Worum geht es?

Das *JMS SDK for ArgusTV Recorder Development* ist der Versuch, die Entwicklung von Aufzeichnungsdiensten für Argus TV soweit zu vereinfachen, dass mit wenig Aufwand ein erster Durchstich zur Nutzung eigener Softwarekomponenten zum digitalen Empfang in ArgusTV erstellt werden kann. Unabhängig von dem relativ rudimentären Status der aktuellen Version (e.g. kein Konzept für LIVE, kein Logging, keine ausreichende Stabilität gegen Fehlersituationen / Fehlerbehandlung etc.) soll das SDK im Wesentlichen die Hürde in diese Art der Softwareentwicklung verringern – vielleicht ein einfacherer Einstieg als das MediaPortal Beispiel auf GIT. Ob und in welchen Umfang sich dieser erste Schritt in ein echtes SDK entwickeln wird, wird die Zeit zeigen.

Was wird installiert?

Im Hauptverzeichnis der Installation findet man:

- *JMS.ArgusTV.dll*: die primäre Bibliothek für die Entwicklung eigener Aufzeichnungsdienste. Die Architektur des SDK sieht vor, dass man sich erst einmal nicht um die ArgusTV Infrastruktur kümmert, sondern auf Basis einer Klasse namens *RecordingDevice* eine hoffentlich einfache Komponente entwickelt, die sich ausschließlich um die Ansteuerung der Hardware zum digitalen Empfang kümmert.
- *JMS.ArgusTV.GenericService.exe*: ein Prototyp für einen Windows Dienst, der als Aufzeichnungsdienst für ArgusTV agiert. Je nach konkreten Aufrufparametern wird ein Aufzeichnungsdienst installiert, deinstalliert oder ausgeführt – auf Wunsch als eigenständiges Programm zu Testzwecken, normalerweise als Dienst. Die Installation selbst kopiert ausschließlich Dateien und installiert keine Windows Komponenten oder Dienste.
- *ArgusTV.*.dll*: die notwendigen Kommunikationsbibliotheken aus der ArgusTV 2.1 Installation.
- *Backgrounder.pdf*: dieses Dokument.

Im Unterverzeichnis *Sources* befindet sich der gesamte Quellcode des SDK – zum Selbststudium und eigener Erweiterung / Verbesserung. Neben den bereits aufgeführten Kernkomponenten des SDK ist hier auch eine auf dem DVB.NET Card Server basierende Implementierung als *Proof-Of-Concept* (und nicht mehr) zu finden.

Für Implementierungen der Hardwareanbindung wird vorgeschlagen, das Unterverzeichnis *devices* zu verwenden. Die Installation stellt hier folgende Dateien ein:

- *JMS.ArgusTV.DVBNETRecorder.dll*: der PoC auf Basis von DVB.NET.
- *DvbNet.config*: die zum Betrieb des PoC notwendige Konfigurationsdatei, mit deren Hilfe die SDK Komponenten einen Aufzeichnungsdienst ausführen können.

Was ist zu entwickeln?

Das Beispiel *Sources\JMS.ArgusTV.DVBNETRecorder* soll grob zeigen, welche Implementierungen notwendig sind, damit ein erster Durchstich erreicht werden kann:

- Die primäre Schnittstelle zum SDK bietet eine Klasse mit der Schnittstelle `IRecordingDeviceFactory`. Über diese Schnittstelle werden Geräteinstanzen erzeugt.
- Eine Geräteinstanz leitet von der Basisklasse `RecordingDevice` ab. Das SDK bietet hier zwei Varianten an, die unterschiedliche Abstraktionen verwenden. Der PoC setzt auf die höherwertige Basisklassen, bei der zwar mehr Methoden zu implementieren sind, diese sich aber quasi ausschließlich auf die Ansteuerung der Empfangshardware konzentrieren.
- Die Basisklasse verwendet dabei zwei generische Parameter. Der erste bezeichnet die Art, wie eine Quelle (aka Sender) vollständig beschrieben wird – für DVB.NET ist das eine `SourceSelection`, in der alle DVB Empfangsparameter beschrieben sind.
- Alle Aufzeichnungsoperationen sind von Natur aus asynchron. Die Idee des SDK ist, dass beim Starten jeweils ein Steuerobjekt gemeldet wird, dessen Art der zweite generische Parameter der Basisklasse ist – der DVB.NET Card Server setzt hier auf `IAsyncResult` als .NET Standardmechanismus.
- Die Basisklasse im SDK verwaltet eine Warteschlange, in der die asynchronen Operationen streng geordnet ausgeführt werden. Dazu wird auf einem separaten Thread mit Hilfe des Steuerobjektes auf den Abschluss der Operation gewartet – die Fehlerbehandlung im SDK ist hier äußerst rudimentär bis nicht vorhanden! Als Hilfestellung dazu muss in der eigenen Implementierung die Methode `TResult GetResult<TResult>` zur Verfügung gestellt werden.
- Die Methoden `ReserveDevice` und `ReleaseDevice` verwalten den exklusiven Zugriff des Aufzeichnungsdienstes auf die Empfangshardware.
- Mit `Tune` wird im Sinne des DVB Empfangs eine Quellgruppe (aka Transponder / Frequenz) angesteuert. Es ist durchaus denkbar, dass sich dieses Konzept nicht grundsätzlich auf andere Empfangssysteme übertragen lässt. Die Idee der aktuellen Implementierung ist folgende: der Empfang erfolgt über Quellgruppen. Ist einmal eine Quellgruppe angesteuert, so können beliebig viele Quellen, die dieser zugeordnet sind, gleichzeitig empfangen werden. Genauso steuert das SDK dann auch die eigene Implementierung an: bei der ersten Aufzeichnung auf einem Gerät wird dieses reserviert und eine Quellgruppe gewählt. Danach wird die Aufzeichnung von passenden Quellen aktiviert und deaktiviert. Ist keine Aufzeichnung mehr aktiv, so wird die Reservierung aufgehoben.
- Die Methoden `BeginRecording` und `EndRecording` stehen für die eigentliche Durchführung von Aufzeichnungen.
- `OnResolve` ermittelt zu einem Sendernamen eine Liste möglicher Quellen – Sender wie *Channel 4* können über verschiedene Quellgruppen aufgezeichnet werden.
- `CanRecordedInParallel` prüft, ob zwei Quellen zur gleichen Quellegruppe gehören und gleichzeitig aufgezeichnet werden können. Liefert dieser Aufruf etwa immer `false`, so kann pro Gerät immer nur eine Quelle aufgezeichnet werden.
- Mit `OnDispose` kann unsere Implementierung sich selbst ordnungsgemäß beenden.

