certain rules and limitations, which will limit your ability to interact. You can see them listed at the limitation's section. This paper will take you through the steps you need to follow to develop a Screen Scraper program. You can find code listing with detailed description of different windows API and their working. This document will give you sufficient knowledge using which you will be able to develop an application implementing the Screen Scraper technology.

Table of Contents:

Abstract	2
Table of Contents:.....	3
Background.....	4
About the Author	4
Introduction	4
What is Screen Scraper Technique?.....	4
Steps to develop a Screen Scraper application	5
To extract or to post data.....	5
To capture keyboard or mouse click events	5
A Simple Program.....	5
Enumerating the windows	6
Selecting the Controls	7
Transfer data from Source to Destination controls	9
Note	10
Limitations.....	10
Conclusion.....	11
Appendix A: Glossary.....	12
Appendix B: For Further Reference	13

Background

About the Author

Mr Sudharsan is a Senior Consultant working with Gislen Software Pvt Ltd., He is having a development experience of 6 years. This document was a result of his and his team's effort in building an application called Screen Agent for their client Living Questions AB Sweden.

Introduction

Screen Scraper technology is a name given to a technique using which one can write a program

- To extract or post user data from windows enabled controls
- To capture keyboard or mouse click events
- To create and display windows enabled controls dynamically at a pre-defined location
- To define event handlers for the dynamically created controls at run time.

The above actions can be performed on any application's user interface which is running in the same desktop as that of the Screen Scraper program. The specialty of this technique is that this can be achieved without adding a single line of code in the application whose user interface is being accessed to perform one or more of the above operations.

In this article we will discuss about the technique associated with the Screen Scraper technology and how to implement it in Delphi under Windows environment. We will also discuss the limitations of this technology.

To understand the terminologies used in this article better, I would advice you to browse through the Appendix section once before you start.

What is Screen Scraper Technique?

Screen Scraper technique depends on the set of API's provided by the windows operating system. Windows provides a rich set of API's for message handling, using which we can interact with any application, which is active in the desktop. Screen Scraper deals with the user interface of the application. However, there is no need for that application to know what Screen Scraper program is going to do with its user interface. So without adding a single line of code in the client application the Screen Scraper program can interact with that application's screen.

There is no restriction on which development tool with which the client application can be developed. This gives wider coverage for the Screen Scraper program and possibly a desire to explore for the programmers.

Steps to develop a Screen Scraper application

To extract or to post data

The first step, the user has to identify the application and its user interface with which the Screen Scraper needs to interact. We call the selected application as the client application.

The second step is to identify the windows enabled control from which the data need to be extracted or posted as the case may be.

The third step is the actual transfer of data from the source to the destination. The source and the destination controls can be in different application's user interface.

To capture keyboard or mouse click events

The first step is the same as the above. The user has to identify the application and its user interface.

The second step is to define the key strokes or the combination of key strokes to be monitored in case of keyboard events and click or double click for right, left or middle buttons as the case may be for the mouse events.

The third step is to write the code to perform the desired action once the desired events are fired.

To create and display controls and to assign event handlers:

The first step as usual is to identify the application and its user interface.

The second step is to create a windows enabled control of the given type. Make it appear on the user interface of the client window with the attributes like top, left, width, height, caption etc., as set by the user.

The third step is to assign the event handler for the control.

A Simple Program

Let us write a simple program in Delphi to convert the above logical steps in to actual code and see how it works out.

Create a new application in Delphi. I have named it ScreenScraper1.0. The new application should have provision to select the source and destination client windows as well as the windows enabled controls. Fig 1 refers a window where you can define the said things. As you can notice in the Fig 1, the window has an "Execute" button. After defining the source and destination details, if you press this button you can see the data being transferred from the source to the destination.

