Windows Services

Thomas Reinwarte

Einleitung

Selbst in Zeiten von Web Services, mobilen Anwendungen und Cloud gibt es in der Software Architektur immer noch die Notwendigkeit von Windows Services am Backend. Ursprünglich mit Windows NT 4.0 eingeführt als Gegenstück zum Unix daemon, haben sie noch immer ihre Daseinsberechtigung. Im Zuge von Projekt Anforderungen stößt man als Entwickler immer wieder mal an Einschränkungen oder Grenzen bei Windows Services, die gefundenen Lösungen dafür sind hier im Folgenden zusammengefasst.

Einsatzgebiete von Services

Bei dem Service handelt es sich um einen Dienst, der im Hintergrund unter dem Account des konfigurierten Users läuft. Der Start des Services ist Automatisch–sofort oder verzögert – unmittelbar nach dem Bootvorgang des Servers. Die Möglichkeit der Startverzögerung kam erst mit einer späteren Windows Version und hatte den Sinn, das System durch massive parallele Startvorgänge von Services nicht unnötig zu belasten und Folgefehler daraus zu vermeiden. Aber selbst dieses Feature reicht nicht immer aus, ein Service immer erfolgreich zu starten – Code Sample zur Vermeidung erfolgt später.

Visual Studio Template

Das Template für ein Windows Service findet sich in Visual Studio unter C#\Windows Classic Desktop. Inhaltlich ist das Service Projekt Template das selbe wie bei .net 2.0 geblieben, lediglich die .net Version kann bis zur aktuellen Version gewählt werden. Das liegt wohl auch daran, dass vom Windows Betriebssystem her zu Windows Service keine neuen Features dazugekommen sind.

Events von Services auf die man im Code reagieren muss

Vom SCM (Service Control Manager) werden diese Events gesendet:

OnStart

Wird einmalig beim Start aufgerufen. Hier sollte man einen eigenen Timer für den eigentlichen Start des Services implementieren. Macht man das nicht, gibt es oft Probleme, das Service zu starten weil das Service nicht innerhalb des vom Windows Betriebssystems vorgesehenen Zeitraumes beim Starten reagiert. Durch den verwendeten Timer erhält das Betriebssystem sein OK, das Service wird als Started angezeigt. Den eigentlichen intensiven Prozess startet man dann erst etwa eine Minute später mit dem eigenen Timer.

OnStop

Das Beenden eines Services kann durch User Interaktion erfolgen. Wichtig ist hier, dass man an dieser Stelle ebenfalls im eigenen Service darauf regiert. In diesem Codeabschnitt beendet man seine Transaktionen oder Threads sauber, um Folgefehler beim nächsten Start seines Service zu vermeiden.

Die folgenden Events sind im Service Template nicht enthalten und muss manuell im Code hinzugefügt werden.

OnPause - OnContinue

Der Benutzer hat im Service Control GUI die Möglichkeit, das Service in einen Pause Modus zu schalten. Auf diesen Event muss man im Code reagieren, und seine im Service gestarteten Threads ebenfalls pausieren, ansonsten hat die Pause und Continue Funktion im Windows Control Center im eigenen Service keine Auswirkung.

OnShutdown

Dieser Event erfolgt beim Shutdown des Systems. Zu diesem Zeitpunkt erhält man die Chance, sein Service sauber zu beenden. Also sollte man hier das gleiche verhalten wie bei OnStop implementieren.

OnPowerEvent

Im mitgelieferten Enum `PowerBroadcastStatus` kann man hier auf den Event des Ladezustandes reagieren. Es kann sich anbieten, sein Service passend zum Ladezustand in einen Systemschonenderen Zustand zu bringen, etwa die Prozess Priority anzupassen oder sein Service wie beim OnPause Event schlafen zu legen.

OnSessionChange

Bei der Windows Terminal Server Edition erhält man damit den Change Event, Details dazu werden in Parameter `SessionChangeDescription` mitgeliefert.

