

FileInterface

FileSystemWatcher + CronScheduler

Installations-, Konfigurations- und Betriebsanleitung

Projekt	FileinterfaceWatcherCronSchedulerProject
Zielpattform	.NET Framework 4.8, x86
IMSAPI	IMSApiDotNet.dll 10.0.0-3
Webclient	HTTP Administration mit Basic Authentication
Stand	19.08.2026

Standardzugang Webclient

Benutzer: admin | Kennwort: admin. Die Werte sind in App.config konfigurierbar und sollten vor dem produktiven Einsatz geändert werden.

Diese Anleitung bezieht sich auf das Archiv
FileinterfaceWatcherCronSchedulerProject-COMPLETE.zip

Inhalt

1. Zweck und Funktionsumfang
2. Architektur und Ablauf
3. Voraussetzungen
4. Projektstruktur
5. Erste Inbetriebnahme in Visual Studio
6. App.config im Detail
7. IMSAPI Initialisierung und Login
8. HTTP-Webclient: lokal und im Netzwerk
9. FileSystemWatcher konfigurieren
10. CronScheduler konfigurieren
11. inArgs und Platzhalter
12. Dateinhalt aufbereiten
13. Datei nach erfolgreicher Verarbeitung verschieben
14. Parallelität und Schutz vor Doppelverarbeitung
15. Praxisbeispiele
16. Persistente Konfiguration und Sicherung
17. Windows-Dienst installieren und betreiben
18. Update und Deployment
19. Runtime-Log und Diagnose
20. Fehlerbehebung
21. Sicherheit und Produktionsbetrieb
22. Bekannte Implementierungsdetails und Grenzen
23. Go-Live-Checkliste
- A. HTTP-API-Endpunkte
- B. App.config Referenz

1. Zweck und Funktionsumfang

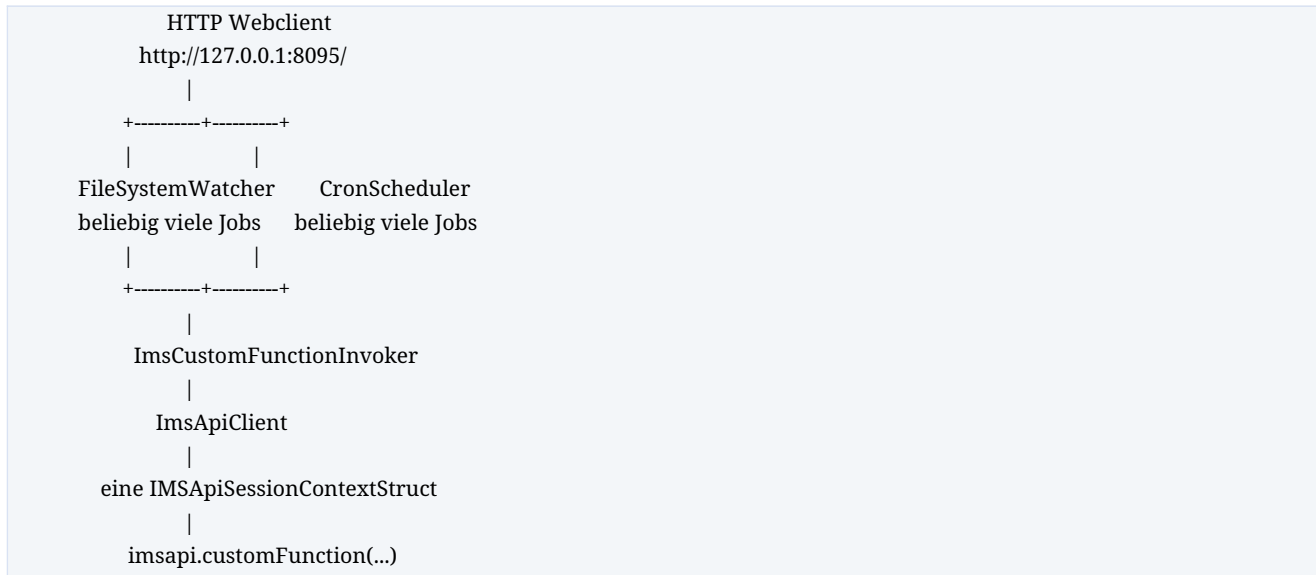
Das Projekt kombiniert zwei voneinander unabhängige Mechanismen zur Verarbeitung von Dateien und zum Aufruf von iTAC-IMS-CustomFunctions:

- FileSystemWatcher: reagiert ereignisbasiert auf neu angelegte, geänderte oder umbenannte Dateien.
- CronScheduler: startet frei konfigurierbare Jobs nach einem 5-Feld-Cron-Ausdruck.
- HTTP-Webclient: verwaltet Watcher und Cronjobs zur Laufzeit.
- IMSAPI: eine gemeinsame Anmeldung wird beim Programmstart aufgebaut und von allen Jobs verwendet.
- Beliebig viele inArgs: jeder Watcher und jeder Cronjob besitzt seine eigene dynamische Argumentliste.
- Optionale Dateiverschiebung: Dateien können ausschließlich nach erfolgreichem CustomFunction-Aufruf in ein Zielverzeichnis verschoben werden.

