Manual del código

de la

máquina láser



Gonzalo Valero Domingo

Ringo Válvulas

Resumen

El código se puede dividir en dos ramas diferentes, extracción de los archivos desde un servidor FTP y desde un directorio local. La puesta en marcha es igual para ambas ramas, se lanza un thread (hilo) para realizar un envío de archivo.

Este envío de archivo tiene diferentes etapas; la primera es extraer la información del directorio origen, será diferente en cada rama, posteriormente se espera a que el directorio destino esté liberado. Una vez que lo esté se ordenan los archivos leídos del directorio y se selecciona el primero que tenga la extensión correcta. Una vez seleccionado el archivo se extrae del directorio origen y se copia en el directorio destino y el directorio de archivos procesados.

Este proceso tiene ligeras variaciones entre las dos ramas, pero el cuerpo principal es el mismo.

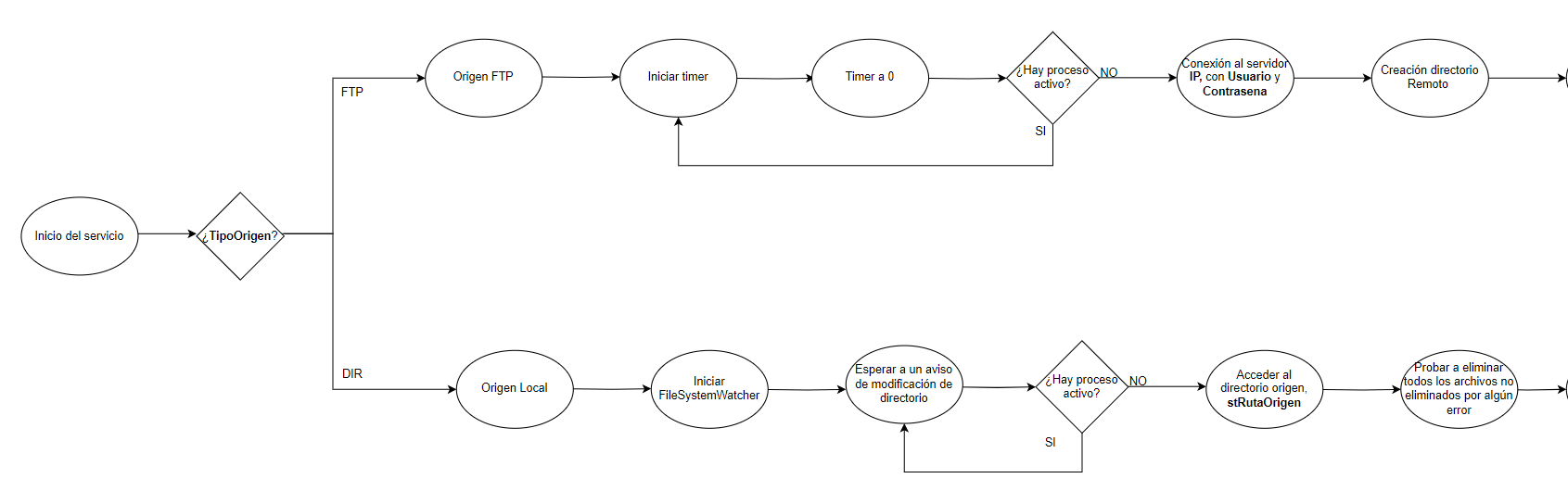
La mayor diferencia entre las dos ramas es el envío continuo de archivos posterior al primero.

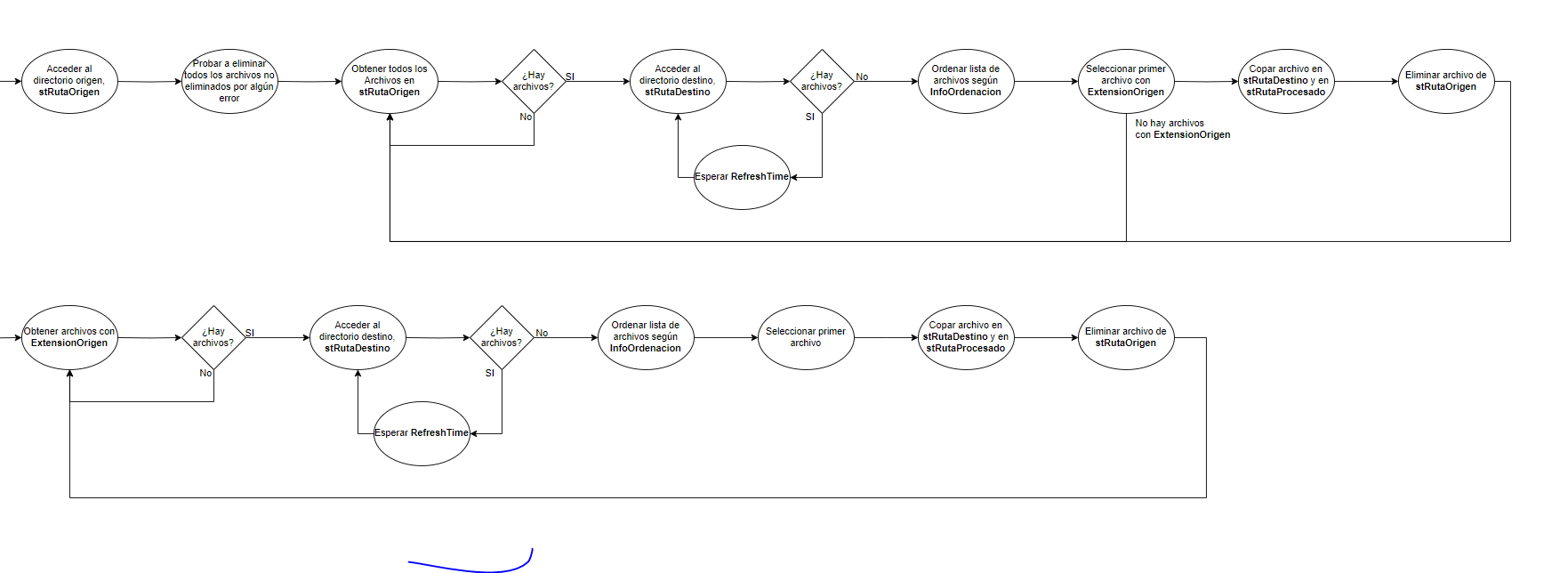
En el caso de origen local se utiliza el controller FileSystemWatcher, este controller se asocia a un directorio local y cada vez que se realiza un cambio en este directorio realiza una acción, es decir, cada vez que se copia o se elimina un archivo en el directorio asociado este realiza una acción, en este caso enviar un archivo. También se puede modificar el controller para que solo reaccione a archivos con un nombre o extensión.

