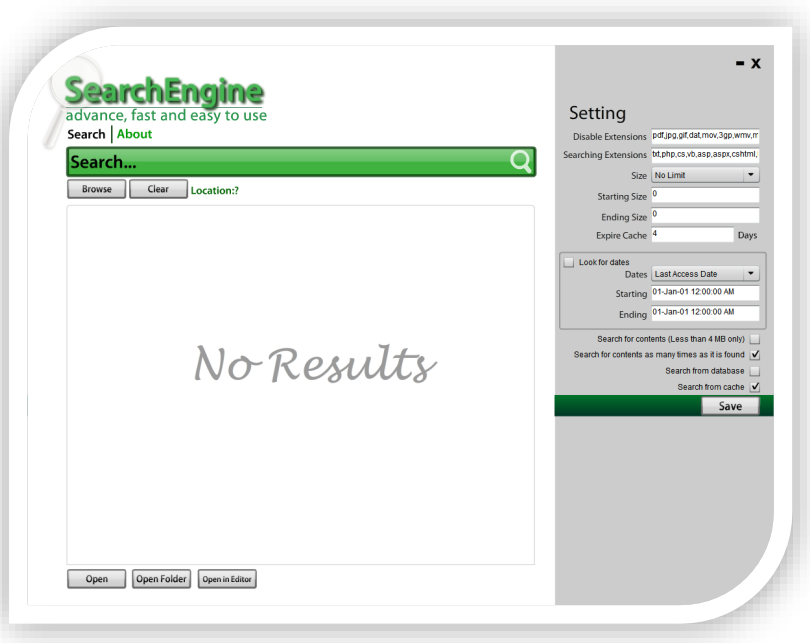


Search Engine

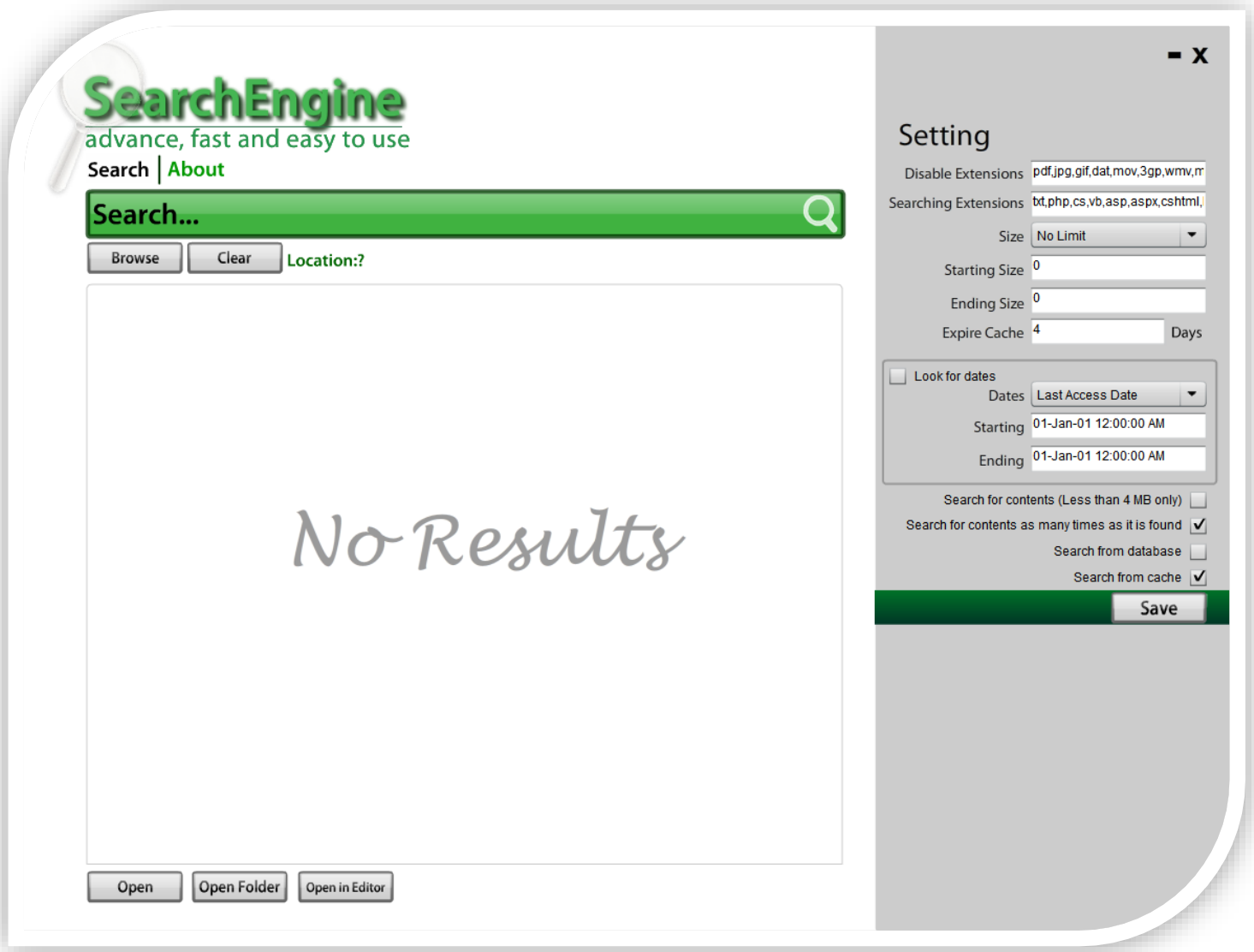
Advance and easy to use

ABSTRACT

Used Technologies: C#.NET 4, Flash CS5.5, Entity Framework, SQL Express and so on.

By Alim Ul Karim

CSE 373 Introduction to Algorithms

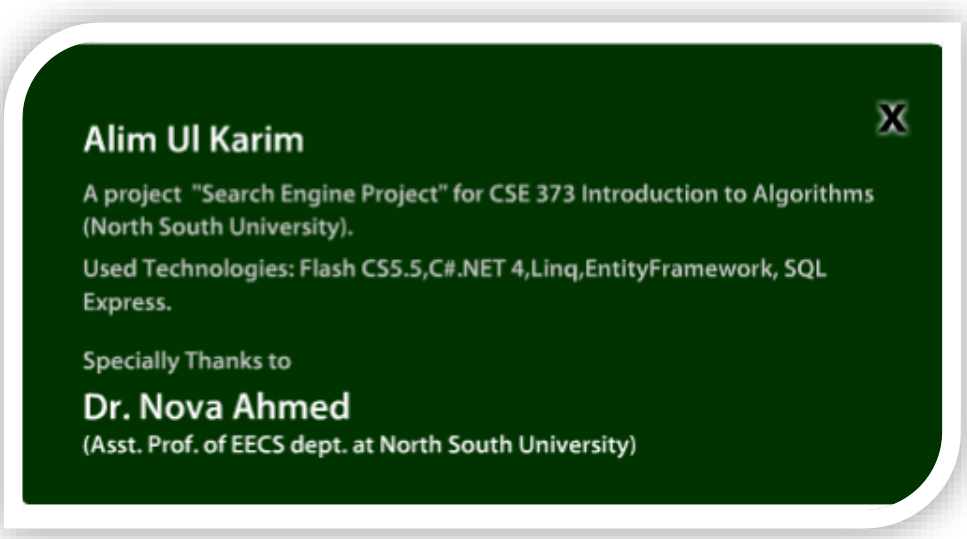


Search Engine Project Image

Acknowledgement

Firstly, I would like to take immense pleasure in thanking **Dr. Nova Ahmed** for permitting me to complete a project on Search Engine using C#.net.