Wichtig

FileSystemWatcher und CronScheduler laufen gleichzeitig. Der Scheduler ersetzt den Event-Handler nicht.

2. Architektur und Ablauf



Beim Programmstart wird zuerst die IMSAPI initialisiert und regLogin() ausgeführt. Erst wenn dieser Schritt erfolgreich ist, werden CronScheduler, FileSystemWatcher und HTTP-Webclient gestartet.

Folge bei Loginfehler

Wenn IMS regLogin() fehlschlägt, ist auch der HTTP-Webclient nicht erreichbar. In diesem Fall zuerst die IMS-Konfiguration bzw. die Konsolen-/Windows-Dienst-Fehlermeldung prüfen.

3. Voraussetzungen

Komponente	Anforderung
Betriebssystem	Windows; für den Windows-Dienst sind Administratorrechte zur Installation erforderlich.
Visual Studio	.NET-Framework-fähige Visual-Studio-Installation, z. B. mit Desktopentwicklung für .NET.
.NET Framework	Target: .NET Framework 4.8.
Plattform	x86 / 32 Bit. Das Projekt setzt PlatformTarget=x86.
IMSAPI	IMSApiDotNet.dll ist im Projekt unter lib\IMSApiDotNet.dll enthalten.
Netzwerk	Erreichbarkeit des konfigurierten IMS-Servers; für entfernten Webzugriff zusätzlich TCP-Port des Webclients.
Dateisystem	Dienstkonto benötigt Lesen im Quellverzeichnis und bei Verschiebung Schreiben/Löschen in Quelle und Ziel.

4. Projektstruktur

```
FileInterfaceWatcherCronSchedulerProject\
|-- FileInterfaceCronScheduler.sln
|-- README.md
|-- BUILD_INFO.txt
|-- scripts\
|   |-- install-service.bat
|   |-- uninstall-service.bat
|   `-- allow-webinterface-network.bat
`-- FileInterfaceCronScheduler\
    |-- App.config
    |-- Program.cs
    |-- RuntimeFactory.cs
    |-- FileInterfaceCronSchedulerService.cs
    |-- ImsCustomFunctionInvoker.cs
    |-- IMSApi\
    |   `-- ImsApiClient.cs
    |-- lib\
    |   `-- IMSApiDotNet.dll
    |-- cronjobs.json
    |-- watcherjobs.json
    |-- ihas.properties
    `-- Scheduler\
        |-- CronScheduler.cs
        |-- FileSystemWatcherManager.cs
        |-- WebAdminServer.cs
        |-- CronJobExecutor.cs
        |-- WatcherJobExecutor.cs
        `-- ...
```

Beim Build werden cronjobs.json, watcherjobs.json und ihas.properties in das Ausgabeverzeichnis kopiert. Die App.config wird dort als FileInterfaceCronScheduler.exe.config abgelegt.

5. Erste Inbetriebnahme in Visual Studio

1. ZIP-Archiv in ein dauerhaftes Arbeitsverzeichnis entpacken.
2. FileInterfaceCronScheduler.sln in Visual Studio öffnen.
3. App.config öffnen und mindestens IMS.ServerUrl sowie IMS.StationNumber auf die Zielumgebung anpassen.
4. Prüfen, ob IMS.ClientNumber und IMS.RegistrationType zur Umgebung passen.
5. Konfiguration Debug wählen und das Projekt starten.
6. Bei erfolgreichem IMS-Login erscheint in der Konsole die Meldung, dass FileSystemWatcher, CronScheduler und Webinterface gestartet wurden.
7. Im Browser <http://127.0.0.1:8095/> öffnen.
8. Mit admin / admin anmelden.
9. Zunächst einen deaktivierten Test-Watcher oder Test-Cronjob anlegen, speichern, Eingaben prüfen und erst danach aktivieren.

Empfohlener Ersttest

Vor der Installation als Windows-Dienst immer zuerst im Konsolen-/Debug-Modus testen. So sind IMS-Login- und Konfigurationsfehler unmittelbar sichtbar.

6. App.config im Detail

```
<appSettings>
  <add key="IMS.AppID" value="FileInterfaceCronScheduler" />
  <add key="IMS.ServerUrl" value="http://YOUR-IMS-SERVER:PORT" />
  <add key="IMS.StationNumber" value="YOUR_STATION" />
  <add key="IMS.ClientNumber" value="1" />
  <add key="IMS.RegistrationType" value="S" />
  <add key="IMS.PropertyDirectory" value="." />
  <add key="IMS.SerializeCalls" value="false" />

  <add key="Scheduler.ConfigFile" value="cronjobs.json" />
  <add key="Watcher.ConfigFile" value="watcherjobs.json" />
  <add key="Scheduler.WebPrefix" value="http://127.0.0.1:8095/" />

  <add key="Scheduler.WebUser" value="admin" />
  <add key="Scheduler.WebPassword" value="admin" />
</appSettings>
```

