Grinder

Developer's Guide

Introduction

Grinder's intended purpose is to make processing files as easy as possible. Included with Grinder are a number of basic actions which should handle the basic. Most processing though goes beyond the basic as such Grinder provides an object model and interface to make it easier to create custom actions to deal with situations the included actions don't.

Grinder Workflow Actions

FileCopy

Copies a file from the SourceFilePath to the DestinationFilePath.

FileDelete

Deletes the specified file

FileMove

Moves the specified file from the SourceFilePath to the DestinationPath

ExecuteApplication

Executes the specified program passing it the specified arguments

UnzipFile

Extracts the specified file from the archive.

ZipFile

Places the file in the specified archive.

Grinder Object Model

Grinder's object model is in the FileProcessorUtilities assembly which should be referenced by custom actions that are going to work with the Grinder service.

WorkItem

The WorkItem class is found in the GuerillaProgrammer.FileProcessor namespace. When Grinder is processing a file it first copies the file from its original location in to a working directory. This is done to avoid processing the same file multiple times. When the file is moved its name is changed by adding a guid to the end of the filename. After

the file has been moved Grinder creates and queues a workitem that will be passed to the workflow and used by the workflow actions during processing. The WorkItem has the following properties:

Filename	The name of the file being processed. This is the full path to the working copy of the file. The filename will have a guid added to the end.
OriginalName	Path to the original file. The file no longer exists b/c the service deletes it after it has copied the file to the working directory.
FileTypeID	ID of the file type. Used to get configuration information about the file (properties, workflow strong name, file pattern, etc)
DirectoryID	ID of the directory the file was found in. Used to get configuration information about the directory.

All custom actions that interact with the file will need to use the WorkItem object to find the file. The Filename property points to the file in the working directory. This filename can be used to manipulate the actual file (ie open the file and read the contents). The OriginalName is the original path in case it is necessary to know the original filename without the added information. This value cannot be used to manipulate the file because there should not be a file at the location pointed to.
The FileTypeID and DirectoryID are keys in to the configuration system. As explained in Configuration, using FileTypeID will allow you to get the FileType information so that you can use the properties in your own code. The same is true of the DirectoryID property.

Configuration
Grinder's configuration object is loaded from an XML file and serialized in to deceptively named Configuration class inside the GuerillaProgrammer.FileProcessor.Settings namespace. The configuration file's persistence is controlled by the ConfigurationManager object in the GuerillaProgrammer.FileProcessor namespace.

Using the Configuration object you can read and write the settings used by the grinder service. In the case of a custom action you can use the configuration object to get the settings for a FileType or Directory using the IDs from the WorkItem object.

Building Custom Actions

All of Grinder's actions are Workflow Actions that are children of the Activity class. Any book or article on Windows Workflow Activities will explain the basics of writing a general WF Activity. The specifics for Grinder has to do with how Grinder hosts its custom activities and communicates with them.

If your activity needs to access information about the WorkItem being processed it will need to reference the FileProcessorUtilities assembly. Inside the Execute event of the activity you will need to access the IProcessorDataService in order to communicate with

the Grinder service. Once you have an instance of the IProcessorDataService you can call GetWorkItem and GetConfiguration to access the information your Action needs. Keep in mind that your Action can change values inside the WorkItem, but that will not affect the workflow itself. So for instance you can change the FileName value if your action moved it or changed it. *While you can do this it is recommended that you don't.*

To illustrate the basics of writing an activity here is the code for the Debug Execute event handler:

```
protected override ActivityExecutionStatus Execute(ActivityExecutionContext executionContext)
{
    Logger.LogEnter();
    IProcessorDataService dataService = executionContext.GetService(typeof(IProcessorDataService)) as IProcessorDataService;
    if (null == dataService)
    {
        throw new InvalidOperationException("Can't get the service!");
    }

    WorkItem workItem = dataService.GetWorkItem();
    /*
        The trace message will look like:
        Grinder Trace - WorkItem information
        Processing time: yyyyMMdd hhmmss
        DirectoryID:
        FileTypeID:
        Filename:

    */
    string message = string.Format("Grinder Trace - WorkItem information\nProcessing time" + "
                                + ": {0}\nDirectory ID: {1}\nFiletype ID: {2}\nFilename: {3}"
                                + "\n",DateTime.Now.ToString("yyyyMMdd hhmmss")
                                ,workItem.DirectoryID, workItem.Filename, workItem.FileTypeID);
    Trace.Write(message);
    Logger.LogLeave();
    return base.Execute(executionContext);
}
```

The important line is the second line where executionContext.GetService. This is where we get the IProcessorDataService. The rest of the code is just a standard debug trace.

Once you have created and tested your action you have to register it in the designer's configuration file. To do that put the strong name of the class in DesignerActivities section. For the debugger action:

```
<DesignerActivities>
    <Activity Type="DebugTrace,FileProcessorActivities" />
    <Activity Type="FileCopy,FileProcessorActivities" />
    <Activity Type="FileDelete,FileProcessorActivities" />
    <Activity Type="FileMove,FileProcessorActivities" />
    <Activity Type="ExecuteApplication,FileProcessorActivities" />
    <Activity Type="UnzipFile,FileProcessorActivities" />
    <Activity Type="ZipFile,FileProcessorActivities" />
</DesignerActivities>
```

Building Workflows with Visual Studio

The Grinder workflow design is very restrictive in terms of what is allowed. It doesn't allow looping, branching, or code activities. The intent of Grinder's designer is to allow non-programmers to create workflows. However, some workflows will require actual

programming so Visual Studio is needed. Create a Workflow project, build it as usual and add it to Grinder's configuration as usual.