It was a great experience throughout, even though the project is not successfully completed as I has hoped. However, I had learned a lot. Experience that I had gained throughout will not be a part of my life, if Dr. Nova hasn't permitted me in the first place. So all the credit goes to our excellent faculty of North South University **Dr. Nova Ahmed**.



Finally, I would like to express my heartfelt thanks to my beloved parents for their blessings and my friends/classmates for their help and wishes to try my best in this project.

Abstract

(How the information has been processed)

First I have used Flash CS5.5 to design the user interface. In addition, I have used C#.NET 4, Entity framework 5, Microsoft LinQ and SQL Express to develop this software. For core programming I have used pure C#.NET with its CLR libraries. To communicate with database I have using Microsoft LinQ and Entity framework. I have also made learning video on linq and entity framework 5. For additional helps I have used google, stackoverflow and so on.

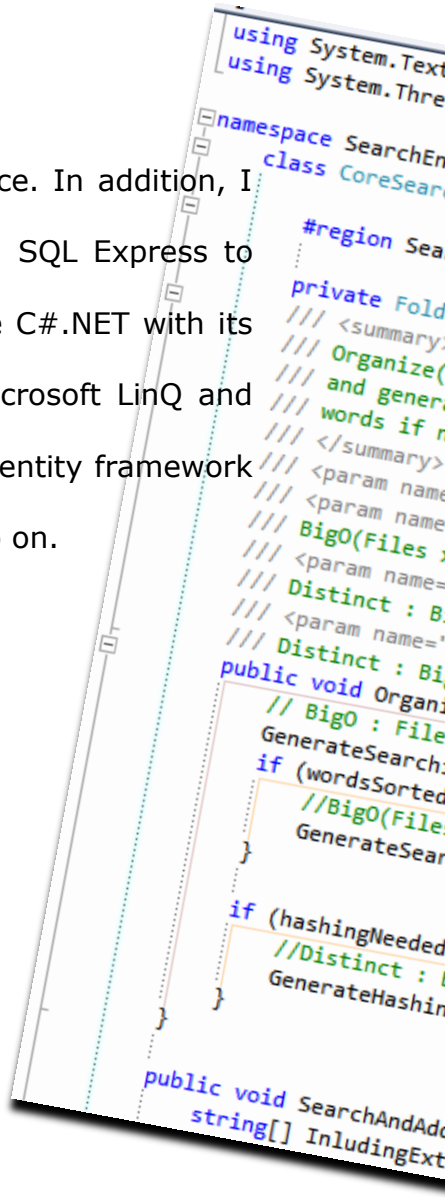


Table of Contents

Introduction

(What is the project is about)

The primary goal of the project was to make a search engine that can search text files data from a given folder. First, the search algorithm should look for the exact given text and then if not found then look for single parts without escape sequences (i.e. 'I','am','was','were' and so on) and then search for pluralize syntax or singularize syntax from a given dictionary. Finally, list the files as if found by those given search text.

Background

(History)

In CSE 225 Introduction to Data Structures and Algorithms course at North South University, I have made a similar type of project in C++, however it was not much robust and to make it better I have taken this opportunity.

First I took Microsoft WPF platform to design the user interface. I have never worked with WPF, and so after 7 days of wasting in learning and designing interface in WPF; next I realize that it wouldn't be possible to complete the project in time, if I don't use the technology that I already know of. Previously, I had 5 years of working experience with flash and as a result I switched my UI platform to flash.

Hypothesis

I think I will make a search engine or algorithm where search will not be $O(n)$ rather it would be less than that. In the processing term we are going to take all files inside all nested folder into the memory so it must be greater than $O(n)$ but when we are searching, it should be $O(n = words)$ to find that each word from the search terms**.** I should be able to outrun windows search using C#.NET 4 task parallel library and threading. It should be as flexible as windows search engine but better in searching terms with more options. It should be advance, fast and easy to use.

Coding in Flash

I have used flash to design my user interface with nice animations. It is really hard to explain in text to "how to make animation in flash" and mostly it would be redundant because there are already so much learning on flash over internet, however for demonstration purposes , flash animations can be divide into below categories :

1. **Frame by frame animation**
2. **Classic Tween (which was previously Motion Tween)**
3. **Shape Tween**
4. **Motion Tween (For 3d animation)**

In my project I have used almost all except the motion tween. In flash we can use our object oriented concepts to design and write object oriented programs. Flash is very powerful, however to keep it simple and sound I have used the procedural programming method in the ActionScript 3.0.