Schlüssel	Bedeutung	Hinweis
IMS.AppID	Application-ID für IMSApiDotNet.	Pflichtwert.
IMS.ServerUrl	IMS-Server / Clusterknoten.	Pflichtwert; wird als itac.artes.clusternodes gesetzt.
IMS.StationNumber	Station für regLogin().	Pflichtwert.
IMS.ClientNumber	IMS Client.	Pflichtwert; Legacy-Namen ClientNO/CleintNO werden im Code ebenfalls unterstützt.
IMS.RegistrationType	Registrierungstyp.	Standard S.
IMS.PropertyDirectory	Verzeichnis für ihas.properties.	Relative Pfade werden relativ zum EXE-Verzeichnis aufgelöst.
IMS.SerializeCalls	Globales Serialisieren der CustomFunction-Aufrufe.	false = parallele Aufrufe; true = jeweils nur ein IMS-Aufruf gleichzeitig.
Scheduler.ConfigFile	JSON-Datei der Cronjobs.	Standard cronjobs.json.
Watcher.ConfigFile	JSON-Datei der Watcher.	Standard watcherjobs.json.
Scheduler.WebPrefix	HttpListener-Prefix.	Lokal: http://127.0.0.1:8095/; Netzwerk z. B. http://+:8095/.
Scheduler.WebUser	Basic-Auth Benutzer.	Standard admin. Leer = Authentifizierung wird deaktiviert.
Scheduler.WebPassword	Basic-Auth Kennwort.	Standard admin.

Nach dem Build

Im Ausgabeverzeichnis zählt FileInterfaceCronScheduler.exe.config. Wird dort direkt deployed, müssen produktive Änderungen in dieser Datei vorgenommen werden oder die geänderte App.config muss neu gebaut/kopiert werden.

7. IMSAPI Initialisierung und Login

Die Klasse IMSApi/ImsApiClient.cs führt beim Start exakt folgende Schritte aus:

- Konfigurationswerte lesen.
- PropertyDirectory auflösen und bei Bedarf anlegen.
- ihas.properties sicherstellen.
- IMSApiDotNet.setProperty("itac.appid", IMS.AppID).
- IMSApiDotNet.setProperty("itac.artes.clusternodes", IMS.ServerUrl).
- IMSApiDotNet.setProperty("itac.propdir", IMS.PropertyDirectory).
- IMSApiDotNet.loadLibrary().
- imsapiGetLibraryVersion().
- imsapiInit().
- IMSApiSessionValidationStruct mit Station, Client und RegistrationType aufbauen.
- regLogin() aufrufen und IMSApiSessionContextStruct speichern.

```
var validation = new IMSApiSessionValidationStruct
```

```
{
    stationNumber = station,
    client = client,
    registrationType = registrationType
};

IMSApiSessionContextStruct newSessionContext;
int loginResult = _imsapi.regLogin(validation, out newSessionContext);
```

Alle Watcher und Cronjobs verwenden anschließend dieselbe SessionContext-Instanz für customFunction(...).

Logout

Im aktuellen Projekt ist kein expliziter regLogout()/imsapiFinish()-Aufruf implementiert. Beim Beenden werden die verwalteten Referenzen freigegeben. Falls eure IMSAPI-Version zwingend einen expliziten Logout verlangt, sollte dieser in ImsApiClient.Dispose() ergänzt werden.

8. HTTP-Webclient: lokal und im Netzwerk

8.1 Lokaler Zugriff

Standardmäßig bindet der Webclient nur an die Loopback-Adresse:

```
http://127.0.0.1:8095/
```

Auf demselben Rechner im Browser öffnen und mit den Werten aus Scheduler.WebUser / Scheduler.WebPassword anmelden. Standard: admin / admin.

8.2 Zugriff aus dem Netzwerk

Für einen Zugriff von anderen Rechnern muss der Prefix geändert werden:

```
<add key="Scheduler.WebPrefix" value="http://+:8095/" />
```

Danach scripts\allow-webinterface-network.bat als Administrator ausführen. Das Skript richtet URLACL und Windows-Firewall-Regel ein:

```
netsh http add urlacl url=http://+:8095/ user=Everyone
netsh advfirewall firewall add rule name="FileInterface HTTP Admin 8095" ^
dir=in action=allow protocol=TCP localport=8095
```

Sicherheit

HTTP Basic Authentication verschlüsselt Benutzername und Kennwort nicht. Bei Netzwerkzugriff nur in einem geschützten internen Netz, über VPN oder hinter einem HTTPS-Reverse-Proxy verwenden. Das Standardkennwort admin muss produktiv geändert werden.

Nach Änderungen an Scheduler.WebPrefix, WebUser oder WebPassword ist ein Neustart des Programms bzw. Windows-Dienstes erforderlich.

9. FileSystemWatcher konfigurieren

Der Reiter FileSystemWatcher verwaltet beliebig viele voneinander unabhängige Watcher. Nach Speichern oder Löschen wird der WatcherManager automatisch neu geladen; ein Dienstneustart ist nicht erforderlich.

Feld	Funktion	Empfehlung
Name	Anzeigenname des Watchers.	Eindeutigen fachlichen Namen vergeben.
Aktiv	Schaltet Watcher ein/aus.	Neue Jobs zuerst deaktiviert testen.
Quellverzeichnis	Zu überwachender Ordner.	Absoluten Pfad verwenden.
Dateifilter	FileSystemWatcher SearchPattern, z. B. *.xml.	So eng wie möglich filtern.

