TheFolderSpy SDK for servers

Contents:

P.S. Are you in a hurry? Click here to see a fully functional class (15 line code) made using TFS.

1. Introduction

a. What is TheFolderSpy?

TheFolderSpy is a program that monitors and alerts the user if files/folders are created, changed, added or deleted within a folder.

TFS is a very mature and robust program and is being used for more than 4 years.

TheFolderSpy SDK is the 11 KB implementation of the same functionality in a dll so that other programmers may also benefit from it.

This is a short documentation of details on how to use the SDK and the finer points of the implementation.

b. Ok That's Fine, But Why Should I Use It?

There are several reasons. Normally the detection mechanism is single threaded and not suitable for heavy post-processing of the changes.

Consider the case of a simple backup program. The logic says that, if a file X changes or a new file is created, update/create its backup. Now this involves opening the file X, reading the content and writing them in the backup location. This may take some time if the file is of reasonable size and if this happens repeatedly and quickly (say *1000* or *10,000 times a second*), the processes [e.g., user uploading or editing files] that make these changes in files may be stalled because copying the content takes time and the whole process is sequential and operates in a single thread so calls are blocking. In these cases, some notifications are missed also which you may even not notice.

Another common scenario is the duplicate reception of changes in a file. Windows writes data chunk by chunk and all of these chunks are picked by the detection mechanisms, thus giving rise to duplicate notifications or alerts.

TheFolderSpy runs these post-processing operations in multiple threads and uses a very robust algorithm to supress the duplicate notifications [One-Notification-Per-Detection [ONPD]] and never misses any notification. Never.

We did a stress test to prove the mettle of TFS.
App using this SDK succeeded to detect:

1. **10,000 files created and logged in about 8.1 seconds. That's approx. 2 files per millisecond.**
2. **100,000 files created and logged in about 142 seconds. That's approx. 0.7 files per millisecond.**

The SDK allows you to use these features in your application via a roughly 10KB sized dll file. You just need **4 lines of code** to use and feel the power, simplicity and the robustness of TFS SDK.

```
callBack = new TfsDemo.CallBack(callBackMethod);
tfs = TfsDemo.getInstance(callBack, txtPathToMonitor.Text, false, "*.*", true,
3);
tfs.addNotification(TfsDemo.DetectionType.Created);
tfs.startMonitoring();
```

2. Getting Started

To get started with development using TFS SDK, you just need the tfs.dll (or TfsSdkDemo.dll). You will find all the needed API reference here in this document. From here on we will assuming that we have the TfsSdkDemo.dll file.

Just reference the dll and you can access all the power of TFS.

3. API Reference

You may want to have a look at a sample, working program. Only 15 lines of code.

1. Emailer Class

The Emailer class is a very simple emailing class, abstracting all the complexities involved.
You get only 2 public methods: one is the *constructor* itself and the other one is overloaded *send*.

Members:

Constructor	`Emailer(String from, String pwd, int port, String host, bool ssl)`
send	`void send(String subject, String body, String[] recipients)`
send	`void send(String subject, String body, String[] recipients, String[] attachments)`

Description:

1. `Emailer(String from, String pwd, int port, String host, bool ssl):`

   Parameters:

   a. `String from`: The email address of the sender. E.g., thefolderspy@gmail.com
   b. `String pwd: The password of the above entered email address.`
   c. `int port: The port number of the SMTP server.`
   d. `String host: The host name of the SMTP server.`
   e. `bool ssl: true, if the SMTP server requires SSL.`

   Description:
   Initializes the Emailer class.

2. void send(String subject, String body, String[] recipients):

   Parameters:

   a. String subject: The subject of the email.
   b. String body: Body of the email.
   c. String[] recipients: String Array containing email addresses of the recipients.

   Description:
   Sends an email with specified *body* and *subject* to the *recipients*.

3. void send(String subject, String body, String[] recipients, String[] attachments):

   Parameters:

   a) String subject: The subject of the email.
   b) String body: Body of the email.
   c) String[] recipients: String Array containing email addresses of the recipients.
   d) String[] attachments: String Array containing paths of the attachments.

   Description:
   Sends an email with attachments to the *recipients*.