Let's take a look at my main actionscript, even though there are many of them in the file but this one is the main. ActionScript is similar to C or JavaScript, so it would be hard to understand the codes if someone has basic programming knowledge in any of those.

```
stop();


import flash.external.ExternalInterface;
import flash.events.MouseEvent;

function getBool(s:String):Boolean {
        if(s == ""){
                return false;
        }
        s = s.toLowerCase();
        if( s == "true" ||
           s == "y" ||
           s== "1" ||
           s == "yes"
           ){
                return true;
        }
        return false;
}
```

```
var CountTimerStart:Boolean = false;
//2592000000
var timer:Timer = new Timer(1000);
var timer2:Timer = new Timer(1000);

var secs:Number = 1;
var days:Number = 24 * 3600 *secs ;
var hrs:Number = 3600  * secs;
var mins:Number = 60 * secs;


browseBtnx.addEventListener(MouseEvent.CLICK, browseEvt);
browseClearBtn.addEventListener(MouseEvent.CLICK, browseClearBtn_Click);

fileOpenBtn.addEventListener(MouseEvent.CLICK, openFile);
openItemFolderBtn.addEventListener(MouseEvent.CLICK, openFolder);
openInEditorBtn.addEventListener(MouseEvent.CLICK, openFolderInEditor);

SearchBarClip.addEventListener(MouseEvent.MOUSE_MOVE, MouseMoveOnSearchbar_In);
SearchBarClip.addEventListener(MouseEvent.MOUSE_OUT, MouseMoveOnSearchbar_Out);

searchBtn.addEventListener(MouseEvent.MOUSE_MOVE, MouseMoveOnSearchbar_In);
searchBtn.addEventListener(MouseEvent.MOUSE_OUT, MouseMoveOnSearchbar_Out);

searchBtn.addEventListener(MouseEvent.CLICK, search_clicked);
SearchingInput.addEventListener(KeyboardEvent.KEY_DOWN, search_keyDown);

SearchingInput.addEventListener(MouseEvent.MOUSE_MOVE, MouseMoveOnSearchbar_In);
SearchingInput.addEventListener(MouseEvent.MOUSE_OUT, MouseMoveOnSearchbar_Out);

SearchingInput.addEventListener(FocusEvent.FOCUS_OUT, SearchBarFocusOut);
SearchingInput.addEventListener(FocusEvent.FOCUS_IN, SearchBarFocusIn);


searchBoolBtn.addEventListener(MouseEvent.MOUSE_MOVE, SearchBoolBtnIn);
searchBoolBtn.addEventListener(MouseEvent.MOUSE_OUT, SearchBoolBtnOut);
searchBoolBtn.addEventListener(MouseEvent.MOUSE_DOWN, SearchBoolBtnDown);
foundItemsList.addEventListener(MouseEvent.MOUSE_OVER, FoundList_hover);
foundItemsList.addEventListener(MouseEvent.MOUSE_OUT, FoundList_out);
//list double click event

foundItemsList.addEventListener(MouseEvent.DOUBLE_CLICK,FoundList_dbl_click);
foundItemsList.addEventListener(MouseEvent.CLICK, FoundList_click);


ExternalInterface.addCallback("setBarProgress", setBarProgress);
ExternalInterface.addCallback("setLocation", setLocation);
ExternalInterface.addCallback("addListItem", addListItem);
ExternalInterface.addCallback("clearListItem", clearListItem);
ExternalInterface.addCallback("getListItemAt", getListItemAt);

ExternalInterface.addCallback("setSetting", setSetting);
ExternalInterface.addCallback("setTimeText", setTimeText);
ExternalInterface.addCallback("setIndtProgressor", setIndtProgressor);
ExternalInterface.addCallback("startGlobalTimer", startGlobalTimer);
ExternalInterface.addCallback("stopGlobalTimer", stopGlobalTimer);
ExternalInterface.addCallback("hideIndtProgressor", hideIndtProgressor);
ExternalInterface.addCallback("getSearchText", getSearchText);

ExternalInterface.addCallback("readFoundFilesList", readFoundFilesList);



function getSearchText() :String {
        var str:String = Object(this).SearchingInput.text;

        if(str == "Search..."){
                str = "";
        }
    return str;
};

function startGlobalTimer() :void {
```

```
            // timer start
                //for timer http://adobe.ly/12ifWOB , http://adobe.ly/12ig4O7
                timer2 = new Timer(1000); //1 month is given
                timer2.addEventListener(TimerEvent.TIMER, timeCounterIncrement2);
                timer2.start();
        };

        function stopGlobalTimer() :void {
            // timer start
                //for timer http://adobe.ly/12ifWOB , http://adobe.ly/12ig4O7
                timer2.removeEventListener(TimerEvent.TIMER, timeCounterIncrement2);
                timer2.stop();
        };


        function setIndtProgressor(processing:String) :void {
            // timer start
                //for timer http://adobe.ly/12ifWOB , http://adobe.ly/12ig4O7
                timer = new Timer(1000); //1 month is given
                timer.addEventListener(TimerEvent.TIMER, timeCounterIncrement);
                timer.start();

                Object(root).indtProgressor.ProcessingText.text = "Processing : " + processing;
                Object(root).indtProgressor.visible = true;
                NoResults.visible = false;
                startGlobalTimer();
        };
        //setIndtProgressor("Folder");


        function hideIndtProgressor() :void {
            Object(root).indtProgressor.visible = false;
                Object(root).indtProgressor.ProcessingText.text = "Processing";
                if(foundItemsList.length == 0) {
                        NoResults.visible = true;
                }

                // stop timer
                //for timer http://adobe.ly/12ifWOB , http://adobe.ly/12ig4O7
                timer.removeEventListener(TimerEvent.TIMER, timeCounterIncrement);
                timer.stop();

        };

        hideIndtProgressor();
        //setIndtProgressor("folder");


        /*function onEnterFrameGlobal(event:Event) {

        }*/

        function timeCounterIncrement2(evt:TimerEvent):void {
                var spend = timer2.currentCount;
                var str:String;

                if(spend >= days){
                        // days
                        str = (spend/days).toFixed(1).toString() + " Days";
                } else if(spend >= hrs){
                        // hours
                        str = (spend/hrs).toFixed(1).toString() + " Hours";
                } else if(spend >= mins){
                        // mins
                        str = (spend/mins).toFixed(1).toString() + " Minutes";
                        //hideIndtProgressor();
                } else if(spend >= secs){
                        // mins
                        str = (spend/secs).toFixed(1).toString() + " Seconds";
                } else {
                        // mins
                        str = (spend).toFixed(1).toString() + " Miliseconds";
                }
                Object(this).timeTextBox.text = str;
```

```
	}

	function timeCounterIncrement(evt:TimerEvent):void {
		var spend = timer.currentCount;
		var str:String;

		if(spend >= days){
			// days
			str = (spend/days).toFixed(1).toString() + " Days";
		} else if(spend >= hrs){
			// hours
			str = (spend/hrs).toFixed(1).toString() + " Hours";
		} else if(spend >= mins){
			// mins
			str = (spend/mins).toFixed(1).toString() + " Minutes";
			//hideIndtProgressor();
		} else if(spend >= secs){
			// mins
			str = (spend/secs).toFixed(1).toString() + " Seconds";
		} else {
			// mins
			str = (spend).toFixed(1).toString() + " Miliseconds";
		}
		Object(this).indtProgressor.timeText.text = str;
	}


	function search_keyDown(e:KeyboardEvent):void {
		trace ("ase " + e.keyCode);
	    if( e.keyCode == 13) {
		var str = SearchingInput.text;
			if(str == "Search..."){
				str = "";
			} else {
				SearchingInput.text = str;
			}
			fscommand("search-clicked" , str)
			trace(str);
	    }
	}
	function search_clicked(e: MouseEvent) : void {
		var str = SearchingInput.text;
		if(str == "Search..."){
			str = "";
		}
		fscommand("search-clicked" , str)
		trace(str);
	}

	function setTimeText(str:String) :void {
	    timeTextBox.text = str;
	};
	setTimeText("");



	function clearListItem() :void {
	    foundItemsList.dataProvider.removeAll();
		NoResults.visible = true;
	};

	function addListItem(labelDataStr:String, locationx:String):void {
		NoResults.visible = false;

	    foundItemsList.addItem({label:labelDataStr,data: locationx});

	};

	function getListItemAt(index:int) :String {