En el caso de origen FTP se utiliza un Timer (temporizador), este tiene un tiempo de ejecución que viene dado por una de las variables del fichero .ini. Su funcionamiento es sencillo, cada x tiempo se realizará una acción, que en este caso es enviar un archivo.

Aunque no siempre realizará esta acción dependerá de si ya hay un thread que esté en proceso de enviar un archivo, en cuyo caso no se realizará ninguna acción.

Diagrama de uso





Distribución

El servicio Windows está compuesto por cuatro ficheros de código en c# y dos ficheros de configuración.

Lo ficheros de configuración son:

* Fichero .ini: Este contiene todos los parámetros configurables necesarios para la ejecución del código. Son un total de trece:

1. **bLogDetallado:** De tipo booleano, si vale true indica que se requiere toda la información del programa en el fichero log, en caso contrario solo se recogerá los warnings y los errores fatales que se generen.
2. **TipoOrigen:** De tipo string, puede tomar dos valores, “DIR” si se desea que el directorio del que se extraigan los archivos sea uno local, o “FTP” si se desea que este en un servidor FTP.
3. **stRutaOrigen:** De tipo string, tendrá el valor de la ruta absoluta del directorio del que se extraigan los archivos, esta ruta absoluta puede ser del equipo local o del servidor FTP.
4. **stRutaDestino:** De tipo string, tendrá el valor de la ruta absoluta del directorio al que se enviarán los archivos para la posterior extracción de la maquina láser.
5. **stRutaProcesado:** De tipo string, tendrá el valor de la ruta absoluta del directorio al que se copiaran los archivos que hayan sido copiados en el directorio destino, es decir los archivos que hayan sido enviados a imprimir.
6. **DescProcesado:** De tipo string, descripción que se añadirá al nombre del archivo que se copia en el directorio stRutaProcesado.

Ejemplo: valor =\_procesado, archivo.txt se copia como archivo\_procesado.txt.

1. **NombreDestino:** De tipo string, nombre que detecta la máquina láser. Este valor es el que obtendrá el archivo una vez se copie en stRutaDestino.

Ejemplo: valor = Test\_1, archivo.txt se copia como Test\_1.txt.

1. **InfoOrdenacion:** De tipo string, tipo de ordenación con la que se enviarán los archivos desde el directorio stRutaOrigen hasta stRutaDestino. Puede tomar hasta seis valores diferentes, AlfAsc (Alfabético ascendente), AlfDesc (Alfabético descendente), CreAsc (Fecha de creación ascendente), CreDesc (Fecha de creación descendente), ModAsc (Fecha de última modificación ascendente), ModDesc (Fecha de última modificación descendente).
2. **RefreshTime:** De tipo entero, tiempo de espera hasta volver a revisar si existe el archivo con nombre NombreDestino en stRutaDestino. Además, es el intervalo que tiene el controlador timer.
3. **IP:** De tipo string, en el caso de que el TipoOrigen sea FTP almacenará la dirección IP del servidor FTP.
4. **Usuario:** De tipo string, en el caso de que el TipoOrigen sea FTP almacenará el nombre del usuario con el que se conectará al servidor FTP.
5. **Contrasena:** De tipo string, en el caso de que el TipoOrigen sea FTP almacenará la contraseña del usuario con el que se conectará al servidor FTP.
6. **ExtensionOrigen:** De tipo string, contiene la extensión de los archivos que se desean enviar a la máquina laser. Ejemplo: “.txt”, “.xml”.

* Fichero de configuración del log: Se utiliza para controlar el comportamiento del registro de eventos en una aplicación. A través de este archivo, se pueden especificar diferentes niveles de registro, destinos de registro (appender) y el formato del mensaje de registro. La explicación apartado por apartado se encuentra en el código.