Damit eine eigene Bibliothek auch vom SDK Windows Dienst verwendet werden kann, muss diese im Installationsverzeichnis abgelegt werden. Alternativ und vorgeschlagen kann dies auch im *devices* Unterverzeichnis erfolgen.

Was steht in der Konfiguration?

Zu jeder Implementierung für einen Aufzeichnungsdienst auf Basis des SDK gehört auch mindestens eine Konfigurationsdatei – für den DVB.NET PoC wäre das *DvbNet.Config*. Im Rahmen dieses ersten Versuchs sind dort folgende Informationen abgelegt:

- Der Name des Aufzeichnungsdienstes, wie er in ArgusTV verwendet wird – ebenso als Name des Windows Dienstes.
- Der UDP Port, an den sich der Aufzeichnungsdienst zur Kommunikation mit ArgusTV anmeldet.
- Die Namen der zu verwendenden Geräte, passend zur Implementierung. Ohne weitere Maßnahmen definiert diese Liste auch die Priorität des Gerätes, wobei ein Gerät vor einem anderen bevorzugt wird, wenn es weiter am Anfang der Liste aufgeführt ist – der DVB.NET PoC verwendet hier die eigenen Aufzeichnungsprioritäten.
- Der Vergleichsalgorithmus für Namen – DVB.NET Gerätenamen sind etwa unabhängig von Groß- und Kleinschreibung.
- Der volle Name der .NET Klasse mit der Implementierung der Klasse, über die Geräte erzeugt werden können.
- Die Liste der Aufzeichnungsverzeichnisse, jeweils als lokaler und UNC Pfad. Der lokale Pfad wird für die Ansteuerung der eigenen Geräteimplementierung verwendet, der UNC Pfad steht der ArgusTV Infrastruktur zur Verfügung. Es ist natürlich tunlichst darauf zu achten, dass diese Werte konsistent sind.

Woher kommen die kleinen Windows Dienste?

Zum Erstellen eines Windows Dienstes wird der SDK Prototypdienst *JMS.ArgusTV.GenericService.exe* mit dem vollen Pfad zu einer Konfigurationsdatei und dem zusätzlichen Parameter */install* aufgerufen – analog */uninstall* zum Entfernen des Dienstes. Der technische Name des Dienstes setzt sich aus dem Präfix *ArgusTVRecorder* gefolgt von dem Namen aus der Konfigurationsdatei zusammen – der im Windows Dienstmanager angezeigt Anzeigename beginnt mit diesem Namen aus der Konfigurationsdatei und sollte einfach wiederzufinden sein. Der Dienst wird aus Sicherheitsgründen im Startmodus *Manuell* eingerichtet und läuft unter dem Systemkonto des Rechners – beides ist bei Bedarf anzupassen.

Zum Testen kann als zweiter Parameter beim Aufruf */debug* verwendet werden. Der Aufzeichnungsdienst wird dann direkt in der Konsole gestartet und beendet sich durch Betätigen der Eingabetaste. Das ist vermutlich auch die häufigste Verwendung während der Entwicklung eigener Geräteimplementierungen.

Und sonst?

Selbstverständlich erfolgt die Nutzung des SDK vollständig auf eigene Gefahr. Für Schäden, die auch bei einer dieser Einführung entsprechenden Nutzung auftreten, wird keine Haftung übernommen. Da der gesamte Quellcode mit installiert wird, sollte es bei etwaigen echten Fehlverhalten auch kein großes Problem sein, kurzfristig eine eigene Korrektur vorzunehmen. Ich empfehle grundsätzlich, das SDK erst nach Studium der Infrastruktur und eingehendem Verständnis der Abläufe und Konzepte einzusetzen.