		return (foundItemsList.getItemAt(index).data);
	};
```

```
        function setLocation(locationT:String ) :void {
            locationText.text = locationT;
        };

        fscommand("load-setting");


        function setSetting(s:String) :void {

                var sArray = s.split(";");

                var settingMC = settingMovieClip;

            settingMC.searchingDisableExtension.text = sArray[0];
                settingMC.searchingExtension.text = sArray[1];
                settingMC.searchingSizeType.selectedIndex = Number(sArray[2].toString());
                settingMC.startingSize.text = sArray[3];
                settingMC.endingSize.text = sArray[4];
                settingMC.lookForDates.selected = getBool(sArray[5]);
                settingMC.searchingDatesType.selectedIndex = Number(sArray[6].toString());
                settingMC.startingDate.text = sArray[7];
                settingMC.endingDate.text = sArray[8];
                settingMC.searchForContent.selected = getBool(sArray[9].toString()) ;
                settingMC.contentFoundMany.selected = getBool(sArray[10].toString()) ;
                settingMC.searchFromDatabase.selected = getBool(sArray[11].toString()) ;
                settingMC.searchFromCache.selected = getBool(sArray[12].toString()) ;
                settingMC.expireCache.text = sArray[13] ;
        };

        //setSetting("3pdf,jpg,gif,dat,mov,3gp,wmv,mp3,wav,ogg;txt,php,cs,vb,asp,aspx,cshtml,html,htm,
js;3;52;0;False;2;05-Jan-01 12:00:00 AM;01-Jan-01 12:00:00 plwdAM;false;true;false");


        ExternalInterface.addCallback("getSetting", getSetting);
        function getSetting() :Array {
            var settingMC = settingMovieClip;
                return [settingMC.searchingDisableExtension.text, //0
                                settingMC.searchingExtension.text, //1
                                settingMC.searchingSizeType.selectedIndex.toString(), //2
                                settingMC.startingSize.text, //3
                                settingMC.endingSize.text, //4
                                settingMC.lookForDates.selected.toString(), //5
                                settingMC.searchingDatesType.selectedIndex.toString(), //6
                                settingMC.startingDate.text, //7
                                settingMC.endingDate.text, //8
                                settingMC.searchForContent.selected.toString(), //9
                                settingMC.contentFoundMany.selected.toString(), //10
                                settingMC.searchFromDatabase.selected.toString(), //11
                                settingMC.searchFromCache.selected.toString(), //12
                                settingMC.expireCache.text]; //13

        };




        var searching = true;


        function SearchBarFocusIn(event:FocusEvent):void
        {
                trace(SearchingInput.text);
                if (SearchingInput.text == "Search..." || SearchingInput.text == "")
                {
                        SearchingInput.text = "";
                }

        }


        function SearchBarFocusOut(event:FocusEvent):void
        {
```

```
                if (SearchingInput.text == "Search..." || SearchingInput.text == "")
                {
                        SearchingInput.text = "Search...";
                }

        }


        function MouseMoveOnSearchbar_In(event:MouseEvent):void
        {
                /*var tfStyle:TextFormat = new TextFormat();
                tfStyle.color = "0x000000";
                SearchingInput.setTextFormat(tfStyle);
                SearchBarClip.alpha = 1.0;*/

        }


        function MouseMoveOnSearchbar_Out(event:MouseEvent):void
        {
                /*SearchBarClip.alpha = .75;
                var tfStyle:TextFormat = new TextFormat();
                tfStyle.color = "0xCCCCCC";
                SearchingInput.setTextFormat(tfStyle);*/

        }
        var settingArray = [];

        /*
        for (var i in array)
        {
                trace("row :" + array[i][0]);
                trace("col :" + array[i][1]);
        }
        */



        searchBoolBtn.buttonMode = true;
        /*
        if(searching){
                searchBoolBtn.gotoAndStop(3);
        } else {
                organizeBoolBtn.gotoAndStop(3);
        }


        function OrganizeBoolBtnIn(event:MouseEvent):void
        {
                if(searching){
                        organizeBoolBtn.gotoAndStop(2);
                }
        }

        function OrganizeBoolBtnOut(event:MouseEvent):void
        {
                if(searching){
                        organizeBoolBtn.gotoAndStop(1);
                }
        }

        function OrganizeBoolBtnDown(event:MouseEvent):void
        {
                if(searching){
                        organizeBoolBtn.gotoAndStop(3);
                        searching =false;
                        fscommand("organizingEnabled");

                        //reset searching
                        searchBoolBtn.gotoAndStop(1);
                }
        }
        */
```

```
function SearchBoolBtnIn(event:MouseEvent):void
{
	if(!searching){
		searchBoolBtn.gotoAndStop(2);
	}
}

function SearchBoolBtnOut(event:MouseEvent):void
{
	if(!searching){
		searchBoolBtn.gotoAndStop(1);
	}
}

function SearchBoolBtnDown(event:MouseEvent):void
{
	if(!searching){
		searchBoolBtn.gotoAndStop(3);
		fscommand("searchingEnabled");
		searching = true;

		//reset organizing
		//organizeBoolBtn.gotoAndStop(1);
	}
}



function browseEvt(event:MouseEvent):void
{
	fscommand("browse");
	trace("browse");
}


function browseClearBtn_Click(event:MouseEvent):void
{
	fscommand("browse-clear");
	trace("browse-clear");
}




// o to 100
function setBarProgress(value:Number)
{
	//progressbar.bar.scaleX = value;
	if ( value >= 96 || value <= 0 )
	{
		trace("complete or not start hide");
		processorBar.visible = false;
	} else {
		processorBar.visible = true;

	}


	var percent = Math.round(value);
	var percentText:String = percent + "%";
	/*Object(root).processorBar.gotoAndStop(percent);
	Object(root).processorBar.process.process.visible = true;
	Object(root).processorBar.process.gotoAndStop(percent);
	Object(root).processorBar.process_text_animation.gotoAndStop(percent);
	Object(root).processorBar.process_text_display.text = percentText;*/

	Object(this).processorBar.gotoAndStop(percent);
	Object(root).processorBar.process_text_display.text = percentText;
	Object(root).processorBar.process.gotoAndStop(percent);
}

setBarProgress(0);
```

```
function FoundList_hover(event:MouseEvent):void
{
	fscommand("foundlist-hover");
	trace("hover list");
}


function FoundList_out(event:MouseEvent):void
{
	fscommand("foundlist-out");
	trace("out list");
}

function FoundList_click(event:MouseEvent):void
{
	fscommand("foundlist-click",getListItemAt(foundItemsList.selectedIndex));
	trace("click list");
}

function FoundList_dbl_click(event:MouseEvent):void
{
	fscommand("foundlist-dblclick",getListItemAt(foundItemsList.selectedIndex));
	trace("dblclick list");
}




function openFile(event:MouseEvent):void
{
	if(foundItemsList.length > 0){
		trace("openFile :"  +  getListItemAt(foundItemsList.selectedIndex));
		fscommand("openFile", getListItemAt(foundItemsList.selectedIndex));
	}

}

function openFolder(event:MouseEvent):void
{
	if(foundItemsList.length > 0){
		trace("openFolder :"  +  getListItemAt(foundItemsList.selectedIndex));
		fscommand("openFolder", getListItemAt(foundItemsList.selectedIndex));
	}

}

function openFolderInEditor(event:MouseEvent):void
{
	if(foundItemsList.length > 0){
		trace("openFileEditor :"  +  getListItemAt(foundItemsList.selectedIndex));
		fscommand("openFileInEditor", getListItemAt(foundItemsList.selectedIndex));
	}

}




aboutBtnClick.addEventListener(MouseEvent.CLICK, fl_MouseClickHandler_9);

function fl_MouseClickHandler_9(event:MouseEvent):void
{
	this.aboutMe.gotoAndPlay(2);
 }
```