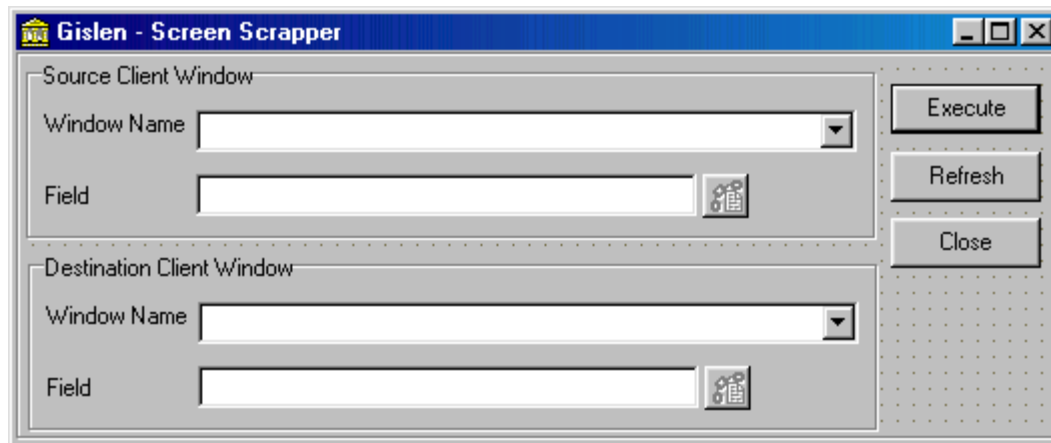


Fig 1 - Screen Scrapper

Let us define a source control from which data need to be extracted and a destination control to which the data can be posted. These source and destination controls may be in two different applications. So as a first step the user has to identify the source and the destination applications from the list of active applications in the desktop. To get the list of all active desktop windows we use a process called enumeration.

Enumerating the windows

Enumeration is the process of counting all the active desktop windows. EnumWindows() is the windows function which will get you all the active top level desktop windows. The following code will add the EnumWindows() function in the FormCreate() event handler of the above form. The call back function EnumWindowsProc() will be called for each and every active desktop widow identified by the EnumWindows(). So you can see inside EnumWindowsProc() the names of the windows are extracted and being added to the source and destination combo boxes of the above window.

```
function EnumWindowsProc(hwnd: HWND; lParam: LPARAM) : Bool; stdcall;
var
  sCaption : array[0..MAX_PATH] of char;
begin
  Result := True;
  sCaption := '';
  GetWindowText(hwnd, sCaption, MAX_PATH);
  if ((Length(trim(sCaption)) > 0) and (IsWindowVisible(hwnd))) then
  begin
    frmAdmin.cmbSource.Items.Add(sCaption);
    frmAdmin.cmbDestination.Items.Add(sCaption);
  end;
end;

procedure TfrmAdmin.FormCreate(Sender: TObject);
var
  lParam : LongInt;
  ini: TIniFile;
begin
  try
    lParam := 0;
    EnumWindows(@EnumWindowsProc, lParam);

    ini := TIniFile.Create('ScreenScrapper.ini');
    ini.WriteString('Handle', 'Source', IntToStr(edtSource.Handle));
    ini.WriteString('Handle', 'Destination',
```

```

        IntToStr(edtDestination.Handle));
    ini.Free;
except
    on E: Exception do
        MessageDlg(E.Message, mtInformation, [mbOk], 0);
    end;
end;

```

The EnumWindows function enumerates all top-level windows on the screen by passing the handle to each window, in turn, to an application-defined callback function. EnumWindows continues until the last top-level window is enumerated or the callback function returns FALSE.

The EnumWindowsProc function is an application-defined callback function used with the EnumWindows function. It receives top-level window handles. EnumWindowsProc is a placeholder for the application-defined function name.

The GetWindowText function copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied.

In the above code the enumeration of the window will happen during the form creation itself and the visible window names are added to two combo boxes, cmbSource and cmbDestination.

You can notice that the edtSource and edtDestination controls handles are being stored in an ini file. These are the controls, which are going to hold the reference to the source and destination controls of the respective client windows. You will see in the following code how these handles are used to create the references.

Selecting the Controls

Once a window is selected, the user needs to select the control with which he would like to interact (extract / post). Let us start with selecting the control from which the data need to be extracted (Source). It can be achieved in many ways. Let us say, whenever the user clicks the right mouse button over a control in a client window the data from that control should be extracted. This can be achieved by hooking the mouse messages.

To hook mouse messages (or any system messages for that matter) SetWindowsHookEx() function is used.