Los ficheros de código son:

* Variables: Es el encargado de almacenar, en una clase estática, el valor de los parámetros asignados en el fichero de configuración, la variable del log y un método que recibe como parámetro InfoOrdenacion (variable del fichero .ini) y devuelve el constructor de la clase que corresponde a ese tipo de ordenación.
* ClasesOrdenacion: Contiene un conjunto de clases. La clase abstracta de la que derivan otras seis clases se llama TiposDeOrdenacion, esta implementa un comparador para las clases FileInfo (Es una clase estandar que contiene la información de los archivos, se utilizará cuando TipoOrigen sea DIR) y InformacionFicheros.

InformaciónFicheros es una clase que almacena la información necesaria para los seis tipos de ordenación que se tiene. En el caso de querer añadir otro tipo de ordenación sería necesario añadir una variable a esta clase y leerla del archivo al igual que el resto (nombre, fecha creación y fecha última modificación). Esta clase se ha creado debido a no poder leer la información de los archivos del servidor FTP para su posterior ordenación.

* OrigenLocal: Clase que contiene dos parámetros y dos métodos. El método de esta clase será llamado en el caso de que la ejecución sea de origen local.

El primer parámetro es **Error**, es estático y de tipo booleano, inicialmente toma valor false y solo tomará valor true en el caso de que haya sucedido un error que impida el funcionamiento del servicio (stRutaOrigen no exista, hayá habido un error no controlado o CopiarArchivo ha devuelto false), se utilizará para detener la lectura y envío de archivos a stRutaDestino.

El segundo parámetro se llama **ArchviosNoElimnados** es una lista de strings que almacena archivos que no se hayan podido eliminar de stRutaOrigen pero que si hayan sido enviados y por ello impresos. Se almacena la ruta absoluta del archivo. El servicio no enviará de nuevo estos archivos y en cada nuevo envió de archivo se intentarán eliminar.

El primer método de la clase, que es el único accesible desde el exterior, es **OrigenLocal**. No recibe ningún parametro. Se encarga de comprobar que stRutaOrigen existe, posteriormente intenta eliminar los archivos almacenados en ArchivosNoEliminados, por último, obtiene la información de los archivos con ExtensionOrigen en stRutaOrigen y los almacena en una lista para en el caso que no esté vacía llamar al otro método de la clase, CopiarArchivo pasándole como parámetro esta lista de información de archivos.

El segundo método de la clase es **CopiarArchivo.** Recibe como parámetro una lista, no vacía, con la información de archivos. Este método se encarga de realizar la espera hasta que se pueda enviar más archivos a stRutaDestino, para posteriormente ordenar la lista según el método TipoOrdenación y su parámetro InfoOrdenación. De esta lista ya ordenada se seleccionará el primer archivo que no esté en la lista de Archivos no eliminados. Una vez seleccionado el archivo se copiará en stRutaDestino y en stRutaProcesado y se procederá a su eliminación del directorio stRutaOrigen.

Este método devuelve un valor booleano, este tomará valor true si la ejecución ha ido bien o ha habido un error que no impide un correcto funcionamiento futuro (por ejemplo que no se haya podido copiar el archivo en alguno de los dos directorios o que no se haya podido eliminar de stRutaOrigen, en este caso el archivo se añadirá a la lista de ArchivosNoEliminados) y valor false en aso de un error que impide el funcionamiento futuro (stRutaDestino y stRutaProcesado no existen o un error no controlado).

* OrigenFTP: Clase que contiene dos parámetros y dos métodos. El método de esta clase será llamado en el caso de que la ejecución sea de origen local.

El primer parámetro es **Error**, es estático y de tipo booleano, inicialmente toma valor false y solo tomará valor true en el caso de que haya sucedido un error que impida el funcionamiento del servicio (stRutaOrigen no exista, hayá habido un error no controlado o CopiarArchivo ha devuelto false), se utilizará para detener la lectura y envío de archivos a stRutaDestino.

El segundo parámetro se llama **ArchviosNoElimnados** es una lista de strings que almacena archivos que no se hayan podido eliminar de stRutaOrigen pero que si hayan sido enviados y por ello impresos. Almacena solo el nombre del archivo. El servicio no enviará de nuevo estos archivos y en cada nuevo envió de archivo se intentarán eliminar.

El primer método de la clase, que es el único accesible desde el exterior, es **OrigenFTP**. No recibe ningún parametro. Se encarga de conectarse al servidor FTP (con IP, usuario y contraseña otorgadas), de la creación de un directorio temporal para la descarga de archivos, de comprobar que stRutaOrigen existe, intenta eliminar los archivos almacenados en ArchivosNoEliminados, y por último, obtiene la información de los archivos en stRutaOrigen y los almacena en una lista para en el caso que no esté vacía llamar al otro método de la clase, CopiarArchivo pasándole como parámetro esta lista de información de archivos, la conexión al servidor y la ruta del directorio remoto.

El segundo método de la clase es **CopiarArchivo.** Recibe como parámetro una lista, no vacía, con la información de archivos, una conexión a un servidor y la ruta de un directorio temporal. Este método se encarga de realizar la espera hasta que se pueda enviar más archivos a stRutaDestino, para posteriormente ordenar la lista según el método TipoOrdenación y su parámetro InfoOrdenación. De esta lista ya ordenada se seleccionará el primer archivo que no esté en la lista de Archivos no eliminados y que tenga la extensión Nobrextension. Una vez seleccionado el archivo se descarará en el directorio remoto, se copiará en stRutaDestino y en stRutaProcesado y se procederá a su eliminación del directorio stRutaOrigen.