Unterverzeichnisse	Überwacht rekursiv.	Nur aktivieren, wenn benötigt.
Created	Reaktion auf neu erzeugte Dateien.	Für klassische Eingangsschnittstellen meist aktiv.
Changed	Reaktion auf Dateiveränderungen.	Kann viele Events erzeugen; Debounce beachten.
Renamed	Reaktion auf Umbenennungen.	Wichtig, wenn Produzent per temp-Datei + Rename arbeitet.
Debounce (ms)	Unterdrückt wiederholte Events für gleiche Job/Datei-Kombination.	Standard 1000 ms.
File-Ready Versuche	Anzahl Prüfungen, bis Datei als stabil/readable gilt.	Standard 10.
Abstand (ms)	Wartezeit zwischen File-Ready-Prüfungen.	Standard 250 ms.
Leere Zeilen entfernen	Erzeugt {{CONTENT}} aus nicht-leeren, getrimmten Zeilen.	RAW_CONTENT bleibt unverändert.
Zeilentrenner	Trenner beim Zusammenfügen der Zeilen.	Standard ;
CustomFunction	Name der IMS CustomFunction.	Pflichtfeld.
inArgs	Dynamische Argumentliste.	Reihenfolge entspricht inArgs[0], inArgs[1], ...
Datei verschieben	Nur nach erfolgreichem IMS-Aufruf.	Für Einmalverarbeitung empfohlen.
Zielverzeichnis	Archiv-/Done-Verzeichnis.	Pflicht bei aktivem Verschieben.
Zieldatei überschreiben	Bestehende Datei ersetzen.	Standard aus; dann wird Zeitstempel angehängt.

9.1 Event-Verarbeitung

```

Dateisystem-Event
|
+-- Zielordner ignorieren, falls Datei bereits im MoveTarget liegt
|
+-- Debounce nach Watcher-ID + Dateipfad
|
+-- Schutz: dieselbe Datei im selben Watcher nicht parallel
|
+-- File-Ready-Prüfung
|
+-- Datei lesen
|
+-- Platzhalter in inArgs expandieren
|
+-- imsapi.customFunction(...)
|
+-- Erfolg? -> optional Datei verschieben

```

Der Watcher verwendet FileShare.ReadWrite und prüft zusätzlich Dateigröße und LastWriteTime. Erst wenn die Datei in zwei aufeinanderfolgenden Prüfungen stabil ist, wird sie gelesen.

Der Button Vorhandene Dateien verarbeiten führt einen manuellen Scan des Quellverzeichnisses aus. Dabei wird als {{EVENT}} der Wert ManualScan verwendet.

10. CronScheduler konfigurieren

Der CronScheduler prüft alle 5 Sekunden, welche aktivierten Jobs in der aktuellen lokalen Minute fällig sind. Derselbe Cronjob wird innerhalb derselben Minute nur einmal gestartet.

Feld	Funktion
Name	Eindeutiger Anzeigename.
Cron (5 Felder)	minute hour day-of-month month day-of-week.
Aktiv	Job wird nur bei aktivem Haken automatisch gestartet.
CustomFunction einmal ohne Datei aufrufen	Kein Dateiscan; CustomFunction genau einmal pro Ausführung.
Quellverzeichnis	Bei Datei-Jobs Pflicht.
Dateifilter	z. B. *.xml.
Unterverzeichnisse	Rekursiver Dateiscan.
Leere Zeilen entfernen / Zeilentrenner	Erzeugt {{CONTENT}} wie beim Watcher.
CustomFunction	Pflichtfeld.
inArgs	Beliebig viele Argumente, inklusive Platzhalter.

Datei verschieben	Nach erfolgreichem Aufruf pro Datei.
Zielverzeichnis / überschreiben	Verhalten identisch zum Watcher.
Jetzt ausführen	Startet den gespeicherten Job sofort, unabhängig vom Cron-Ausdruck.

10.1 Cron-Syntax

Unterstützt werden *, Schrittwerte, Listen und Bereiche. Tag der Woche: 0 oder 7 = Sonntag.

Ausdruck	Bedeutung
* * * * *	jede Minute
*/5 * * * *	alle 5 Minuten
0 */2 * * *	alle 2 Stunden zur Minute 0
15 2 * * *	täglich um 02:15
0 6 * * 1-5	Montag bis Freitag um 06:00
0,15,30,45 * * * *	jede Viertelstunde
0 8-18/2 * * *	zwischen 08:00 und 18:00 alle 2 Stunden

Cron-Zeitbasis

CronExpression.IsMatch() verwendet DateTime.Now. Damit gilt die lokale Zeitzone des Windows-Rechners bzw. Servers.

Keine Überlappung desselben Cronjobs

Läuft ein Cronjob noch, wird ein weiterer Start desselben Jobs übersprungen und im Log als 'still running' protokolliert. Unterschiedliche Cronjobs können parallel laufen.

Cron-Dateien während des Schreibens

Der CronExecutor besitzt keine File-Ready-Stabilitätsprüfung wie der FileSystemWatcher. Wenn Produzenten Dateien langsam schreiben, ist der Watcher oder ein atomarer Übergang (z. B. Temp-Datei + Rename) robuster.

11. inArgs und Platzhalter

Für jeden Watcher und Cronjob können über + inArg beliebig viele Argumente in definierter Reihenfolge angelegt werden. Jeder Eintrag wird vor dem IMS-Aufruf als String expandiert.