EventLog

Ein Windows Service schreibt seine Zustandsänderungen, also die vorhin beschrieben erhaltet Events vom SCM, in das EventLog. Es gibt dazu das bool Property `AutoLog`, das per default auf true ist und die Start, Stop und Continue Events verwaltet. Will man seine Logik auf weitere Events ausweiten, muss man an dieser Stelle seinen Code erweitern.

Tipps zu Service Erweiterungen

Out oft the Box bietet ein .net Standard Service, das mittels Visual Studio mit dem Projekt Template erzeugt wurde, nicht alle Möglichkeiten, die in der Praxis benötigt werden, darunter fallen etwa:

Service Installer

Im Visual Studio Default Projekt Typ ist die Service Installationsmöglichkeit, also der Vorgang zur Registrierung des Services in das Betriebssystem, nicht vorhanden. Diese fügt man mittels Referenz auf *System.Configuration.Install* hinzu. ***(Siehe Code-Rahmen auf der nächsten Seite.)***

Service Multi Register

Jede Registrierung eines Windows Service bezieht sich auf eine physische Datei auf dem Dateisystem, das ist der Standard. Um jedoch seine Installation, also der physischen Datei n Services mit unterschiedlichen Namen im Service Control GUI zuzuordnen, bedarf es einer Code Erweiterung.

Erst mit dieser ist es möglich, sein Service mehrmals parallel laufen zu lassen. Ansonsten bliebe bei der Standard Implementierung nur die Alternative, das Service mehrfach am System parallel zu installieren.

Der Sinn ein Service mehrfach laufen zu lassen kann eine Ausfallsicherheit, Gründen der Performance oder Skalierbarkeit sein. Vom Betriebssystem her gibt es dazu nur etwas zur Ausfallsicherheit – der Wiederherstellung eines Services.

Damit lässt sich ein Service mittels `installutil` deinstallieren (`-u` für uninstall) bzw. registrieren (`-i` für `install`). Im Code ergänzt wurde der Parameter `servicename`, damit lässt sich ein Multiregister eines Services durchführen. ***(Siehe Rahmen unten.)***

Starten und Stoppen lassen sich Windows Services in der Systemsteuerung oder durch Cmd Line mittels *net start [serviceName]* bzw. *net stop [serviceName]* wie bisher verwenden.

Der Servicename kann übrigens maximal 80 Zeichen lang werden, wieder eine von Microsoft magischen Längenbegrenzungen. Man findet die Angabe dazu in der Basisklasse `ServiceBase`, von der jedes Service ableitet.

Mehrmalige Installation von Services bei nur einer Installation: ***(Siehe Rahmen nächste Seite oben.)***

Service Mandanten Fähigkeit

Um nun dem Multi Fach registierten Service mitzuteilen, um welchen Context (Bsp. Der Mandant) es sich bei seiner Businesslogik in seiner Instanz handelt, muss man diese Information beim Start übergeben.

Das Windows ServiceControl GUI bietet dazu einen Start Parameter.

```
cd \windows\Microsoft.Net\Framework\v4.0.303197
installutil -u /servicename="Sample Service" "c:\program files\ RIT\SampleService\SampleService.exe"
installutil -i u /servicename="Sample Service" "c:\program files\ RIT\SampleService\SampleService.exe"
```

```
cd \windows\Microsoft.Net\Framework\v4.0.30319
installutil -u /servicename="Sample Service 1" "c:\program files\ RIT\SampleService\SampleService.exe"
installutil -i u /servicename="Sample Service 1" "c:\program files\ RIT\SampleService\SampleService.exe"
installutil -u /servicename="Sample Service 2" "c:\program files\RIT\SampleService\SampleService.exe"
installutil -i u /servicename="Sample Service 2" "c:\program files\ RIT\SampleService\SampleService.exe"
```