2. TfsDemo Class

This is the heart and soul of the SDK. It has 5 methods, no constructor, 1 struct, 2 enum, 1 property and 1 delegate.
As this is just a demo class, you can get a single instance of this class.

- **Structures, Enums, Properties:**

The types of detections TFS can make.

```
public enum DetectionType
{
    Created,
    Changed,
    Deleted,
    Renamed
}
```

TFS provides this information:

```
public struct tfsVars
{
    public DateTime dateTime;
    public String fileName;
    public String Path;
    public String userName;
    public DetectionType detectionType;
    public String oldFileName;
    public String oldPath;
}
// oldFileName and oldPath are only available in case of rename detection.
```

MonitoringTypes defines the properties to be monitored for a change to be notified.
These can be combined using the bitwise OR operator "|".

```
public enum MoniteringTypes
{
    fileName = 1,
    directoryName = 2,
    attributes = 4,
    size = 8,
    lastWrite = 16,
    lastAccess = 32,
    creationTime = 64,
    security = 256,
}
```

- **Delegate:**

```
        delegate void CallBack(TfsDemo.tfsVars param);
```

You register a method as a *CallBack* for each of the detections that TFS detects.

Whenever an event occurs such as a file is created, the registered method is fired by TFS.

The whole procedure is multithreaded and is very effectively managed.
If you need to update GUI, you need to call the update operations using a delegate or MethodInvoker. Any of them will do the job. This has been implemented in the sample.

A possible implementation can be like this:

```
private void callBackMethod(TfsDemo.tfsVars vars)
        {
            this.Invoke((MethodInvoker)delegate
            {
                // update the GUI.
                lbl.Text = ++counter + " items.";
            });
        }
```

- **Methods:**
  1. *getInstance()*:

```
TfsDemo getInstance(
       CallBack callback,
       string pathToMonitor,
       bool monitorSubDirectories = false,
       string detectionFilter = "*.*",
       bool heavyDuty = false,
       int numberOfThreads = 2
)
```

Parameters:

a. CallBack callback: The callback delegate.
b. string pathToMonitor: The path to be monitored for changes.
c. bool monitorSubDirectories: `true`, if sub-directories should be monitored. [As this is a demo class, you won't be able to monitor sub-directories.]
d. string detectionFilter: String containing wildcards which defines which files are to be monitored.
e. bool heavyDuty: `true`, if the application has to monitor folder which have a lot of activity. [Varies from system to system but changes to more than 500 files/second may require this mode.]
f. int numberOfThreads: The number of threads which handle the detections and call the callback method.

Description:
Returns a TfsDemo object. As this is a demo class only one instance can be created and sub-directories cannot be monitored.

Exceptions:
`DirectoryNotFoundException`: If the directory to be monitored is not present.

2. *addNotification()*:

```
bool addNotification(DetectionType d)
```

Parameters:

a. DetectionType d: The detection which you want to register for.

Description:
Registers for one of the [Created, Changed, Deleted, Renamed] notifications.
Returns `true`, if successful.

3. *removeNotification()*:

```
bool removeNotification(DetectionType d)
```

Parameters:

a. DetectionType d: The detection which you want to unregister for.

Description:
Unregisters for one of the [Created, Changed, Deleted, Renamed] notifications.
Returns `true`, if successful.

4. *startMonitoring()*:

```
void startMonitoring()
```

Starts monitoring for changes.

5. *stopMonitoring()*:

```
void stopMonitoring()
```

Stops monitoring for changes.

4. The First TFS SDK Program

Create a new .Net project in any version of Visual Studio or your preferred editor and add a reference to the above mentioned dll (from References -> Add Reference...)

All the codes in this file are in C#, which can be easily translated into other languages as well.

If you have created a Console Application, go straight ahead into the **PROGRAM.CS** file and start coding or if you have a Windows Form Application, you may want to design a minimal GUI before entering into code writing part.

The steps in a single line: **Initialize class, setup callback method, register for notifications & start monitoring**.

First, you need to add a reference of the TfsSdkDemo.dll file.