Este método devuelve un valor booleano, este tomará valor true si la ejecución ha ido bien o ha habido un error que no impide un correcto funcionamiento futuro (por ejemplo, que no se haya podido descargar el archivo, que no se haya podido copiar el archivo en alguno de los dos directorios o que no se haya podido eliminar de stRutaOrigen, en este caso el archivo se añadirá a la lista de ArchivosNoEliminados) y valor false en aso de un error que impide el funcionamiento futuro (stRutaDestino y stRutaProcesado no existen o un error no controlado).

On start

El comienzo del servicio pondrá la variable ProcesoActivo a true para indicar que ya hay un proceso en ejecución, el resto depende de si el origen es local o desde un servidor. En el caso de que sea local, asociará al controller FileSystemWatcher la ruta del directorio a observar y la extensión de los archivos relevantes. En cambio, si es desde un servidor, da un valor al intervalo del temporizador y lo comenzará.

Para finalizar lanzará el proceso correspondiente y realizará un envío de archivo.

1. ProcesoActivo = **true**;
2. Thread mensajero;
4. // Lanzamos la primera ejecución, que generará una reacción en cadena
5. **if** (VariablesGlobales.TipoOrigen.Equals("DIR"))
6. {
7. **try**
8. {
9. // Añadimos la ruta del directorio que estará vigilando
10. visorFichero.Path = VariablesGlobales.stRutaOrigen;
12. }
13. **catch** (Exception ex)
14. {
15. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL: La ruta del directorio origen \"" + VariablesGlobales.stRutaOrigen + "\" no existe ERROR: " + ex.Message);
17. }
18. // El visor solo reaccionará a cambios en la carpeta con la extensión dada
19. visorFichero.Filter = "\*" + VariablesGlobales.ExtensionOrigen;
20. mensajero = [new](http://www.google.com/search?q=new+msdn.microsoft.com) Thread([new](http://www.google.com/search?q=new+msdn.microsoft.com) ThreadStart(Local.OrigenLocal));
21. }
22. **else** // if (VariablesGlobales.TipoOrigen.Equals("FTP"))
23. {
24. Temporizador.Interval = VariablesGlobales.RefreshTime;
25. Temporizador.Start();
26. mensajero = [new](http://www.google.com/search?q=new+msdn.microsoft.com) Thread([new](http://www.google.com/search?q=new+msdn.microsoft.com) ThreadStart(FTP.OrigenFTP));
28. }
30. mensajero.Start();
31. mensajero.**Join**();
32. ProcesoActivo = **false**;
33. }

On stop

El proceso de parada elimina el directorio remoto creado cuando el origen es desde un servidor. Tambien se detiene el temporizador.

1. **if** (VariablesGlobales.TipoOrigen.Equals("FTP"))
2. {
3. Temporizador.Stop();
4. **if** (Directory.Exists("REMOTO\\"))
5. {
6. Directory.Delete("REMOTO\\",**true**);
7. }
8. }

Controllers

FileSystemWatcher

En el caso de que el origen sea local se utiliza este controller. Lo primero que hace es comprobar que el booleano error de la clase Local no sea true, este tomará valor true cuando haya habido un error que impida la ejecución por completo del programa (no existe la ruta origen, destino o procesado). Posteriormente se hace una parada para que se copien todos los archivos en el caso de que se copien muchos archivos a la vez.

Si no hay ningún proceso activo, este pone ProcesoActivo a true y lanza el nuevo proceso una vez que acaba vuelve a poner ProcesoActivo a false. En el caso de que si haya un proceso en ejecución termina esta acción.

1. **if** (Local.Error)
2. {
3. visorFichero.EnableRaisingEvents = **false**;
4. }
5. Thread.Sleep(10); // Pausa para darle tiempo a los archivos a copiarse
6. **if** (ProcesoActivo) { **return**; }
7. ProcesoActivo = **true**;
9. Thread mensajero;
10. mensajero = [new](http://www.google.com/search?q=new+msdn.microsoft.com) Thread([new](http://www.google.com/search?q=new+msdn.microsoft.com) ThreadStart(Local.OrigenLocal));

13. mensajero.Start();
14. mensajero.**Join**();
15. ProcesoActivo = **false**;

Timer

Se asegura de que el origen es desde un servidor FTP. Al igual que FileSystemWatcher lo primero que hace es comprobar que el booleano error de la clase FTP no sea true, este tomará valor true cuando haya habido un error que impida la ejecución por completo del programa (no sea haya podido conectar al servidor o no existe la ruta origen, destino o procesado).

Para finalizar, si no hay ningún proceso activo, este pone ProcesoActivo a true y lanza el nuevo proceso una vez que acaba vuelve a poner ProcesoActivo a false. En el caso de que si haya un proceso en ejecución termina esta acción.