Code Sample

```
namespace SampleService
{
    [RunInstaller(true)]
    public class ServiceInstaller : System.Configuration.Install.Installer
    {
        /// <summary>
        /// c'tor
        /// </summary>
        public ServiceInstaller()
        {
            InitializeComponent();
        }

        public override void Install(System.Collections.IDictionary stateSaver)
        {
            RetrieveServiceName();
            base.Install(stateSaver);
        }

        public override void Uninstall(System.Collections.IDictionary savedState)
        {
            RetrieveServiceName();
            base.Uninstall(savedState);
        }

        #region Component Designer generated code

        private void InitializeComponent()
        {
            _serviceProcessInstaller1 = new System.ServiceProcess.ServiceProcessInstaller();
            _serviceInstaller1 = new System.ServiceProcess.ServiceInstaller();

            _serviceProcessInstaller1.Password = null;
            _serviceProcessInstaller1.Username = null;
            _serviceProcessInstaller1.Account = System.ServiceProcess.ServiceAccount.LocalSystem;

            _serviceInstaller1.ServiceName = _product;

            Installers.AddRange(new System.Configuration.Install.Installer[]
            {
              _serviceProcessInstaller1,
              _serviceInstaller1});
        }
        #endregion

        private void RetrieveServiceName()
        {
            var serviceName = Context.Parameters["servicename"];

            Console.WriteLine(string.Format("RetrieveServiceName: {0}", servic-
eName));

            if (!string.IsNullOrEmpty(serviceName))
            {
                this._serviceInstaller1.ServiceName = serviceName;
                this._serviceInstaller1.DisplayName = serviceName;
            }
            else
            {
                _serviceInstaller1.ServiceName = _product;
            }
        }

        private readonly string _product = "Sample Service";
        private System.ServiceProcess.ServiceProcessInstaller
_serviceProcessInstaller1;
        private System.ServiceProcess.ServiceInstaller _serviceInstaller1;
```

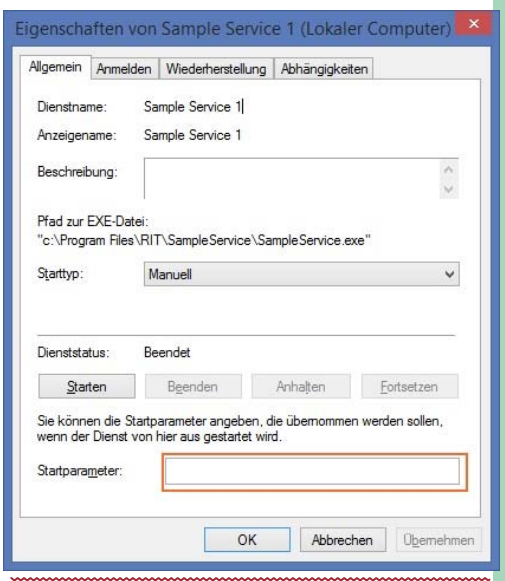


Abbildung 1: Service Startparameter

Achtung um Verwechslungen zu vermeiden: der hier eingetragene Wert ist nur innerhalb es geöffneten GUI Dialogs beim manuellen Start des Service gültig. Beim Verlassen des Dialogs ist der hier eingetragene Wert verloren, Windows speichert diesen Wert nicht. Windows bietet dazu per GUI noch immer keine Möglichkeit, seinen Start Parameter zu verwalten. Das wäre aber eine sinnvolle Verbesserung, die Startparameter hier an dieser Stelle im Dialog verwalten zu können.