Communicate with Flash via .NET

In previous version of flash activeX control(OCX) for .NET,VB, JavaScript there was two methods named SetVariable(VariableName,setValue) and GetVariable(NameOfTheVarible) to communicate with flash , but in latest version it doesn't work. Consequently to communicate with flash I had to write external methods learned from http://forums.adobe.com/thread/84670 and in the flash activeX control (OCX) I have used the CallFunction method to call that method as below example.

Flash ActionScript 3.0 Coding

```
ExternalInterface.addCallback("getSearchText", getSearchText);

function getSearchText() :String {
	var str:String = Object(this).SearchingInput.text;

	if(str == "Search..."){
		str = "";
	}
    return str;
};
```

C#.NET 4.0 Coding

```
string req = "<invoke name='getSearchText' returnType='String'></invoke>";
string Value = FlashOCX.CallFunction(req);
```

Data Structures for the Application

To keep the application object oriented I have written few data structures. I have blog (http://bit.ly/Z6ACFK) where I basic concepts about the base data types that I have inherited to implement my custom data structures.

FileStructure & HashingOrganize are my basic data type for holding a single file.

```
[Serializable]
    public class FileStructure {

        public long ID { get; set; }
        public string FileName { get; set; }
        public string ExactLocation { get; set; }
        public string MD5 { get; set; }
        public long SizeBytes { get; set; }
        public string Extension { get; set; }
        public DateTime CreatedDate { get; set; }
        public DateTime ModifiedDate { get; set; }
        public DateTime LastAccessDate { get; set; }
        public string Folder { get; set; }

        /// <summary>
        /// File content if less than 3 MB
        /// </summary>
        public string Content { get; set; }

        /// <summary>
        /// Divided by space, comma , semicolon , or any other punctuation
        /// </summary>
        public string[] SearchingWords { get; set; }
        /// <summary>
        /// Divided by lines based on 'Enter'
        /// </summary>
        public string[] SearchingSentences { get; set; }

        /// <summary>
        /// Words Organized By Hashing System of 'a','b','c' etc...
        /// </summary>
        public List<HashingOrganize> HashingWords { get; set; }

        private byte isContentEmpty = 0;

        /// <summary>
        /// Found as a file
        /// </summary>
        public bool IsFound { get; set; }

        public string FoundPresentString { get; set; }

        public bool IsFolderFound { get; set; }


        /// <summary>
        /// Returns if content is empty and saves it in a field.
        /// </summary>
        public bool IsContentEmpty {
            get {
                if (isContentEmpty == 0) {
                    if (String.IsNullOrEmpty(Content) ||
```

```
String.IsNullOrWhiteSpace(Content)) {
                        isContentEmpty = 1;
                    } else {
                        isContentEmpty = 2;
                    }
                }
                return (isContentEmpty == 1);
            }
        }



    }



    public class HashingOrganize {
        public char FirstLetter { get; set; }
        public List<string> Words { get; set; }
    }
```

ListOfFiles is a type of List(generic object) and inherited from FileStructure.

ListOfFiles holds a list of files type of FileStructure.

```
[Serializable]
    public class ListOfFiles<FileStructure> : List<FileStructure> {



        /// <summary>
        /// Gets all given words in the search phrase.
        /// </summary>
        public List<string> AllWordsArray { get; private set; }

        /// <summary>
        /// Gets all given words with plural terms
        /// </summary>
        public List<string> AllWordsAndPluralsArray { get; private set; }

        private string[] SkipSequences = {"i","are","you","doing","were", "was","doing",
"to","this","these"};

        /// <summary>
        /// All Words - Skip words + replaced Plurals
        /// </summary>
        public List<string> PluralsWordsList { get; private set; }

        /// <summary>
        /// All Words - Skip words + replaced singulars
        /// </summary>
        public List<string> SingularWordsList { get; private set; }

        /// <summary>
        /// Gets exact search phrase in double quotations.
        /// </summary>
        public string ExactSearchText { get; private set; }

        /// <summary>
        /// Gets only the text combining text to search with operator 'AND"
        /// </summary>
        public string CombiningSearchText { get; private set; }
```

```
        /// <summary>
        /// Gets all words that should searched with 'OR' operator
        /// </summary>
        public string OrSearchText { get; private set; }
        /// <summary>
        /// Gtes all words that should searched with 'NOT' operator
        /// </summary>
        public string NotSearchText {
            get;
            private set;
        }

        /// <summary>
        /// List of Directories to search files for.
        /// </summary>
        public List<string> ListOfSearchingDirectories { get; set; }

        /// <summary>
        /// List of folders are already in the memory.
        /// </summary>
        public List<string> FolderAlreadyRead { get; set; }




        public bool IsWordsGenerated { get; set; }
        public bool IsSentenceGenerated { get; set; }
        public bool IsHashingGenerated { get; set; }

        /// <summary>
        /// Prerequisite: IsSortingEnabled must be enabled.
        /// Sort the files words based on a list of alphabet hashing table.
        /// Where 'a' => 'abcd' , 'awfef' etc.
        /// This algorithm is not good for programming script search.
        /// </summary>
        public bool IsHashingEnabled { get; set; }
        /// <summary>
        /// Sort the files words based on a to z.
        /// This algorithm is not good for programming script search.
        /// </summary>
        public bool IsSortingEnabled { get; set; }

        /// <summary>
        /// Look for exact phrases when something is double quoted.
        /// This algorithm is not good for programming script search.
        /// </summary>
        public bool IsNaturalSearchEnabled { get; set; }

        /// <summary>
        /// Save the information to the database.
        /// </summary>
        public bool IsSaveToDatabase { get; set; }

        /// <summary>
        /// Try to search from database if it is 7 day old.
        /// </summary>
        public bool IsSearchFromDatabaseEnabled { get; set; }

        /// <summary>
        /// Database info how many days old accepts.
        /// </summary>
        [DefaultValue(7)]
        public int DaysOld { get; set; }

        /// <summary>
```

```
        /// Last time of execution of any methods
        /// (Generated Sentence,Words or Hashing);
        /// </summary>
        public DateTime GeneratedTime { get; set; }


        /// <summary>
        /// All joining text 'i','want', 'to' from "I want to"
        /// </summary>
        private List<string> searchTextCombiningList = new List<string>();
        private List<string> searchTextOrList = new List<string>();
        private List<string> searchTextNotList = new List<string>();


        private string searchingText = "";

        private bool isSearchTextEmpty() {
            return String.IsNullOrEmpty(searchingText) ||
String.IsNullOrWhiteSpace(searchingText);
        }

         /// <summary>
        /// Get the searching text in an organize way
        /// hello, world , "I" am here because => am because hello here I world
        /// Container
        /// </summary>
        private string _searchingTextAsOrganizeString;
        /// <summary>
        /// Get the searching text in an organize way
        /// hello, world , "I" am here because => am because hello here I world
        /// </summary>
        public string GetSearchingTextAsOrganizeString { get { return
_searchingTextAsOrganizeString; } }

        /// <summary>
        /// Gets the whole search text and
        /// sets the whole searching text.
        /// </summary>
        public string SearchString {
            get {
                return searchingText;
            }



            set {
                SearchEngineEntities2 db = new SearchEngineEntities2();
                searchingText = value.ToLower();
                char[] splitBasedOnWords = "\"'\' ;:|\\><!@#$%^&*()~!.?*".ToCharArray();

                //remove string tag
                searchingText = searchingText.Replace("<string>", "");
                searchingText = searchingText.Replace("</string>", "");

                AllWordsArray = searchingText.Split(splitBasedOnWords).Where(m=> m !=
"").ToList();
                AllWordsArray = AllWordsArray.OrderBy(n => n).ToList();

                _searchingTextAsOrganizeString = String.Join(",", AllWordsArray);

                AllWordsAndPluralsArray = new List<string>();
                AllWordsAndPluralsArray.Capacity = 50;
                //remove repeated words
                AllWordsArray = AllWordsArray.Distinct().ToList();
                var allwordsWithoutSkip = AllWordsArray.Except(SkipSequences).ToList();
                PluralsWordsList = new List<string>();
```

```
                SingularWordsList = new List<string>();
                foreach (var word in allwordsWithoutSkip)
                    {
                     PluralDictionary plural;

                     plural = db.PluralDictionaries.FirstOrDefault(n => n.Single == word);

                     PluralDictionary singular = null;
                     if (plural == null) {
                        singular = db.PluralDictionaries.FirstOrDefault(n => n.Plural ==
word);
                     }


                     if (plural != null) {
                         SingularWordsList.Add(plural.Plural);
                     } else if (singular != null) {
                         PluralsWordsList.Add(singular.Single);
                     } else {
                         PluralsWordsList.Add(word);
                         SingularWordsList.Add(word);
                     }
                    }



        private string getExactSearchPhrase(ref string s, int startIndex = 0) {
            string extract = "";
            if (!isSearchTextEmpty()) {

                var doubleQuoteIndex = s.IndexOf('\"', startIndex);
                if (doubleQuoteIndex > -1) {
                    //double quote exist at least one time
                    var doubleQuoteIndexEndingPoint = s.IndexOf('\"', doubleQuoteIndex +
1);
                    if (doubleQuoteIndexEndingPoint > 0) {
                        // we have 2 double quotes
                        // now extract the double quoted text
                        int len = doubleQuoteIndexEndingPoint - doubleQuoteIndex;
                        extract = s.Substring(doubleQuoteIndex + 1, len - 1);
                        ConsoleLog("Full Text : ",s);
                        ConsoleLog("Extraction On Double Quote : ", extract);
                        // now remove exact from the text
                        s = s.Remove(doubleQuoteIndex, doubleQuoteIndexEndingPoint + 2);
                    }
                }
            }
            return extract;
        }




        void ConsoleLog(string CaptionFrom, string value) {
            Console.Out.WriteLine(CaptionFrom + " : " + value);
        }

    }
```