Platzhalter	Watcher	Cron	Inhalt
{{CONTENT}}	Ja	Ja	Aufbereiteter Dateiinhalt.
{{RAW_CONTENT}}	Ja	Ja	Originaler Dateiinhalt.
{{FILE_NAME}}	Ja	Ja	Nur Dateiname, z. B. test.xml.
{{FILE_PATH}}	Ja	Ja	Vollständiger Dateipfad.
{{FILE_DIR}}	Ja	Ja	Verzeichnis der Datei.
{{FILE_EXT}}	Ja	Ja	Dateiendung inkl. Punkt, z. B. .xml.
{{JOB_NAME}}	Ja	Ja	Name des Watchers/Cronjobs.
{{NOW}}	Ja	Ja	Lokaler Timestamp im Format yyyy-MM-ddTHH:mm:ss.fffK.
{{EVENT}}	Ja	Nein	Created, Changed, Renamed oder ManualScan.

CustomFunction: BroseREST.restCallCogi1

```
inArgs[0] = RPCServices-WS
inArgs[1] = {{CONTENT}}
inArgs[2] = {{FILE_NAME}}
inArgs[3] = {{EVENT}} // nur Watcher
```

Bei einem Cronjob mit CustomFunction einmal ohne Datei aufrufen sind Datei-Platzhalter leer. {{JOB_NAME}} und {{NOW}} funktionieren weiterhin.

12. Dateiinhalt aufbereiten

Die Datei wird mit StreamReader unter Verwendung von UTF-8 als Standard und BOM-Erkennung gelesen.

12.1 Leere Zeilen entfernen = aus

{{CONTENT}} entspricht dann dem gelesenen Text. {{RAW_CONTENT}} ist ebenfalls der gelesene Originaltext.

12.2 Leere Zeilen entfernen = an

Alle Zeilen werden normalisiert, leere/Whitespace-Zeilen entfernt, die verbleibenden Zeilen getrimmt und mit dem konfigurierten Zeilentrenner verbunden.

Quelldatei:

A

B

C

Zeilentrenner = ;

{{RAW_CONTENT}} = "A\r\nB\r\n\r\nC"

{{CONTENT}} = "A;B;C"

13. Datei nach erfolgreicher Verarbeitung verschieben

Die Datei wird nur verschoben, wenn die IMS-CustomFunction RES_OK zurückliefert. Bei Fehler bleibt die Datei im Quellverzeichnis.

Situation	Verhalten
Zieldatei existiert nicht	Datei wird unter gleichem Namen verschoben.
Zieldatei existiert, Überschreiben = Ja	Vorhandene Zieldatei wird gelöscht, danach Datei verschoben.
Zieldatei existiert, Überschreiben = Nein	Zielname erhält Zeitstempel yyyyMMdd_HHmms_fff; bei weiterer Kollision zusätzlich laufende Nummer.
Zielverzeichnis fehlt	Directory.CreateDirectory() legt es automatisch an.
CustomFunction liefert Fehler	Keine Verschiebung.

Watcher und Archiv unterhalb der Quelle

Der FileSystemWatcher ignoriert Events, deren Pfad innerhalb des konfigurierten MoveTargetDirectory liegt. Dadurch wird ein Archiv-Unterverzeichnis nicht erneut verarbeitet.

MoveFileAfterSuccess = Nein

Beim Cronjob wird dieselbe Datei beim nächsten Cronlauf erneut gefunden und erneut verarbeitet. Beim Watcher erfolgt eine erneute Verarbeitung nur bei einem neuen passenden Dateisystem-Event oder über den manuellen Scan.

14. Parallelität und Schutz vor Doppelverarbeitung

Ebene	Verhalten
Unterschiedliche Cronjobs	Können parallel ausgeführt werden.
Derselbe Cronjob	Kein paralleler Doppelstart; zweiter Start wird übersprungen.
Unterschiedliche Watcher	Können parallel arbeiten.
Ein Watcher, unterschiedliche Dateien	Können parallel verarbeitet werden.
Ein Watcher, dieselbe Datei	runningFiles verhindert parallele Verarbeitung derselben Job/Datei-Kombination.
Watcher Event-Sturm	DebounceMilliseconds unterdrückt Folgeevents innerhalb des Zeitfensters.
IMS.SerializeCalls=false	CustomFunction-Aufrufe dürfen parallel in die IMSAPI gehen.
IMS.SerializeCalls=true	Alle IMS-CustomFunction-Aufrufe werden global serialisiert.

Wenn die konkrete IMSAPI-/Serverumgebung parallele customFunction-Aufrufe mit derselben Session nicht zuverlässig unterstützt, IMS.SerializeCalls auf true setzen. Die Scheduler bleiben unabhängig, nur der eigentliche IMS-Aufruf wird dann nacheinander ausgeführt.