The syntax is:

```
SetWindowsHookEx(WH_MOUSE, @HookProcedure, hInstance, 0)
```

The first parameter specify the type of message hook. WH_MOUSE specifies that the hook is to capture mouse messages. The second parameter is the hook procedure. Whenever the hooked message occurs the control will be transferred to the hook procedure.

When you set a mouse hook as described above the hook procedure will be called for all the mouse movements and clicks. But we are interested only in right mouse button click. So inside the hook procedure check for the occurrence of WM_RBUTTONDOWN in the wParam.

```
if wParam = WM_RBUTTONDOWN then
```

```
{do the processing here}
```

Then use `GetActiveWindow()` function to get the handle of the selected control (i.e the control over which the user has clicked the right mouse button. If there is no control then the function will return the application handle).

Use `GetWindowText()` or `SendMessage` with `WM_GETTEXT` to extract the text which the control holds. This can in turn be passed to the Screen Scraper program using the details stored in the `ScreenScraper.ini` as discussed earlier. The `ScreenScraper.ini` will hold the handle of the controls where the Source and Destination details are to be stored. We can use `SendMessage` function with `WM_SETTEXT` to write the extracted data into the source or the destination control in the ScreenScraper program as the case may be.

To capture keyboard generated events you can use `WH_KEYBOARD` in place of `WH_MOUSE`. It is also possible to check for combination of key strokes like `Ctrl + any alphabet` or `Ctrl + Shift + any alphabet`. The following piece of code illustrates this.

```
{ check the 29 bit in the lParam to check if the Alt key is pressed }
if (lParam and 2684354560) = 2684354560 then
    sKeyboardResult := 'Alt + ';

{ if Ctrl key or Shift key is pressed then }
case wParam of
    VK_CONTROL : begin
        sKeyboardResult := 'Ctrl + ';
        bSend := False;
    end;
    VK_SHIFT   : begin
        sKeyboardResult := sKeyboardResult + 'Shift + ';
        bSend := False;
    end;
end;

{ check for the character key pressed }
For I := $41 to $5A do
    if I = wParam then
        begin
            sKeyboardResult := sKeyboardResult + Char(I);
            break;
        end;

    if not bSend then
        begin
            { if the previous key state is key down for the key that generated the
            key
            stroke then check for the 30th bit of lParam to check the previous
            key state }
            if (lParam and 1073741824) = 1073741824 then
                sKeyboardResult := '';
            end
        else
            begin
                if Length(sKeyboardResult) > 1 then
                    { send the message to the Screen Scrapper program }
                    SendMessage(hwndKeyboard, WM_SETTEXT, 0, LongInt(sKeyboardResult));
                sKeyboardResult := '';
            end;
        end;
    end;
```

The above code should be written in the call back hook procedure.

The first line in the above code sample checks the 29th bit of the lParam value, which is being passed to the call back hook procedure. If that bit is set then

```
if (lParam and 2684354560) = 2684354560 then
```

it means that the Alt key is in the pressed state.

The wParam can be checked to find out whether the user has pressed the Ctrl or the Shift button. The sKeyboardResult is a global variable. In the above code if the user has pressed only the Ctrl or Shift button the result won't be sent to the Screen Scraper program. Instead the keystrokes will be stored in the sKeyboardResult variable. Only if the user combines an alphabet with the above keystroke the result will be sent to Screen Scraper.

The statement "if (lParam and 1073741824) = 1073741824 then" will check whether there is continuity in the combination of keystrokes. For ex: it should be a continuous Ctrl + A and not Ctrl and after some time the keystroke of A. The above checking will prevent sending these types of broken keystrokes.

The above code will find out the key combination which the user has pressed over a window and will pass it back to the Screen Scraper. If you want to use a keyboard combination instead of right mouse button click as discussed earlier to select a control, and then include that checking accordingly in the above code listing.

Important: It is always advisable to have the hooking code in a separate .dll.

Transfer data from Source to Destination controls

Now you have selected the source and the destination control and the handle of these controls are stored in the ScreenScraper.ini file. The following code will transfer the data from the source to the destination control.

```
SendMessage(hSource, WM_GETTEXT, MAX_PATH, LongInt(sValue));
```

```
SendMessage(hDestination, WM_SETTEXT, 0, LongInt(sValue));
```

The hSource in the first SendMessage represent the handle of the source control which is read from the ScreenScraper.ini. WM_GETTEXT will instruct the windows to extract the data from the control to which hSource is pointing and will copy it to sValue. The second SendMessage will copy the extracted value in sValue to the control pointed by the hDestination handle. Again hDestination is the handle to the control to which the data need to be posted. This detail as discussed earlier is been read from the ScreenScraper.ini.

