Implementing web application

Project Date	**September 2022**
Team ID	**PNT2022TMID34297**
Project Name	**Containment Zone Alerting Application**

Create UI to interact with applications

USER INTERFACE

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction. UI can be graphical, text-based, audiovideo based, depending upon the underlying hardware and software combination.

The user interface (UI) is the space where interactions between humans and machines occur. UI is an integral aspect of user experience (UX) that consists of two major parts: visual design, which conveys the look and feel of a product; and interaction design, which is the functional and logical organization of elements. The goal of user interface design is to create a user interface that makes it easy, efficient, and enjoyable for users to interact with a product. Learn about the different types of UI as well as fundamental design requirements for each type.

Basics of a UI Automation Client Application

UI Automation has a new COM client API for both unmanaged and managed clients. Unmanaged applications can now use UI Automation without requiring a change in languages or loading the common language runtime (CLR). Creating the CUIAutomation Object

- Include Uiautomation.h in your project headers. This header will bring in the other headers that define the API.
- Include Uiautomation.h in your project headers. This header will bring in the other headers that define the API.
- Initialize COM.
- Create an instance of the CUIAutomation class and retrieve the IUIAutomation interface in your global pointer.

The following example function creates the object instance and stores the retrieved interface address in the global pointer g_pAutomation:

```
HRESULT InitializeUIAutomation()

{
    CoInitialize(NULL);
    HRESULT hr =
```

```
        CoCreateInstance(__uuidof(CUIAutomation),
              NULL, CLSCTX_INPROC_SERVER,
             __uuidof(IUIAutomation),
(void**)&g_pAutomation);      return (hr);
}
```

Directly Obtaining UI Automation Elements

- The UI is modeled as a tree of automation elements
  (IUIAutomationElement interface objects), where each element represents a single piece of UI. The IUIAutomationElement interface has methods relevant to all controls, such as checking properties or setting focus.

- If you have screen coordinates-such as the cursor position in this example-you can retrieve an IUIAutomationElement interface by calling the IUIAutomation::ElementFromPoint method.

- To retrieve an IUIAutomationElement interface from a window handle (HWND), call the IUIAutomation::ElementFromHandle method

- You can retrieve an IUIAutomationElement interface that represents the focused control by calling the IUIAutomation::GetFocusedElement method.

Using Tree Walkers

- Raw or unfiltered, which shows all elements.

- Control view (the default), which filters out elements that either are redundant or are just used for layout.

- Content view, which filters controls even more selectively than Control view does.

The following simple example shows how to walk to the first child of the control referenced by pElement:

```
// Get the control view walker
IUIAutomationTreeWalker * pWalk;
g_pAutomation-
>get_ControlViewWalker(&pWalk);

// Go to the element's first child
IUIAutomationElement * pFirst;
pWalk->GetFirstChildElement(pElement, &pFirst);
```

UI Automation and Screen Scaling

- Make your client application aware of screen resolution by calling the Win32 function SetProcessDPIAware at startup. This function makes the entire process aware of screen resolution, so that no windows belonging to the process are scaled.

- Call GetPhysicalCursorPos to get the current cursor coordinates.

User Account Control and User Rights

This **UIAccess** attribute is included in the <requestedExecutionLevel> tag where the value of the **level** attribute is an example only, as follows:

```
<trustInfo xmlns=
    "urn:0073chemas-microsoft-com:asm.v3">
    <security>
        <requestedPrivileges>
        <requestedExecutionLevel
            level="highestAvailable"
            UIAccess="true" />
        </requestedPrivileges>
    </security>
</trustInfo>
```

The UIAccess attribute value is "false" by default; that is, if the attribute is omitted, or if there is no manifest for the assembly, the application will not be able to gain access to protected UI.

UI Automation and Threading :

Because of the way UI Automation uses Windows messages, conflicts can occur when a client application attempts to interact with its own UI on the UI thread. These conflicts can lead to very slow performance or even cause the application to stop responding.

<**DIV CLASS**="MORE-BUTTON">LISTING 1: THE PROGRAM GETS THE UI ELEMENT AT THE CURRENT CURSOR POSITION AND PRINTS ITS NAME AND CONTROL TYPE

```
int
_tmain(int argc, _TCHAR* argv[])
{
```

```
// Initialize COM and create the main Automation object
```

```
    IUIAutomation *g_pAutomation;
    CoInitialize(NULL);
    HRESULT hr = CoCreateInstance(__uuidof(CUIAutomation), NULL,
        CLSCTX_INPROC_SERVER, __uuidof(IUIAutomation),
        (void**)&g_pAutomation);
    if(FAILED(hr))
        return (hr);

    // Get the element under the cursor
    // Use GetPhysicalCursorPos to interact properly with
    // High DPI
    POINT pt;
    GetPhysicalCursorPos(&pt);

    IUIAutomationElement *pAtMouse;
    hr = g_pAutomation->ElementFromPoint(pt, &pAtMouse);
    if(FAILED(hr))
        return hr;

    // Get the element's name and print it
    BSTR name;
    hr = pAtMouse->get_CurrentName(&name);
    if(SUCCEEDED(hr))
    {
        wprintf(L"Element's Name: %s \n", name);
        SysFreeString(name);
    }

    // Get the element's Control Type (in the current languange)
    // and print it
```

```
    BSTR controlType;
    hr = pAtMouse->get_CurrentLocalizedControlType(&controlType);
    if(SUCCEEDED(hr))
    {
        wprintf(L"Element's Control Type: %s \n", controlType);
        SysFreeString(controlType);
    }

    // Clean up our COM pointers
    pAtMouse->Release();
    g_pAutomation->Release();
    CoUninitialize();
    return 0;
}
```

<DIV CLASS="MORE-BUTTON">LISTING 2: THIS METHOD USES THE RANGEVALUE PATTERN TO SET A CONTROL TO ITS MAXIMUM VALUE.

```
HRESULT TurnItUp(IUIAutomationElement *pElement)
{
    IUIAutomationRangeValuePattern * pRangeVal;

    HRESULT hr =
        pElement->GetCurrentPatternAs(UIA_RangeValuePatternId,
            __uuidof(IUIAutomationRangeValuePattern),
            (void **)&pRangeVal);

    if(FAILED(hr) || pRangeVal == NULL)
    {
        return hr;
    }
```

```
    double max;
    hr = pRangeVal->get_CurrentMaximum(&max);

    if(SUCCEEDED(hr))
    {
        hr = pRangeVal->SetValue(max);
    }

    pRangeVal->Release();
    return hr;
}
```

<DIV CLASS="MORE-BUTTON">LISTING 3: THE FINDFIRST METHOD HERE FINDS A PARTICULAR APPLICATION WINDOW.

```
IUIAutomationElement* GetTopLevelWindowByName(LPWSTR windowName)
{
    if (!windowName)
    {
        return NULL;
    }

    IUIAutomationElement* pRoot;
    IUIAutomationElement* pFound;
    VARIANT varProp;
    varProp.vt = VT_BSTR;
    varProp.bstrVal = SysAllocString(windowName);

    // Get the desktop element
    HRESULT hr = g_pAutomation->GetRootElement(&pRoot);
```

```
    // Get a top-level element by name, such as "Program Manager"
    if (pRoot)
    {
        IUIAutomationCondition* pCondition;
        g_pAutomation->CreatePropertyCondition(UIA_NamePropertyId,
            varProp, &pCondition);
        pRoot->FindFirst(TreeScope_Children, pCondition, &pFound);
        pRoot->Release();
        pCondition->Release();
    }

    VariantClear(&varProp);
    return pFound;
}
```