15. Praxisbeispiele

15.1 Watcher für XML-Eingang

Feld	Beispiel
Name	XML Eingang COGI
Aktiv	Ja
Quellverzeichnis	C:\FileInterface\Input
Dateifilter	*.xml
Created / Changed / Renamed	Ja / Nein / Ja
Debounce	1000 ms
File-Ready	10 Versuche, 250 ms
CustomFunction	BroseREST.restCallCogi1
inArgs[0]	RPCServices-WS
inArgs[1]	{{CONTENT}}
inArgs[2]	{{FILE_NAME}}
inArgs[3]	{{EVENT}}
Verschieben	Ja
Ziel	C:\FileInterface\Archive

Created + Renamed ist bei vielen Schnittstellen ausreichend und reduziert doppelte Changed-Events. Ob Changed benötigt wird, hängt vom Schreibverhalten des Produzenten ab.

15.2 Cronjob für Datei-Import alle 5 Minuten

Feld	Beispiel
Name	COGI Polling
Cron	* / 5 * * * *
CustomFunction ohne Datei	Nein
Quelle	C:\FileInterface\Polling
Filter	*.xml
CustomFunction	BroseREST.restCallCogi1
inArgs	RPCServices-WS {{CONTENT}} {{FILE_NAME}}
Verschieben	Ja
Ziel	C:\FileInterface\Archive

15.3 Cronjob ohne Datei

Feld	Beispiel
Name	Statusprüfung
Cron	0 * * * *
CustomFunction ohne Datei	Ja
CustomFunction	MyNamespace.checkStatus
inArgs[0]	J816352
inArgs[1]	{{NOW}}

Quellverzeichnis, Filter und Datei-Verschiebung sind bei CallOnceWithoutFile nicht relevant.

16. Persistente Konfiguration und Sicherung

Die Konfiguration wird getrennt gespeichert:

```
cronjobs.json
watcherjobs.json
```

Beim Speichern wird zunächst eine .tmp-Datei geschrieben. Existiert die Zieldatei bereits, versucht das Repository File.Replace(..., backup, true). Dadurch kann eine .bak-Datei entstehen.

Backup vor Updates

Vor einem Softwareupdate mindestens FileinterfaceCronScheduler.exe.config, cronjobs.json, watcherjobs.json und gegebenenfalls ihas.properties sichern.

Die Repository-Daten werden beim Start eingelesen. Änderungen über den Webclient werden sofort in den Dateien persistiert.

17. Windows-Dienst installieren und betreiben

17.1 Release bauen

21. In Visual Studio Konfiguration Release wählen.
22. Projekt erstellen.
23. Prüfen, dass bin\Release\FileinterfaceCronScheduler.exe vorhanden ist.
24. Prüfen, dass im Release-Ordner IMSApiDotNet.dll, cronjobs.json, watcherjobs.json, ihas.properties und FileinterfaceCronScheduler.exe.config vorhanden sind.
25. Produktive App-Konfiguration setzen.

17.2 Dienst installieren

scripts\install-service.bat als Administrator ausführen. Das Skript erstellt folgenden Dienst:

```
ServiceName : FileinterfaceCronScheduler
DisplayName : FileInterface Watcher + CronScheduler
Starttyp    : automatisch
binPath     : FileinterfaceCronScheduler.exe --service
```

Das Skript startet den Dienst direkt nach der Erstellung.

17.3 Dienst entfernen

scripts\uninstall-service.bat als Administrator ausführen. Vor Änderungen an Binärdateien den Dienst stoppen.

17.4 Dienstkonto und Rechte

- Leserechte auf alle Quellverzeichnisse.
- Schreib-/Löschrechte auf Quellverzeichnisse, wenn Dateien verschoben werden.
- Schreibrechte auf Ziel-/Archivverzeichnisse.
- Leserechte auf EXE, DLL und Konfiguration.
- Schreibrechte auf IMS.PropertyDirectory, falls ihas.properties erzeugt/aktualisiert werden muss.
- Bei UNC-Pfaden ein Dienstkonto mit passenden Netzwerkrechten verwenden.

18. Update und Deployment

26. Aktuelle Konfiguration sichern.
27. Windows-Dienst stoppen.
28. Neue Release-Dateien in ein separates Verzeichnis kopieren oder bestehende Programmdateien ersetzen.
29. FileinterfaceCronScheduler.exe.config nicht versehentlich mit Platzhalterwerten überschreiben.
30. cronjobs.json und watcherjobs.json erhalten bzw. zurückkopieren.
31. IMSApiDotNet.dll-Version nur gezielt austauschen.
32. Dienst starten.
33. Webclient öffnen, Status prüfen und Log kontrollieren.
34. Je einen Test-Watcher und Test-Cronjob ausführen.

Empfehlung

Produktive Konfiguration und Programm-binärdateien getrennt versionieren/sichern. Besonders bei Copy-Deployments ist die EXE-Konfiguration leicht versehentlich zu überschreiben.

19. Runtime-Log und Diagnose

Unten im Webclient befindet sich Runtime / Log. Dort werden Watcher-, Cron- und Systemmeldungen angezeigt.

Eigenschaft	Aktueller Stand
Speicherort	Nur Arbeitsspeicher.
Kapazität	maximal 5000 Einträge.
Anzeige Webclient	GET /api/logs?limit=400 beim Aktualisieren.
Reihenfolge	Neueste Einträge zuerst.
Persistenz	Nach Programm-/Dienstneustart leer.
Typen	WATCHER, CRON, SYSTEM; Level INFO oder ERROR.

Produktionshinweis

Für revisionssichere oder langfristige Diagnose sollte zusätzlich eine persistente Protokollierung (Datei, Windows Event Log, zentrale Logplattform o. Ä.) ergänzt werden.