Create a window's control and make it appear on another application screen:

It is also possible to create a windows enabled control like a Button or a Text box during runtime and make it appear on a different application screen on a specific location. Let us take an example of creating a button named "My Button" with an onclick event which will display the "My Button" message.

The following code will create a button during runtime

```
MyButton := TButton.Create(nil);  
MyButton.ParentWindow := hwnd;  
MyButton.SetBounds(10, 10, 75, 25);  
MyButton.OnClick := MyClick;
```

Here MyButton is of type Tbutton. The first line will create an instance of type Tbutton. The second line is very important. The ParentWindow property will specify the window at which the button is going to be drawn. Hwnd will be the handle of that window over which the button is going to be displayed. If this hwnd is going to be the handle of the window, which was selected by the user, then button will appear on the user interface of that application. In fact we can set the location of display. SetBounds will set the x, y location and as well as the width and height of the button. MyClick is the onclick event handler for the button.

Note

However, if the client application where the button is being displayed is refreshed or closed and opened again, the button would not appear. You have to re-draw the button.

Limitations

This technique does have some basic requirement on other systems (Client Windows) to make it possible to work with them.

In this technique the client window is identified by its caption and its executable file name. There is a possibility that at a point of time there could be two instance of the same program running. So, it is better to inform the client about how this issue will be tackled.

You have to decide whether the search for client window is case sensitive or not before you start coding. It is better to mention the same in the design document.

When you have a window where caption can differ over time (whether in run time or every time the window comes up), you might loose track.

It is possible to identify the client windows, which are not visible in the desktop. But it is advisable not to identify them since they don't server any purpose.

The client window should have some basic characteristics without which we can't interact with them. Some of the important limitations are listed below

The control should be enabled, i.e. the user should be able to position the cursor over the control.

There might be some problem in working with multi line edit controls. Especially those with line feed or form feed characters.

- In this technique the client window is identified by its caption and its executable file name. There is a possibility that at a point of time there could be two instance of the same program running. So, it is better to inform the client about how this issue will be tackled.

- You have to decide whether the search for client window is case sensitive or not before you start coding. It is better to mention the same in the design document.
- When you have a window where caption can differ over time (whether in run time or every time the window comes up), you might lose track.
- It is possible to identify the client windows which are not visible in the desktop. But it is advisable not to identify them since they don't serve any purpose.
- The client window should have some basic characteristics without which we can't interact with them. Some of the important limitations are listed below
 - The control should be enabled, i.e. the user should be able to position the cursor over the control.
 - There might be some problem in working with multi line edit controls. Especially those with line feed or form feed characters.
 - Does not work with grids.
 - Does not work with calendar controls which are used to display dates
 - Does not work with any type of list box (check, dir, outline, list view)
 - Does not work with controls which can't take the cursor focus (ex: toolbar, tool button, speed button)
 - Does not work with most special type controls which are used to display text's and images and which are not editable.
- It is common that a window might have some container controls. It is possible to add a window enabled control (like a button) to that container, provided we get the handle of it.

Conclusion

We have learnt a new technique, using which interaction with legacy or existing application without adding a single line of code is made possible. Hope you can make use of this technique in whatever way applicable in your projects.

Appendix A: Glossary

Client Application: An active desktop window, which has been selected by the Screen Scraper program.

Windows enabled control: A control, which is available for the users to access and had been registered in the Windows registry.

Windows hook: A mechanism, which enables you to capture all or specified messages which are processed for any valid desktop process. It is also possible to process that message either before or after it was processed by the original application.

Appendix B: For Further Reference

You can get help for most of the issues on MSDN. The syntax or the explanation for API etc. can be found with detailed description.

You can also refer to the following sites for further information.

<http://www.delphi32.com/>

<http://www.tempest-sw.com/opentools/debugger.html>

<http://delphi.icm.edu.pl/>

<http://www.drbob42.com/articles/home.htm>