1. **if** (VariablesGlobales.TipoOrigen.Equals("FTP"))
2. {
3. **if** (FTP.Error)
4. {
5. Temporizador.Enabled = **false**;
6. }
8. **if** (ProcesoActivo) { **return**; }
10. ProcesoActivo = **true**;
12. Thread mensajero;
13. mensajero = [new](http://www.google.com/search?q=new+msdn.microsoft.com) Thread([new](http://www.google.com/search?q=new+msdn.microsoft.com) ThreadStart(FTP.OrigenFTP));

16. mensajero.Start();
17. mensajero.**Join**();
19. ProcesoActivo = **false**;
20. }

Local

OrigenLocal

El método empieza comprobando que la ruta origen exista, en caso de que no exista generará un error fatal, el cual se soluciona reiniciando el servicio e introduciendo la ruta correcta:

1. **if** (!Directory.Exists(VariablesGlobales.stRutaOrigen))
2. {
3. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL La ruta origen no existe " + VariablesGlobales.sLogServiceName + "...");
4. Error = **true**;
5. **return**;
6. }

Posteriormente se accede al directorio origen y se obtiene la información de todos los ficheros con *ExtensiónOrigen* (variable del archivo .ini):

1. DirectoryInfo directorio = [new](http://www.google.com/search?q=new+msdn.microsoft.com) DirectoryInfo(VariablesGlobales.stRutaOrigen); // Se obtiene toda la información del directorio <stRutaOrigen>
3. List<FileInfo> ListaArchivos = directorio.GetFiles("\*" + VariablesGlobales.ExtensionOrigen).ToList<FileInfo>(); // Se almacena la información de sus archivos con <ExtensionOrigen> en una lista

Para acabar con el método OrigenLocal, si existen archivos en la lista llamaremos al método CopiarArchivo que se encarga de enviar el archivo a impresión:

1. **if** (ListaArchivos.Count != 0)
2. {
3. **if** (!CopiarArchivo(ListaArchivos))
4. {
5. Error = **true**;
6. **return**;
7. }
8. }

CopiarArchivo devuelve false si la ruta destino no existe, si la ruta procesados no existe y no se puede crear una, o si ha habido una excepción no controlada.

CopiarArchivo

El método empieza comprobando que la ruta destino exista, en caso de que no exista generará un error fatal, el cual se soluciona reiniciando el servicio e introduciendo la ruta correcta:

1. **if** (!Directory.Exists(VariablesGlobales.stRutaDestino))
2. {
3. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL La ruta destino no existe " + VariablesGlobales.sLogServiceName + "...");
4. Error = **true**;
5. **return**;
6. }

También se comprueba la ruta procesados, en el caso de que no exista se intenta crear un directorio con el nombre solicitado y en el caso de que no se pueda crear dicho directorio generará un error fatal, el cual se soluciona reiniciando el servicio e introduciendo la ruta correcta:

1. // Se comprueba que el directorio <stRutaProcesado> exista, en caso de que no se envía un aviso al log y se intenta crear.
2. // En caso de que no se haya podido crear se envia un mensaje fatal al log.
3. **if** (!Directory.Exists(VariablesGlobales.stRutaProcesado))
4. {
5. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " WARNING La ruta destino procesados no existe, se procede a crear directorio \"" + VariablesGlobales.stRutaProcesado + "\" " + VariablesGlobales.sLogServiceName + "...");
6. **try**
7. {
8. Directory.CreateDirectory(VariablesGlobales.stRutaProcesado);
9. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " WARNING Se ha creado el directorio \"" + VariablesGlobales.stRutaProcesado + "\" " + VariablesGlobales.sLogServiceName);
11. }
12. **catch** (Exception ex)
13. {
14. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL No se ha podido crear el directorio \"" + VariablesGlobales.stRutaProcesado + "\" " + VariablesGlobales.sLogServiceName + "ERROR: " + ex.Message);
15. **return** **false**;
16. }
17. }

La siguiente línea de código se encarga de comprobar cada *RefreshTime* tiempo (variable del archivo .ini), que no exista el archivo con el nombre que lee la máquina láser en el directorio destino, es decir realiza una espera activa hasta que el archivo con *NombreDestino*  (variable del archivo .ini) desaparezca del directorio destino.

1. // Se espera a que no exista un archivo con nombre <NombreDestino> en el directorio <stRutaDestino>, en el caso de uqe exista esperaremos <RefreshTime> ms antes de volver a comprobarlo
2. **while** (File.Exists(VariablesGlobales.stRutaDestino + VariablesGlobales.NombreDestino))
3. { Thread.Sleep(VariablesGlobales.RefreshTime); }

El siguiente fragmento se encarga de los posibles archivos que hayan sido enviados e impresos, pero debido a algún problema no se hayan podido eliminar del directorio origen, estos archivos como se verá más tarde se almacenan en *ArchivosNoEliminados* variable estática de la clase Local.

Este bucle simplemente recorre todos los elementos de *ArchivosNoEliminados* e intenta volver a eliminarlos:

1. // Intentamos volver a eliminar los archivos no eliminados
2. **for** (**int** i = 0; i < ArchivosNoEliminados.Count(); i++)
3. {
4. **string** nombre = ArchivosNoEliminados[i];
5. ArchivosNoEliminados.**Remove**(nombre);
7. **try**
8. {
9. File.Delete(nombre);
10. **if** (VariablesGlobales.bLogDetallado)
11. {
12. VariablesGlobales.Log.Info(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " INFO Se ha eliminado el archivo \"" + nombre + "\" de la carpeta \"" + VariablesGlobales.stRutaOrigen + "\" " + VariablesGlobales.sLogServiceName + "...");
13. }
14. }
15. **catch** (Exception ex)
16. {
17. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL: No se ha podido eliminar el archivo \"" + nombre + "\" de la carpeta \"" + VariablesGlobales.stRutaOrigen + "\" " + VariablesGlobales.sLogServiceName + " ERROR: " + ex.Message);
18. ArchivosNoEliminados.**Add**(nombre);
20. }
21. }