Ein registriertes Windows Service landet mit einem Settings zwangsläufig in der Registry. Also wird man bei HKEY_LOCAL_MACHINE\SYSTEM-\CurrentControlSet\services auch fündig:

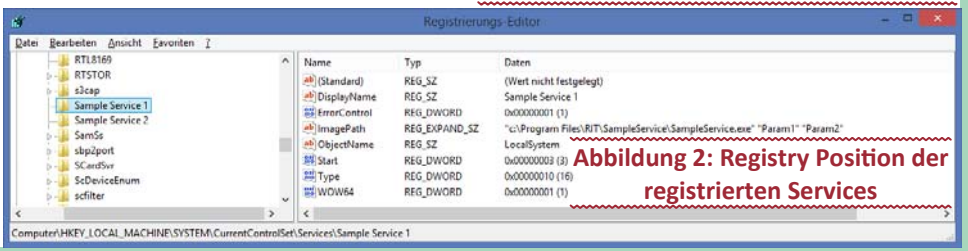


Abbildung 2: Registry Position der registrierten Services

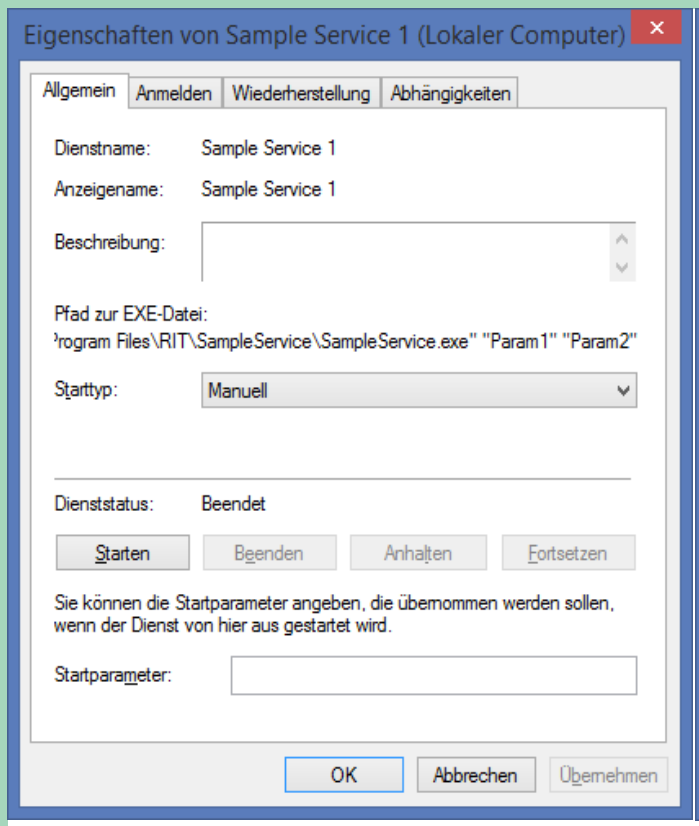


Abbildung 4: Die in der Registry ergänzten Parameter werden im Service Control GUI angezeigt

