Monitoring a remote VFS and downloading files to your local disk when they change

This article shows how to monitor a remote directory and automatically download each new file the moment it appears, using the `RemoteWatcher` object. Unlike a cron-scheduled one-shot script that polls on a fixed interval and then exits, this script runs as a persistent job and reacts in near-real-time: as soon as the next poll cycle detects a new file, your callback fires and the download begins.

How It Works

`RemoteWatcher` takes the name of a VFS profile from the VFS Library and polls the remote directory for you. You configure which events you care about, tell it which directory to watch, and hand it a callback. AFT! takes care of the polling loop, the reconnection logic, and the automatic tracking of newly created subdirectories so you never miss a file that lands in a subfolder that did not exist when the watcher started.

The callback receives an event object with two fields:

- `evt.Event`: the type of change (`EVT_CREATE`, `EVT_REMOVE`, `EVT_MODIFY`)
- `evt.Object`: the full remote path of the file that changed

For an inbound pickup scenario you only care about `EVT_CREATE`. Files that are modified in place or deleted are irrelevant.

The script also opens a separate `VirtualFSByName` handle on the same profile. The watcher uses it internally for polling; this second handle is for the actual `ExportFile` call that copies the remote file to the local inbox.

```
// Remote inbound file pickup
// Monitors a named VFS for newly arrived files and downloads each one to a
// local inbound folder as it appears. Runs persistently until halted.

var remoteVfsName       = "your-sftp-vfs-name";   // named VFS profile from the VFS Library
var remoteInboxDir      = "/remote/inbox";        // remote directory to poll for new arrivals
var localInboxDir       = "/local/inbound";       // local destination for downloaded files
```

```
var pollIntervalSeconds = 30;                    // how often to poll the remote directory

// Configure which remote events to act on.
// For most inbound pickup scenarios only newly created files matter.
var watcher = new RemoteWatcher(remoteVfsName);
watcher.NotifyCreate = true;
watcher.NotifyRemove = false;
watcher.NotifyModify = false;

// Optionally restrict pickups to specific file patterns.
// watcher.InclusionFilter = ["*.csv", "*.xml"];

// Open a VFS handle for downloading each newly detected file.
// The RemoteWatcher monitors the directory internally; this separate handle
// is used for the actual ExportFile call.
var vfs = new VirtualFSByName(remoteVfsName);

try {
   watcher.WatchDir(remoteInboxDir, true);   // true = recurse into subdirectories

   watcher.Start(pollIntervalSeconds, function(evt) {
      if (evt.Event !== EVT_CREATE) {
         return;
      }

      // evt.Object is the full remote path of the newly detected file.
      var remotePath = evt.Object;
      var fileName   = ExtractName(remotePath) + ExtractExt(remotePath);
      var localPath  = localInboxDir + "/" + fileName;

      Log.Info("new remote file detected: " + remotePath);

      var r = vfs.ExportFile(remotePath, localPath);
      if (!r.Ok()) {
         Log.Error("download failed for " + remotePath + ": " + r.ErrorMsg());
      } else {
         Log.Info("downloaded to: " + localPath);
      }
   });
```

```
    // Block here until an operator sends a halt signal via the AFT! web UI or API.
    WaitForHaltSignal();
    Log.Info("halt signal received, shutting down");
} finally {
    // Always stop the watcher on exit, whether normal or due to an error.
    watcher.Stop();
}
```

A few things worth noting.

`watcher.WatchDir(remoteInboxDir, true)`: the second argument enables **recursive watching**. If the partner drops files into subdirectories, they will be picked up automatically. AFT!'s `RemoteWatcher` also tracks newly created subdirectories at runtime, so folders that did not exist at startup are covered without any extra logic on your part.

`watcher.Start(pollIntervalSeconds, callback)`: the first argument is the polling interval in seconds. Unlike `FsWatcher` (which receives OS-level push notifications), a remote watcher can only detect changes by comparing successive directory listings, so it must poll. Tune the interval to balance freshness against load on the remote server. Thirty seconds is a reasonable default for most trading partner scenarios.

`InclusionFilter`: if you only want specific file types, uncomment the filter line and list the patterns. Files that do not match are silently ignored before the callback is ever invoked.

The `try/finally` block ensures that `watcher.Stop()` is always called even if an exception is thrown, releasing the goroutine that drives the polling loop.

Running this script

Because the script calls `WaitForHaltSignal()`, it runs as a persistent job rather than a one-shot execution. Trigger it once from the **Scripts** page in the Admin UI (or via API using the `POST /v1/adm/jobs` endpoint) and leave it running. When you need to stop it, use the halt button in the UI or call the `DELETE /v1/adm/jobs/{id}` API.

Revision #1
Created 12 April 2026 10:48:06 by DevTeam
Updated 12 April 2026 11:20:55 by DevTeam