Se ordena la lista de ficheros *ListaArchivos* pasada como parámetro al método, mediante el tipo de ordenación *InfoOrdenacion* (variable del archivo .ini):

1. ListaArchivos.Sort(VariablesGlobales.TipoOrdenacion(VariablesGlobales.InfoOrdenacion));

Posteriormente se realiza un bucle que se encarga de seleccionar el primer archivo de *ListaArchivos* que no esté en la lista de *ArchivosNoEliminados,* ya que si está en la lista ya ha sido impreso. En el caso de que salga del bucle debido a que *j* es igual al conteo de los componentes de la lista, entonces no existirá ningún archivo restante por enviar, por lo tanto no se enviará ninguno pero seguirá el servicio seguirá activo:

1. FileInfo archivo;
2. **int** j = 0;
3. FileInfo archivo = ListaArchivos[j];
4. **for**(; ArchivosNoEliminados.Contains(archivo.FullName) && (j < ListaArchivos.Count()); j++)
5. {
6. archivo = ListaArchivos[j];
7. }
9. **if**(j == ListaArchivos.Count())
10. {
11. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " WARNING: Todos los archivos del directorio \"" + VariablesGlobales.stRutaOrigen + "\" han sido ya impresos y deben ser eliminados " + VariablesGlobales.sLogServiceName);
12. **return** **true**;
13. }

En este momento ya se tiene un archivo listo para enviar, por lo que se procede a copiarse tanto al directorio destino cambiando el nombre a *NombreDestino* (variable del archivo .ini), como al directorio procesados cambiando el nombre a "<nombreSinExtension> + <DescProcesado> + <ExtensionOrigen>":

1. // El archivo extraido se copia en el directorio <stRutaDestino> con nombre <NombreDestino>
2. File.Copy(archivo.FullName, VariablesGlobales.stRutaDestino + VariablesGlobales.NombreDestino, **true**);
3. // Se elimina la extension del fichero para poder cambiarle el nombre
4. **string** nombreSinExtension = archivo.Name.Replace(VariablesGlobales.ExtensionOrigen, "");
6. // El archivo extraido se copia en el directorio <stRutaProcesado> con nombre "<nombreSinExtension> + <DescProcesado> + <ExtensionOrigen>"
7. File.Copy(archivo.FullName, VariablesGlobales.stRutaProcesado + nombreSinExtension + VariablesGlobales.DescProcesado + VariablesGlobales.ExtensionOrigen, **true**);

Por último, se procede a la eliminación del archivo ya copiado, como previamente se ha indicado en el caso de que un archivo no pueda ser eliminado del directorio origen, guardaremos su ruta en la lista *ArchivosNoEliminados.* En cada ejecución de CopiarArchivo, se intentarán volver a eliminar y en ninguno de los casos se volverá a seleccionar alguno de los archivos de esta lista para enviar:

1. **try**
2. {
3. // El archivo extraído se elimina del servidor FTP de la ruta <stRutaOrigen>
4. File.Delete(archivo.FullName);
5. **if** (VariablesGlobales.bLogDetallado)
6. {
7. VariablesGlobales.Log.Info(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " INFO Se ha eliminado el archivo \"" + archivo.Name + "\" de la carpeta \"" + VariablesGlobales.stRutaOrigen + "\" " + VariablesGlobales.sLogServiceName + "...");
8. }

11. }
12. **catch** (Exception ex)
13. {
14. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL: No se ha podido eliminar el archivo \"" + archivo.Name + "\" de la carpeta \"" + VariablesGlobales.stRutaOrigen + "\" " + VariablesGlobales.sLogServiceName + " ERROR: " + ex.Message);
15. ArchivosNoEliminados.**Add**(archivo.FullName);
16. **return** **true**;
17. }

FTP

OrigenFTP

Se comienza la ejecución de OrigenFTP con la conexión del usuario al servidor, en el caso que el host, el usuario o la contraseña no sean correctos generará un error fatal y parará la ejecución del servicio.

1. **try**
2. {
3. // Se crea un ftp con la ip del host, un usuario y su contraseña
4. ftp = [new](http://www.google.com/search?q=new+msdn.microsoft.com) FtpClient(VariablesGlobales.IP, VariablesGlobales.Usuario, VariablesGlobales.Contrasena);
5. ftp.Connect();
6. }
7. **catch** (Exception ex)
8. {
9. // No se ha podido crear
10. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL No se ha podido conectar al servidor FTP " + VariablesGlobales.sLogServiceName + " ERROR: " + ex.Message);
11. Error = **true**;
12. **return**;
13. }

Se crea un directorio remoto para almacenar los archivos que van siendo descargados del servidor. Este directorio se eliminará cuando se detenga el servicio.