The namespace is TfsSdkDemo, so you can add that at the top:

```
using TfsSdkDemo;
```

Next declare the objects of TfsDemo and TfsDemo.CallBack. Preferably declare them as global variables.

```
TfsDemo tfs;
TfsDemo.CallBack callBack;
```

Next create a method that can be fed into `callBack`. An example:

```
private void callBackMethod(TfsDemo.tfsVars vars)
        {
            // your business logic here.
            this.Invoke((MethodInvoker)delegate
            {
                // update the GUI here.
                lbl.Text = ++counter + " items.";
            });
        }
```

Now, inside the Main or btnStart's Click event add the code to initialize objects.

```
private void btnStart_Click(object sender, EventArgs e)
{
    callBack = new TfsDemo.CallBack(callBackMethod
    tfs = TfsDemo.getInstance(callBack, txtPath.Text, false, "*.*", true, 3);
    tfs.addNotification(TfsDemo.DetectionType.Created);
    tfs.startMonitoring();
}
```

That's it. TFS has been started and is ready to notify about the registered changes.
To stop, you can call `stopMonitoring()` anytime.
Below is a complete Console Application. You can also download a Windows Form Application Sample.

```
using TfsSdkDemo;
class Program
{
    private static void callBackMethod(TfsDemo.tfsVars vars)
    {
        if (vars.detectionType == TfsDemo.DetectionType.Renamed)
            Console.WriteLine(vars.dateTime + " - " + vars.oldFileName + " -
                              was renamed to: " + vars.fileName);
        else
            Console.WriteLine(vars.dateTime + " - " + vars.fileName + " was "
                              + vars.detectionType.ToString());
    }

    static void Main(string[] args)
    {
        TfsDemo.CallBack callBack = new TfsDemo.CallBack(callBackMethod);
        TfsDemo tfs = TfsDemo.getInstance(callBack, "U:\veryImpFolder", false,
                                          "*.*", true, 2);
        tfs.addNotification(TfsDemo.DetectionType.Created);
        tfs.addNotification(TfsDemo.DetectionType.Renamed);
        tfs.startMonitoring();
        Console.WriteLine("Started monitoring..");

        Console.ReadLine(); // waits until Return/Enter key is pressed to give
                            // time to do some copy-pasting of files to check
                            // the working of TFS.

        tfs.stopMonitoring();
        Console.WriteLine("Exiting..");
    }
}
```

The code is pretty straight-forward, but you may want to see the API reference.

5. Possible Uses and Scenarios

TFS has potentially limitless uses. Any task that is triggered by a change in a file or a folder can be handled by TFS. The fact that any .Net language namely **C#, VB, ASP & C++/CLI** can use TFS only increases the usefulness.

I will try to list some uses:

1. Report/ Log generation.
2. Auditing.
3. Email based notification of changes. You can send *customized* emails to different persons based on the directory or file which was changed.
4. Email the file that was changed / created.
5. Backup triggering.
6. Windows Services can be created.
7. Alarm/Email when an important file/folder is changed.
8. Can monitor Network Drives too!
9. Can handle heavy post-processing, so changed files could be analysed and actions based on specific changes can be taken.