Der Statusbereich zeigt die IDs der aktuell aktiven Watcher und laufenden Cronjobs. Im Webclient wird daraus die Anzahl angezeigt.

20. Fehlerbehebung

Problem	Prüfung / Lösung
Webclient nicht erreichbar, auch lokal	Prüfen, ob IMS-Login erfolgreich war. Der Webserver startet erst danach. Danach Port 8095, Prozessstatus und Scheduler.WebPrefix prüfen.
HTTP 401 / Loginfenster wiederholt	Scheduler.WebUser/WebPassword prüfen. Bei Änderung Dienst neu starten. Browser-Zugangsdaten ggf. löschen und neu eingeben.
Netzwerkzugriff funktioniert lokal, extern nicht	Scheduler.WebPrefix auf http://+:8095/ setzen, URLACL/Firewall-Skript als Administrator ausführen, Netzwerkfirewall prüfen.
Dienst startet nicht	Windows-Ereignisanzeige prüfen; im Konsolenmodus starten, um IMS-Fehler direkt zu sehen.
IMS regLogin failed	ServerUrl, StationNumber, ClientNumber, RegistrationType sowie Servererreichbarkeit prüfen.
IMSApiDotNet.loadLibrary() returned null	DLL/Abhängigkeiten, x86-Plattform und Deployment-Dateien prüfen.
Watcher wird nicht aktiv	Quellverzeichnis muss beim Reload existieren. Im Runtime-Log steht sonst 'Watcher source directory does not exist'.
Watcher verarbeitet Datei mehrfach	Changed deaktivieren, Debounce erhöhen, Produzentenverhalten prüfen; besser temp + Rename.
Watcher verarbeitet Datei gar nicht	Eventtypen und Filter prüfen; Dateirechte; über 'Vorhandene Dateien verarbeiten' testen.
File is not readable after retries	FileReadyRetries/Delay erhöhen oder Produzent so ändern, dass Dateien atomar bereitgestellt werden.
Cronjob startet nicht	Aktiv-Haken, 5-Feld-Cron, lokale Serverzeit und Runtime-Log prüfen.
Cronjob wird übersprungen	Derselbe Job läuft noch; Log enthält 'still running'.
Datei wird nach Erfolg nicht verschoben	Move-Haken, Zielpfad und Rechte prüfen. Bei IMS-Fehler wird bewusst nicht verschoben.
Datei wird immer wieder verarbeitet	Bei Cronjob MoveFileAfterSuccess aktivieren oder Eingangsdatei anderweitig aus Quelle entfernen.
CustomFunction bekommt falschen Inhalt	{{RAW_CONTENT}} vs. {{CONTENT}}, RemoveEmptyLines und JoinSeparator prüfen.
Parallelitätsprobleme in IMS	IMS.SerializeCalls=true setzen und Dienst neu starten.

21. Sicherheit und Produktionsbetrieb

- Standardkennwort admin unmittelbar ändern.
- Webclient möglichst nur lokal binden, wenn keine Fernadministration benötigt wird.
- Bei Fernzugriff keine ungeschützten Netze verwenden; HTTP Basic Auth bietet keine Transportverschlüsselung.
- Firewallzugriff auf bekannte Administrationsnetze begrenzen.
- Dienstkonto nach Least-Privilege-Prinzip konfigurieren.
- Quell- und Zielverzeichnisse so berechtigen, dass andere Benutzer keine unerwünschten Dateien einschleusen können.
- CustomFunction-Namen und inArgs nur vertrauenswürdigen Administratoren zugänglich machen.
- Konfigurationsdateien sichern und Änderungen dokumentieren.

WebUser leer

Im Code bedeutet ein leerer Scheduler.WebUser: Authentifizierung ist komplett deaktiviert. Das sollte produktiv nicht verwendet werden.

22. Bekannte Implementierungsdetails und Grenzen

Thema	Aktuelles Verhalten
Webclient-Start	Erst nach erfolgreichem IMS Connect/Login.
HTTP	HttpListener, kein integriertes HTTPS.
Authentifizierung	HTTP Basic Authentication; exakter Stringvergleich.
Cron-Auflösung	Minutengenau; Prüfung alle 5 Sekunden.
Cron-Felder	Genau 5 Felder; keine Sekunden.
Cron Tag der Woche	0 und 7 = Sonntag.
Cron Tag/Monat Logik	Alle fünf Felder müssen gleichzeitig matchen.
Watcher Events	Created, Changed, Renamed; keine Deleted-Verarbeitung.
Watcher File-Ready	Stabilitätsprüfung vorhanden.
Cron File-Ready	Keine Stabilitätsprüfung.
Dateien	Textverarbeitung über StreamReader; Standard UTF-8 mit BOM-Erkennung.
Log	Nur RAM, max. 5000 Einträge.
IMS-Session	Eine Session für alle Jobs.
Logout	Kein expliziter regLogout()/imsapiFinish() implementiert.
Konfigurationsänderung Watcher	Speichern/Löschen löst sofort kompletten Watcher-Reload aus.
Konfigurationsänderung Cron	Repository wird vom Scheduler bei jeder Prüfung neu abgefragt; kein Neustart nötig.