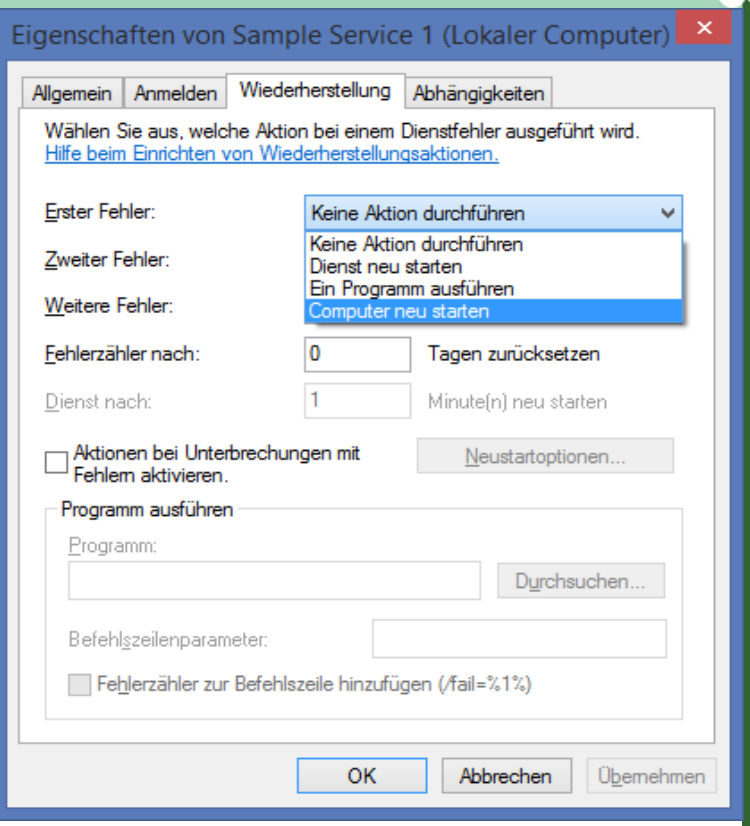


Abbildung 5: Service Wiederherstellungsmöglichkeiten wie sie das Betriebssystem bereitstellt

Also bleibt nichts anderes übrig, als direkt in der Registry den Wert zu ergänzen und somit den Parameter zu persistieren.

An jene Stelle, an der der physische Pfad zum Dateisystem eingetragen ist, kann man nun seine Start Parameter manuell ergänzen.

Achtung: Der Pfad, als auch jeder Parameter müssen unter Hochkomma „“ gesetzt werden. Jeder weitere Parameter durch ein Blank getrennt sein. Ansonsten kann das Service nicht mehr gestartet werden.

Damit die Parameter im Service auch ankommen, wird hier der Code abermals erweitert, in dem man `static void Main (string[] args)` erweitert.

Service Priority festlegen

Die Prozesse die im Service gestartet werden, können einer Prozess Priorität zugeordnet werden.

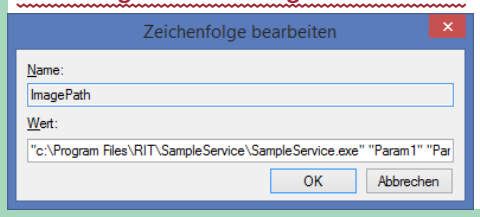
Service recovery

Treten während der Laufzeit des Service Probleme auf, bietet das Betriebssystem Wiederherstellungsmöglichkeiten an. Dazu ist im Code nichts zu ergänzen.

Fazit

Windows Services werden nicht aussterben, es werden immer Lösungen benötigt, die auf einem lokalen Environment oder nur im Intranet laufen werden. Beginnend von systemnahen Prozessen bis zu Importvorgängen. Vor allem dann, wenn man nicht in die Cloud gehen kann oder möchte, bleiben Services weiterhin sinnvoll. Anscheinend ist das der Hintergrund, warum nicht mehr in die Weiterentwicklung von Windows Services unternommen wird. Ein Windows Service implementiert man inzwischen sicherlich nicht mehr so häufig wie ein .net WebService oder eine asp.net WebSeite. Daher auch der Grund meiner Zusammenfassung von Tipps zu Windows Services, die mir während meiner Projekttätigkeit untergekommen sind.

Abbildung 3: Erweiterung um Parameter



Code Sample:

```
static void Main(string[] args)
{
  try
  {
    Console.WriteLine(string.Format("Argument items: {0}", args.Length));
    // Set priority
    System.Diagnostics.Process p = System.Diagnostics.Process.GetCurrentProcess();
    switch (Config.ProcessPriority.ToLower())
    {
       case "abovenormal":
         p.PriorityClass = System.Diagnostics.ProcessPriorityClass.AboveNormal;
         break;
       case "belownormal":
         p.PriorityClass = System.Diagnostics.ProcessPriorityClass.BelowNormal;
         break;
       case "high":
         p.PriorityClass = System.Diagnostics.ProcessPriorityClass.High;
         break;
       case "normal":
         p.PriorityClass = System.Diagnostics.ProcessPriorityClass.Normal;
         break;
       case "realtime":
         p.PriorityClass = System.Diagnostics.ProcessPriorityClass.RealTime;
         break;
       }
       Console.WriteLine(string.Format("KAV.IHE.Service: ProcessPriority: {0}",
p.BasePriority));
```