Alim Ul Karim's Algorithms

My algorithms are easy to read and I have used comments to make it simple. Even though I had written those algorithms, however I didn't get the time to implement my algorithms in action.

```
#region Searching Text Organize Algorithms

        private Folder folder = new Folder();
        /// <summary>
        /// Organize(generate searching sentences BigO : Files x Lines)
        /// and generate searching words , sentences and hashing
        /// words if necessary.
        /// </summary>
        /// <param name="Files">a list of file structure.</param>
        /// <param name="wordsSortedDictionary">Should we generate the SearchingWords dictionary in
sorted order or not.
        /// BigO(Files x Words ln Words) (</param>
        /// <param name="hashingNeeded">Should we generate the Hashing algorithm for each words by the
first letter.
        /// Distinct : BigO(File X Words ^ 2) ; Without Distinct : BigO(File X Words)</param>
        /// <param name="hashingDistinct">If we do hashing should we only keep distinct.
        /// Distinct : BigO(File X Words ^ 2) ; Without Distinct : BigO(File X Words)</param>
        public void OrganizeMethods(ListOfFiles<FileStructure> Files, bool wordsSortedDictionary =
false, bool hashingNeeded = false, bool hashingDistinct = false) {
            // BigO : Files x Lines
            GenerateSearchingSentence(Files);
            if (wordsSortedDictionary) {
                //BigO(Files x Words ln Words)
                GenerateSearchingWords(Files);
            }

            if (hashingNeeded) {
                //Distinct : BigO(File X Words ^ 2) ; Without Distinct : BigO(File X Words)
                GenerateHashingWords(Files, hashingDistinct);
            }
        }

        public void SearchAndAdd(ListOfFiles<FileStructure> fileList,
            string[] InludingExtensionList,
            string[] ExcludingExtensionList,
            DateTime startDate,
            DateTime endDate,
            int dateType = -1,
            int FileSizeType = -1,
            long fileSizeStart = -1,
            long fileSizeEnd = -1,
            bool lookForDate = false,
            AxShockwaveFlashObjects.AxShockwaveFlash flx = null) {

            //if (fileList == null || fileList.Count == 0) {
            //    return;
            //}

            if (FileSizeType == FileSizeTypeStructure.KB) {
                fileSizeStart = fileSizeStart * (int)FileSizeTypeStructure.KB_1;
                fileSizeEnd = fileSizeEnd * (int)FileSizeTypeStructure.KB_1;
            } else if (FileSizeType == FileSizeTypeStructure.MB) {
                fileSizeStart = fileSizeStart * (int)FileSizeTypeStructure.MB_1;
                fileSizeEnd = fileSizeEnd * (int)FileSizeTypeStructure.MB_1;
            } else if (FileSizeType == FileSizeTypeStructure.GB) {
                fileSizeStart = fileSizeStart * (int)FileSizeTypeStructure.GB_1;
                fileSizeEnd = fileSizeEnd * (int)FileSizeTypeStructure.GB_1;
            } else {
                fileSizeStart = -1;
                fileSizeEnd = -1;
            }
```