6. Summary & Limitations

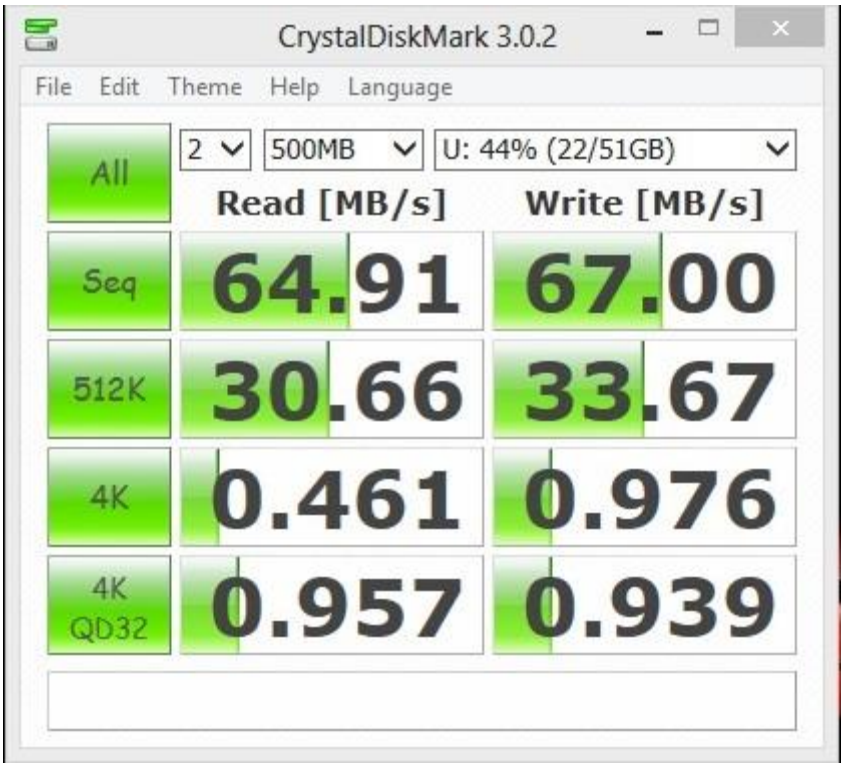
TFS SDK is very robust, efficient & reliable and is matured by 4 years of user base consisting of both developers & end-users and their reviews.

Code writing is simple, short and sweet with just 5 lines to get started.

This demo class is in no way limited in terms of performance.
There are only two limitations, you can't monitor sub-directories and you can create only one instance of the class.

Details of Stress Test:

1. Specifications of computer:
   a. Intel Core i3, first gen CPU.
   b. 6 GB RAM.
   c. 5400 RPM Barracuda HDD, 500 GB. [With millions of files]
   d. OS: Windows 8.
   [Below is the benchmark of HDD by CrystalDiskMark 3.0.2]



HDD Benchmark Screenshot 1

2. The code:
   This code was called for each of the detections:[1]

```
private void callBackFileBackUp(TfsDemo.tfsVars vars)
{
    using (StreamWriter sw = new StreamWriter(path + "\\" + vars.fileName, false))
    {
        using (StreamReader sr = new StreamReader(tfsDir + "\\" + vars.fileName))
        {
            sw.WriteLine(sr.ReadToEnd());
        }
    }
    this.Invoke((MethodInvoker)delegate
    {
        lbl.Text = ++counter + " items.";
    });
}
```

For every detected file, there was another file created with the same content, i.e., the newly created file was backed up. The files were created with 1KB of random data.

The backing up added about 2 seconds of time for 10,000 files, with the complete operation taking a little over 10 seconds.

7. FAQ / CONTACT US

You can contact us at: venussoftcorporation@gmail.com
Visit us at: www.venuslogiclabs.com

1. What to do you exactly mean by multithreading? I don't really understand the flow.

Let me again describe it. TFS monitors a folder for changes. You register for changes by calling *addNotification*() method. Whenever a change is detected, the callback method is triggered and runs in a different thread than your GUI. This distributes the load. All the complexities of synchronization is internally managed by TFS.

[1] This code assumed that the created files were about 1KB in length. For larger files, it is a different story. As the file size is quite small, we can almost always be sure that file copy process completed before TFS post-processes the copied file. If the file is larger, then TFS may report the detection before copying was completed fully. So in that case, you would need to know when exactly the file copying was completed and then do the post-processing. It's a bit complex, but can be done.

2. I really like this! How can I get the full version?

We are glad you found this useful. You can contact us here to get a full version.

3. What are the licenses available?

License	Description	Price(USD)
In House Single Developer License	Allows one developer to create an unlimited number of derived works using the product. The derived works can then be deployed to one physical location within your organization.	100
OEM Single Developer License	Allows one developer to create an unlimited number of derived works using the product. The derived works can be deployed to an unlimited number of sites within or outside of your organization.	500

We also provide source code and individual licenses. Discounts are available for multiple purchases. All licenses are for lifetime.
Please fire us an email here to know more about the licenses.

4. I don't have resources/developers that can use TFS SDK and create software for my needs. Can you create one for us?

Sure. Just mail us your requirements here.

5. What about support?

1 year support is provided free of cost.
We provide email and Skype based support.

6. My question isn't here, what should I do?

No need to worry. You just email us your question. We are happy to help.