23. Go-Live-Checkliste

- ☐ Produktive IMS.ServerUrl gesetzt.
- ☐ Produktive IMS.StationNumber und IMS.ClientNumber gesetzt.
- ☐ IMS-Login im Konsolenmodus erfolgreich getestet.
- ☐ Scheduler.WebUser und Scheduler.WebPassword von admin/admin geändert.
- ☐ Entschieden, ob Webclient nur lokal oder im Netzwerk erreichbar sein soll.
- ☐ Bei Netzwerkzugriff URLACL/Firewall eingerichtet und Netzschutz geprüft.
- ☐ Alle Watcher-Verzeichnisse existieren.
- ☐ Dienstkonto besitzt notwendige Datei- und Netzwerkrechte.
- ☐ Watcher-Events und Debounce pro Schnittstelle getestet.
- ☐ File-Ready-Werte mit realem Produzenten getestet.
- ☐ Cron-Ausdrücke gegen lokale Serverzeit geprüft.
- ☐ Für wiederkehrende Datei-Cronjobs Verschiebestrategie festgelegt.
- ☐ inArgs-Reihenfolge mit der jeweiligen CustomFunction verifiziert.
- ☐ Fehlerfall getestet: IMS-Fehler darf Datei nicht verschieben.
- ☐ Erfolgsfall getestet: Datei wird korrekt archiviert.
- ☐ Parallelität getestet; bei Bedarf IMS.SerializeCalls=true.
- ☐ cronjobs.json, watcherjobs.json und exe.config gesichert.
- ☐ Windows-Dienst auf automatischen Start geprüft.
- ☐ Runtime-Log nach Neustart kontrolliert.

Anhang A. HTTP-API-Endpunkte

Der Webclient verwendet dieselben Endpunkte. Alle Endpunkte sind durch dieselbe Basic-Authentifizierung geschützt, solange Scheduler.WebUser nicht leer ist.

Methode	Pfad	Funktion
GET	/	HTML-Webclient
GET	/api/cronjobs	Cronjobs lesen
POST	/api/cronjobs/save	Cronjob speichern
POST	/api/cronjobs/delete?id=...	Cronjob löschen
POST	/api/cronjobs/run?id=...	Cronjob sofort starten
GET	/api/watchers	Watcher lesen
POST	/api/watchers/save	Watcher speichern und Watcher neu laden
POST	/api/watchers/delete?id=...	Watcher löschen und neu laden
POST	/api/watchers/reload	Alle Watcher neu aufbauen
POST	/api/watchers/scan?id=...	Vorhandene Dateien eines Watchers verarbeiten
GET	/api/logs?limit=400	Runtime-Log abrufen
GET	/api/status	Laufende Cronjobs, aktive Watcher, Serverzeit

Anhang B. App.config Referenz

Key	Default im Projekt	Neustart nötig?
IMS.AppID	FileInterfaceCronScheduler	Ja
IMS.ServerUrl	http://YOUR-IMS-SERVER:PORT	Ja
IMS.StationNumber	YOUR_STATION	Ja
IMS.ClientNumber	1	Ja
IMS.RegistrationType	S	Ja
IMS.PropertyDirectory	.	Ja
IMS.SerializeCalls	false	Ja
Scheduler.ConfigFile	cronjobs.json	Ja
Watcher.ConfigFile	watcherjobs.json	Ja
Scheduler.WebPrefix	http://127.0.0.1:8095/	Ja
Scheduler.WebUser	admin	Ja
Scheduler.WebPassword	admin	Ja

Beispiel produktionsnahe Basis

```
<add key="IMS.AppID" value="FileInterfaceCronScheduler" />
<add key="IMS.ServerUrl" value="http://ims-server:port" />
<add key="IMS.StationNumber" value="J816352" />
<add key="IMS.ClientNumber" value="1" />
<add key="IMS.RegistrationType" value="S" />
<add key="IMS.PropertyDirectory" value="." />
<add key="IMS.SerializeCalls" value="false" />

<add key="Scheduler.ConfigFile" value="cronjobs.json" />
<add key="Watcher.ConfigFile" value="watcherjobs.json" />
<add key="Scheduler.WebPrefix" value="http://127.0.0.1:8095/" />

<add key="Scheduler.WebUser" value="admin" />
<add key="Scheduler.WebPassword" value="HIER_SICHERES_KENNWORT" />
```

Hinweis zum Beispiel

Die Station J816352 ist nur ein Beispielwert. In der Zielumgebung muss die für diese Schnittstelle vorgesehene Station eingetragen werden.

Kurzfassung für die Inbetriebnahme

35. Solution öffnen und App.config mit IMS-Server/Station/Client konfigurieren.
36. Im Debug-Modus starten und erfolgreichen IMS-Login prüfen.
37. Webclient unter <http://127.0.0.1:8095/> mit admin/admin öffnen.
38. Standardkennwort ändern.
39. Watcher und Cronjobs über die beiden Webreiter anlegen.
40. Mit Testdateien und 'Jetzt ausführen' / 'Vorhandene Dateien verarbeiten' testen.
41. Release bauen.
42. Produktive exe.config und JSON-Konfiguration sichern.
43. Windows-Dienst installieren.
44. Status und Runtime-Log im Webclient prüfen.