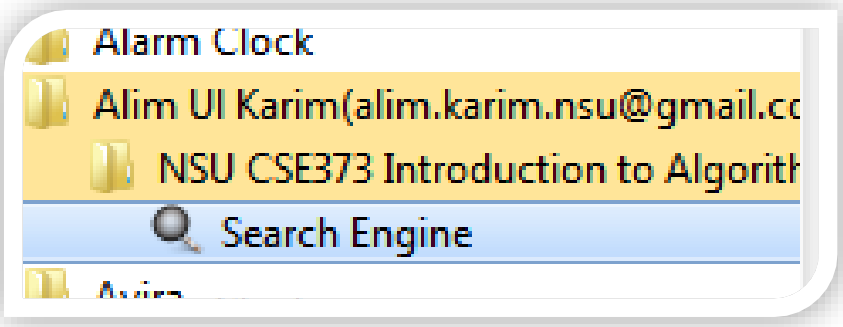
```
            foreach (var filestr in fileList) {
                new Thread(() => {
                    var isExtensionExist = false;
                    if (InludingExtensionList != null) {
                        isExtensionExist = InludingExtensionList.Any(n => n == filestr.Extension);
                    }
                    var isDisableExtensionExist = false;
                    if (ExcludingExtensionList != null) {
                        isDisableExtensionExist = ExcludingExtensionList.Any(n => n ==
filestr.Extension);
                    }
                    bool addFile = false;
                    bool isExtensionRight = false;
                    bool isSizeRight = false;
                    bool isDateRight = false;
                    if (isExtensionExist && !isDisableExtensionExist) {
                        isExtensionRight = true;
                    }


                    if (FileSizeType != FileSizeTypeStructure.NoLimit) {
                        if (filestr.SizeBytes >= fileSizeStart && filestr.SizeBytes <= fileSizeEnd) {
                            isSizeRight = true; //only if size meets the condition then add the file.
                        }
                    } else {
                        isSizeRight = true; //no size limit
                    }

                    if (lookForDate) {
                        if (dateType == DateTypeStructure.Modified_Date) {
                            DateTime dt = filestr.ModifiedDate.Date;
                            if (dt != null && (dt >= startDate.Date && dt <= endDate.Date)) {
                                isDateRight = true;
                            }
                        } else if (dateType == DateTypeStructure.Created_Date) {
                            DateTime dt = filestr.CreatedDate.Date;
                            if (dt != null && (dt >= startDate.Date && dt <= endDate.Date)) {
                                isDateRight = true;
                            }
                        } else if (dateType == DateTypeStructure.Access_Date) {
                            DateTime dt = filestr.LastAccessDate.Date;
                            if (dt != null && (dt >= startDate.Date && dt <= endDate.Date)) {
                                isDateRight = true;
                            }
                        }
                    } else {
                        isDateRight = true;
                    }

                    if (isSizeRight && isDateRight && isExtensionRight) {
                        addFile = true;
                    }

                    if (addFile) {
                        //if found by the size
                        //
                        if (folder.IsMatchInText(filestr.FileName, fileList)) {
                            FlashPropertise.addListItemStatic(filestr.FileName, filestr.ExactLocation,
flx);
                        } else if (!filestr.IsContentEmpty && folder.IsMatchInText(filestr.Content,
fileList)) {
                            FlashPropertise.addListItemStatic(filestr.FileName, filestr.ExactLocation,
flx);
                        }

                    }
                }).Start();
            }
        }
```

```
        /// <summary>
        /// Generate Sentences based on enter. O(Files x Lines)
        /// </summary>
        /// <param name="Files">a list of file structure.</param>
        public void GenerateSearchingSentence(ListOfFiles<FileStructure> Files) {
            if (!Files.IsSentenceGenerated && Files.Count > 0) {
                Files.AsParallel().ForAll(file => {
                    if (!file.IsContentEmpty) {
                        file.SearchingSentences =
file.Content.Split(char.ConvertFromUtf32(13).ToCharArray());
                    }
                });
                Files.IsSentenceGenerated = true;
                Files.GeneratedTime = DateTime.Now;
            }
        }

        /// <summary>
        /// Generate words based on any delimiter (,.<>@#$!%^&*()-=_:;'\"[]{}~! etc...)
        /// Return the words as a sorted list (abc,bcd,cde ...)
        /// BigO(Files x Words ln Words)
        /// </summary>
        /// <param name="Files">a list of file structure.</param>
        public void GenerateSearchingWords(ListOfFiles<FileStructure> Files) {
            if (!Files.IsWordsGenerated && Files.Count > 0) {
                char[] split = ",.<>@#$!%^&*()-=_:;'\"[]{}~! \n".ToCharArray();

                Files.AsParallel().ForAll(file => {
                    if (!file.IsContentEmpty) {
                        // O(Words + Words ln Words + Words) = O(Words ln Words)
                        file.SearchingWords = file.Content.Split(split).OrderBy(m => m).ToArray();
                    }
                });
                Files.IsWordsGenerated = true;
                Files.GeneratedTime = DateTime.Now;

            }
        }

        /// <summary>
        /// Prerequisite: GenerateSearchingWords() must execute before it.
        /// Generate hashing words based on 'a' => all words starting from a.
        /// 'b' => all words starting from b etc...
        /// Starting with number words will be distributed in the '#' char.
        /// Distinct : O(File X Words ^ 2) ; Without Distinct : O(File X Words)
        /// </summary>
        /// <param name="Files">a list of file structure.</param>
        public void GenerateHashingWords(ListOfFiles<FileStructure> Files, bool OnlyKeepDistinct =
true) {
            if (!Files.IsHashingGenerated && Files.IsWordsGenerated && Files.Count > 0) {
                // O(File X Words ^ 2 X 27) => O(File.Words ^ 2)
                Files.AsParallel().ForAll(file => {
                    if (!file.IsContentEmpty) {
                        file.HashingWords = new List<HashingOrganize>();
                        foreach (var word in file.SearchingWords) {
                            char firstLetter = word.ToLower()[0];
                            if (firstLetter >= '0' && firstLetter <= '9') {
                                // if it is a number then
                                // keep inside #
                                firstLetter = '#';
                            }
                            var hashingObj = file.HashingWords.Where(w => w.FirstLetter ==
firstLetter).FirstOrDefault();
                            if (hashingObj != null) {
                                if (hashingObj.Words == null) {
                                    // new list
                                    hashingObj.Words = new List<string>();
                                }

                                if (OnlyKeepDistinct) {

                                    // already char exist in the dictionary
                                    // so add word only, if the word is not already exist.
```

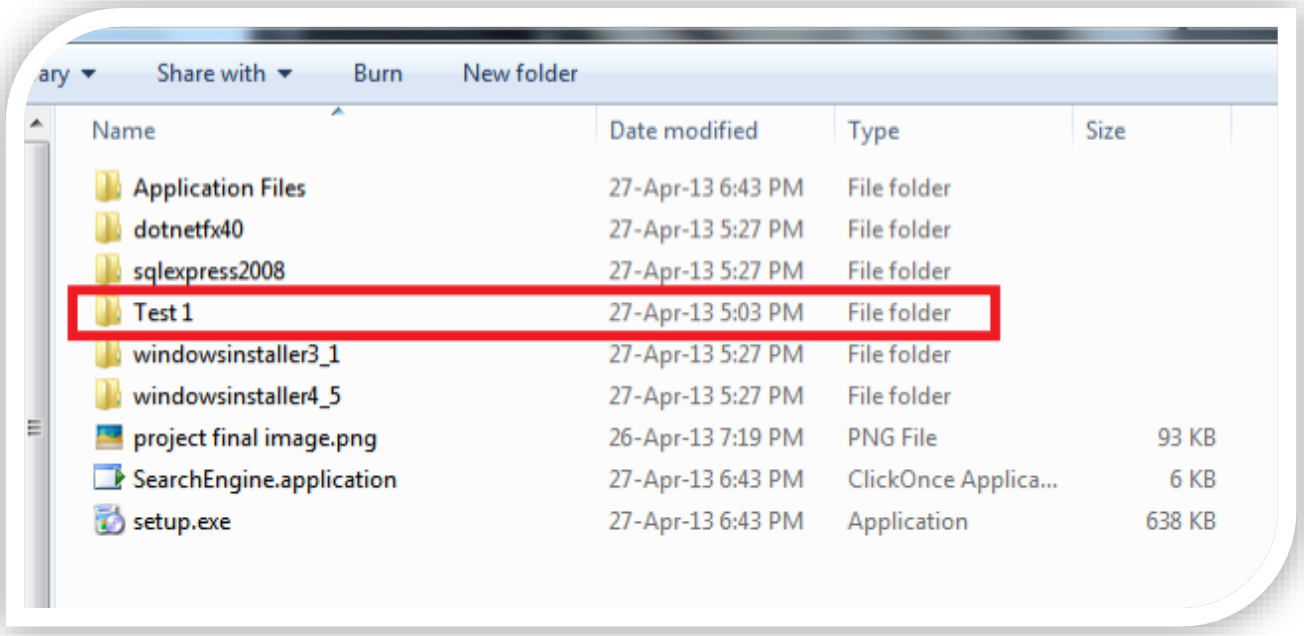
```
                                //if word is not already exist in the dictionary.
                                var wordExist = hashingObj.Words.Exists(w => w == word);
                                if (!wordExist) {
                                    // not already exist .
                                    // add to the dictionary.
                                    hashingObj.Words.Add(word);
                                }
                            } else {
                                // do not check the repeated words.
                                hashingObj.Words.Add(word);
                            }
                        } else {
                            // if the letter not already present then
                            HashingOrganize hO = new HashingOrganize() {
                                FirstLetter = firstLetter,
                                Words = new List<string>()
                            };
                            // adding the word to the dictionary directly.
                            hO.Words.Add(word);
                            // now adding the object to the final list of HashingOrganize
                            file.HashingWords.Add(hO);
                        }
                    }

                }
            });
            Files.IsHashingGenerated = true;
            Files.GeneratedTime = DateTime.Now;
        }
    }
    #endregion

    #region Searching

    #endregion
```