1. **string** remoto = "REMOTO\\";
3. // Se crea un directorio temporal para almacenar los archivos que se vayan descargando
4. **if** (!Directory.Exists(remoto))
5. {
6. **try**
7. {
8. Directory.CreateDirectory(remoto);
9. }
10. **catch** (Exception ex)
11. {
12. // Si no se puede crear se muestra un Warning
13. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " No existe y no se ha podido crear el directorio \"" + remoto + "\" los archivos se descargaran en la ubicación actual " + VariablesGlobales.sLogServiceName + " ERROR: " + ex.Message);
15. }
16. }

Se comprueba que la ruta origen exista, en caso de que no exista generará un error fatal, el cual se soluciona reiniciando el servicio e introduciendo la ruta correcta:

1. **if** (!Directory.Exists(VariablesGlobales.stRutaOrigen))
2. {
3. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL La ruta origen no existe " + VariablesGlobales.sLogServiceName + "...");
4. Error = **true**;
5. **return**;
6. }

Posteriormente se extrae la información de todos los archivos del directorio ruta origen y se almacena en una lista de la clase InformacionFicheros que contiene nombre, fecha de creación y fecha de modificación del archivo.

1. List<InformacionFicheros> ListaArchivos = [new](http://www.google.com/search?q=new+msdn.microsoft.com) List<InformacionFicheros>();
3. // Se almacena la información de todos los archivos del directorio en una lista
4. **foreach** (FtpListItem item **in** ftp.GetListing(VariablesGlobales.stRutaOrigen))
5. {
6. **if** (item.Type == FtpObjectType.File)
7. {
8. InformacionFicheros fichero = [new](http://www.google.com/search?q=new+msdn.microsoft.com) InformacionFicheros();
10. fichero.Name = item.Name;
11. fichero.CreationTime = item.Created.ToString();
12. fichero.LastWriteTime = item.Modified.ToString();
14. ListaArchivos.**Add**(fichero);
15. }
17. }

Para acabar con el método OrigenLocal, si existen archivos en la lista llamaremos al método CopiarArchivo que se encarga de enviar el archivo a impresión:

1. **if** (ListaArchivos.Count != 0)
2. {
3. **if** (!CopiarArchivo(ListaArchivos))
4. {
5. Error = **true**;
6. **return**;
7. }
8. }

CopiarArchivo devuelve false si la ruta destino no existe, si la ruta procesados no existe y no se puede crear una, o si ha habido una excepción no controlada.

CopiarArchivo

El método empieza comprobando que la ruta destino exista, en caso de que no exista generará un error fatal, el cual se soluciona reiniciando el servicio e introduciendo la ruta correcta:

1. **if** (!Directory.Exists(VariablesGlobales.stRutaDestino))
2. {
3. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL La ruta destino no existe " + VariablesGlobales.sLogServiceName + "...");
4. Error = **true**;
5. **return**;
6. }

También se comprueba la ruta procesados, en el caso de que no exista se intenta crear un directorio con el nombre solicitado y en el caso de que no se pueda crear dicho directorio generará un error fatal, el cual se soluciona reiniciando el servicio e introduciendo la ruta correcta:

1. // Se comprueba que el directorio <stRutaProcesado> exista, en caso de que no se envía un aviso al log y se intenta crear.
2. // En caso de que no se haya podido crear se envia un mensaje fatal al log.
3. **if** (!Directory.Exists(VariablesGlobales.stRutaProcesado))
4. {
5. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " WARNING La ruta destino procesados no existe, se procede a crear directorio \"" + VariablesGlobales.stRutaProcesado + "\" " + VariablesGlobales.sLogServiceName + "...");
6. **try**
7. {
8. Directory.CreateDirectory(VariablesGlobales.stRutaProcesado);
9. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " WARNING Se ha creado el directorio \"" + VariablesGlobales.stRutaProcesado + "\" " + VariablesGlobales.sLogServiceName);
11. }
12. **catch** (Exception ex)
13. {
14. VariablesGlobales.Log.Fatal(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL No se ha podido crear el directorio \"" + VariablesGlobales.stRutaProcesado + "\" " + VariablesGlobales.sLogServiceName + "ERROR: " + ex.Message);
15. **return** **false**;
16. }
17. }

La siguiente línea de código se encarga de comprobar cada *RefreshTime* tiempo (variable del archivo .ini), que no exista el archivo con el nombre que lee la máquina láser en el directorio destino, es decir realiza una espera activa hasta que el archivo con *NombreDestino*  (variable del archivo .ini) desaparezca del directorio destino.

1. // Se espera a que no exista un archivo con nombre <NombreDestino> en el directorio <stRutaDestino>, en el caso de uqe exista esperaremos <RefreshTime> ms antes de volver a comprobarlo
2. **while** (File.Exists(VariablesGlobales.stRutaDestino + VariablesGlobales.NombreDestino))
3. { Thread.Sleep(VariablesGlobales.RefreshTime); }

El siguiente fragmento se encarga de los posibles archivos que hayan sido enviados e impresos, pero debido a algún problema no se hayan podido eliminar del directorio origen, estos archivos como se verá más tarde se almacenan en *ArchivosNoEliminados* variable estática de la clase FTP.