Finalization

I haven't put everything in a sequential order, however I tried to explain the key points in my project. Source code of the project and the final project is presented in a dvd. For example I have attached a folder named **'Test 1'** of many files with text from Wikipedia. After installation, there will be a folder in the start bar with my name "Alim Ul karim (alim.karim.nsu@gmail.com)" inside that folder my application will be there.



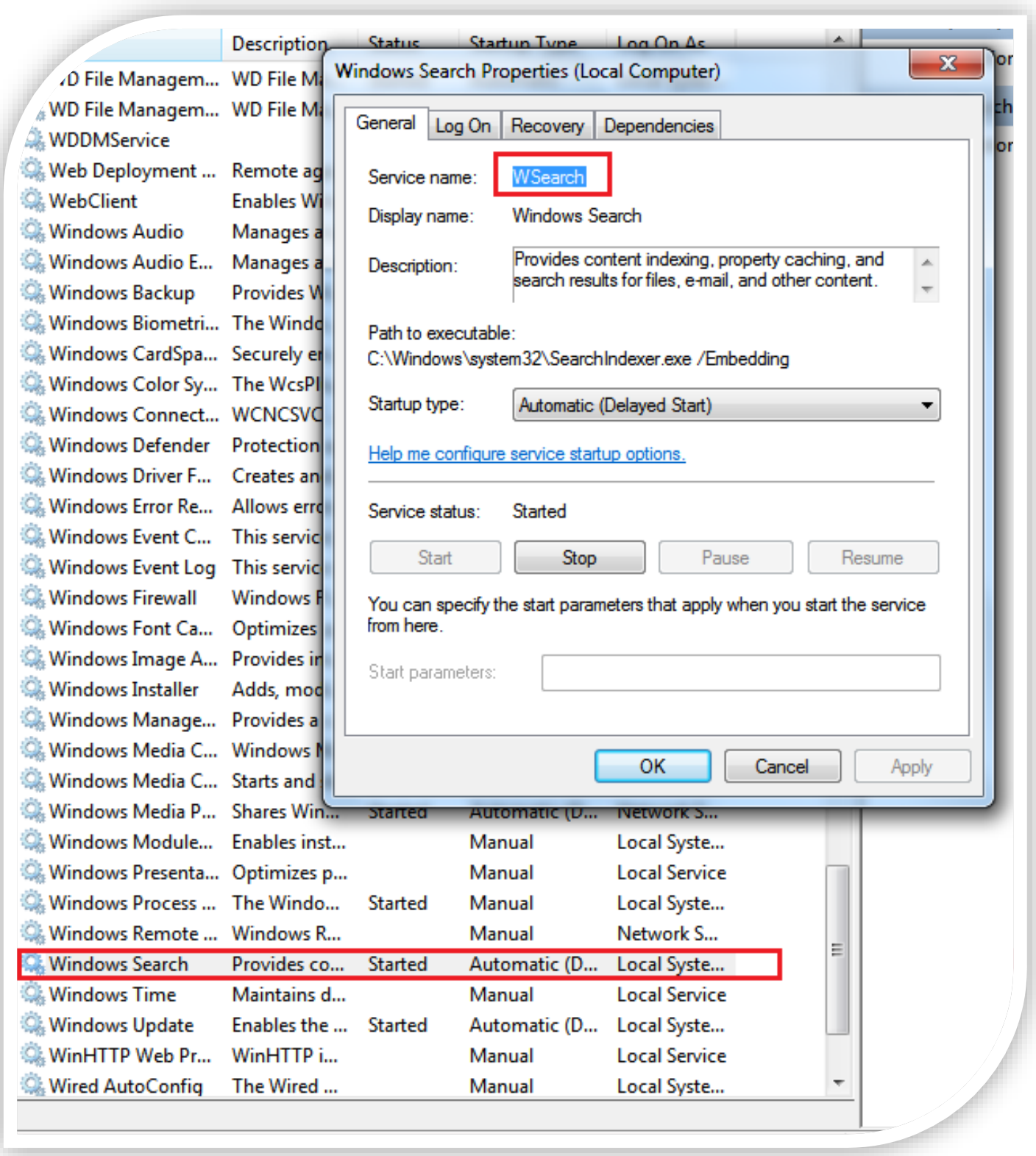
Startmenu Picture



Installation Folder("Test 1" named Example folder)

Limitations

It was that fast as I claimed it would be. In my hypothesis I expect it to outrun windows search but I couldn't windows search is much faster than mine. However, to solve the puzzle "**why?"** windows search is faster than mine I studied for a while and learned that windows have a searching service which they start as soon as we start our windows and then it starts indexing files to their database or some indexing system in the background. Consequently, if I use a process like that I might outrun windows search but without doing so I can't claim anything. In addition, I haven't implemented my searching algorithm and there are some small bugs as well (i.e. for file names it doesn't look from the pluralize dictionary and if we use a search then in 30mins cached results are not always perfect) that I couldn't fix because of the time limit.



Windows Search Service Image

Conclusion

Finally, I have reached to end point. From all my hypothesis only one become successful which is 'advance' and others might be successful if I had more time. However, without real product I can't confirm anything, these are all hunches. Even though theoretically my algorithm should be faster than O(n) but I haven't implemented my algorithm yet so I can't confirm anything. It was a nice project, I have learned a lot and couldn't put all into this paper.

References

1. http://auk-port.blogspot.com/2012/08/difference-among-ienumerable-iqueryable_5620.html
2. http://forums.adobe.com/thread/84670
3. Stackoverflow
4. Google