Este bucle simplemente recorre todos los elementos de *ArchivosNoEliminados* e intenta volver a eliminarlos:

1. **for** (**int** i = 0; i < ArchivosNoEliminados.Count(); i++)
2. {
3. **string** nombre = ArchivosNoEliminados[i];
4. ArchivosNoEliminados.**Remove**(nombre);
6. **try**
7. {
8. // El archivo extraido se elimina del servidor FTP de la ruta <stRutaOrigen>
9. ftp.DeleteFile(VariablesGlobales.stRutaOrigen + nombre);
10. **if** (VariablesGlobales.bLogDetallado)
11. {
12. VariablesGlobales.Log.Info(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " INFO Se ha eliminado el archivo \"" + VariablesGlobales.stRutaOrigen + nombre + "\" del servidor.");
13. }

16. }
17. **catch** (Exception ex)
18. {
19. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL: No se ha podido eliminar el archivo \"" + VariablesGlobales.stRutaOrigen + nombre + "\" del servidor. ERROR: " + ex.Message);
20. ArchivosNoEliminados.**Add**(nombre);
21. **return** **true**;
22. }
23. }

Se ordena la lista de ficheros *ListaArchivos* pasada como parámetro al método, mediante el tipo de ordenación *InfoOrdenacion* (variable del archivo .ini):

1. ListaArchivos.Sort(VariablesGlobales.TipoOrdenacion(VariablesGlobales.InfoOrdenacion));

Posteriormente se realiza un bucle que se encarga de seleccionar el primer archivo de *ListaArchivos* que no esté en la lista de *ArchivosNoEliminados,* ya que si está en la lista ya ha sido impreso. Además comprueba que el archivo tenga la extensión deseada. En el caso de que salga del bucle debido a que *j* es igual al conteo de los componentes de la lista, entonces no existirá ningún archivo con extensión correcta restante por enviar, por lo tanto no se enviará ninguno pero seguirá el servicio seguirá activo:

1. // Bucle encargado de buscar cual es el primer archivo con extension <ExtensionOrigen>
2. **int** j = 0;
3. InformacionFicheros archivo = ListaArchivos[j];
4. **for** (; !archivo.Name.EndsWith(VariablesGlobales.ExtensionOrigen) && ArchivosNoEliminados.Contains(archivo.Name) && (j < ListaArchivos.Count()); j++)
5. {
6. archivo = ListaArchivos[j];
7. }
9. // Si no hay ningun archivo con <ExtensionOrigen>
10. **if** (j == ListaArchivos.Count())
11. {
12. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " WARNING: Los archivos del directorio \"" + VariablesGlobales.stRutaOrigen + "\" han sido ya impresos y deben ser eliminados o no tienen la extension deseada " + VariablesGlobales.sLogServiceName);
13. **return** **true**;
15. }

Se intenta descargar el archivo desde el servidor al directorio remoto, en caso de que suceda algún error, se producirá un error fatal pero la ejecución continuará:

1. **try**
2. {
3. // Se descarga el primer archivo del servidor FTP a un directorio provisional
4. ftp.DownloadFile(remoto + archivo.Name, VariablesGlobales.stRutaOrigen + archivo.Name, FtpLocalExists.Overwrite);
5. }
6. **catch** (Exception ex) // En caso de no poder descargarlo se envía un mensaje fatal al log
7. {
8. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL: No se ha podido descargar el archivo \"" + VariablesGlobales.stRutaOrigen + archivo.Name + "\" del servidor. ERROR: " + ex.Message);
9. **return** **true**;
10. }

En este momento ya se tiene un archivo listo para enviar, por lo que se procede a copiarse tanto al directorio destino cambiando el nombre a *NombreDestino* (variable del archivo .ini), como al directorio procesados cambiando el nombre a "<nombreSinExtension> + <DescProcesado> + <ExtensionOrigen>":

1. // El archivo extraido se copia en el directorio <stRutaDestino> con nombre <NombreDestino>
2. File.Copy(remoto + archivo.Name, VariablesGlobales.stRutaDestino + VariablesGlobales.NombreDestino, **true**);
3. // Se elimina la extension del fichero para poder cambiarle el nombre
4. **string** nombreSinExtension = archivo.Name.Replace(VariablesGlobales.ExtensionOrigen, "");
5. // El archivo extraido se copia en el directorio <stRutaProcesado> con nombre "<nombreSinExtension> + <DescProcesado> + <ExtensionOrigen>"
6. File.Copy(remoto + archivo.Name, VariablesGlobales.stRutaProcesado + nombreSinExtension + VariablesGlobales.DescProcesado + VariablesGlobales.ExtensionOrigen, **true**);

Por último, se procede a la eliminación del archivo ya copiado, como previamente se ha indicado en el caso de que un archivo no pueda ser eliminado del directorio origen, guardaremos su ruta en la lista *ArchivosNoEliminados.* En cada ejecución de CopiarArchivo, se intentarán volver a eliminar y en ninguno de los casos se volverá a seleccionar alguno de los archivos de esta lista para enviar:

1. **try**
2. {
3. // El archivo extraido se elimina del servidor FTP de la ruta <stRutaOrigen>
4. ftp.DeleteFile(VariablesGlobales.stRutaOrigen + archivo.Name);
5. **if** (VariablesGlobales.bLogDetallado)
6. {
7. VariablesGlobales.Log.Info(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " INFO Se ha eliminado el archivo \"" + VariablesGlobales.stRutaOrigen + archivo.Name + "\" del servidor.");
8. }
9. }
10. **catch** (Exception ex)
11. {
12. VariablesGlobales.Log.Warn(DateTime.Now.ToString() + " SERVICIO: " + VariablesGlobales.sLogServiceName + " FATAL: No se ha podido eliminar el archivo \"" + VariablesGlobales.stRutaOrigen + archivo.Name + "\" del servidor. ERROR: " + ex.Message);
13. ArchivosNoEliminados.**Add**(archivo.Name);
14. **return** **true**;
15. }