Podcast Anforderung  
  
Der Schlüssel hier ist, dass ich einen RSS-Podcast-Feed haben wollte, der sich wie eine Wiedergabeliste in WPF verhält.   
  
Um dieses ASP.NET-Skript zu verwenden, müssen Sie nur die URL des RSS-Feeds übergeben, und es gibt die ASX-Datei zurück (entweder als Inline-XML oder als Download). Beispiel: http: // [yourhost] /RssToPL.aspx?RssUrl=http://swr3.msdn.com/Media/Videos/feed/wmvhigh/ gibt eine Wiedergabeliste mit den MSDN Channel 9-Videos zurück (in umgekehrter chronologischer Reihenfolge) ).

Um noch einen Schritt weiter zu gehen, wollte ich auch, dass mehrere Podcasts in derselben Wiedergabeliste angezeigt werden. Ich wollte zum Beispiel, dass die Podcasts ".NET SWR" und andere Xml Chanels zusammen in einer Wiedergabeliste nach Datum sortiert angezeigt werden.

Sie (unter anderem) benutzerdefinierte RSS-Feeds erstellen können, die auf mehreren Quellen basieren und auf viele verschiedene Arten bearbeitet werden können.

RssFeedProcessor:

* **Die Klasse stellt Methoden bereit um:**
* **-ein XmlDocument aus einer uri zu erstellen.**
* **-Einen MemoryStream aus einem XmlDocument zu erstellen.**
* **-Die Position eines MemoryStreams auf 0 zu stellen.**
* **Alle Methoden können von außen angesteuert werden.**
* Lädt den Inhalt einer im Internet gehosteten Xml-Datei in handlebares XmlDocument.
* Ein XmlDocument, dass mit dem Inhalt der übergebenen Uri geladen ist  
    
    
  Lädt den Inhalt eines XmlDocuments in einen MemoryStream.

Setzt die Positon des instanzierten MemoryStreams auf 0 diesen zu lesen.

* param name="loadedXml">ein XmlDocument, welches geladene und lesbare Daten enthält
  + <returns>Ein MemoryStream, der mit dem Inhalt des übergebenen XmlDocuments geladen ist</returns>
* Setzt die Positon des übergebenen MemoryStreams auf 0 um diesen erneut zu lesen.

Namespace RssFeedProcessor

* **Diese Klasse regelt den Ablauf verschiedener, an der Xml-Deserialisierung teilhabenden, Klassen.**
* **Mit dem externen Aufruf der Methode "DeserializeRssXml(string xmlUri)" wird eine online gehostete Uri eines Rss-Feeds übergeben.**
* **Die Xml des Rss-Feeds enthält Daten eines Podcasts.**
* **Die Xml wird in einen Stream geladen und zur Deserialisierung weitergeleitet. Daraus entstehen zwei Objekte mit Relation zueinander.**
* **Die Liste enthält alle "Episoden" einer spezifischen "Show".**
* **Diese beiden Objekte werden gruppiert und als "Podcast" zurückgegeben.**

public class DeserializingManager

* + **Steuert den Fluss als Methodenhelfer.**
  + **Lädt Xml in einen MemoryStream. Die MemoryStreams werden an zwei Klassen verteilt.**
* **Der MemoryStream enthält eine einzige "Show" mit allen "Episoden". Diese Objekte werden aus dem Stream serialisiert und in einem "Podcast"-Objekt gruppiert.**

* <param name="xmlUri">Uri einer online gehosteten Xml Datei</param>
* <returns>Gruppierung aller Episoden einer Show</returns>

public Podcast DeserializeRssXml(string xmlUri)

{

XmlLoader xmlLoader = new XmlLoader();

XmlDocument loadedXml = xmlLoader.CreateXmlDocument(xmlUri);

using (MemoryStream memoryStreamWithXml = xmlLoader.LoadXmlDocumentIntoMemoryStream(loadedXml))

{

Show show = CreateShowObject(memoryStreamWithXml);

xmlLoader.SetMemoryStreamPositionToStart(memoryStreamWithXml);

List<Episode> episodeList = CreateEpisodeListObject(memoryStreamWithXml);

Podcast newPodcast = CreatePodcast(show, episodeList);

return newPodcast;

}

}

* **Erstellt ein Klassenobjekt (Podcast) aus den beiden Parametern.**
* **Gruppiert alle Episoden einer Show in ein "Podcast"-Objekt.**

/// </summary>

* <param name="show">Deserialisiertes Objekt.--> Stellt eine einzelne Show dar.</param>
* <param name="episodeList">Deserialisiertes Objekt.--> Enthält alle Episoden einer Show</param>

/// <returns>Erstelltes Klassenobjekt "Podcast"</returns>

private Podcast CreatePodcast(Show show, List<Episode> episodeList)

{

Podcast newPodcast = new Podcast

{

ShowInfo = show,

EpisodeList = episodeList

};

return newPodcast;

}

* **Instanziert die Klasse ShowDeserializer. Reicht einen, mit Xml geladenen, MemoryStream weiter.**
* **Erzeugt ein aus der Xml deserialisiertes "Show"-Objekt.**

* **<param name="memStream">Stream: enthält Xml einer Show mit beliebig vielen Episoden</param>**

/// <returns>Objekt einer "Show"</returns>

private Show CreateShowObject(MemoryStream memStream)

{

ShowDeserializer showDeserializer = new ShowDeserializer();

Show deserializedShow = showDeserializer.XmlToDeserializedShow(memStream);

return deserializedShow;

}

* **Instanziert die Klasse "EpisodeDeserializer". Reicht einen, mit Xml geladenen, MemoryStream weiter.**
* **Erzeugt eine aus der Xml deserialisierte Listensammlung der Klasse "Episode".**
* <param name="memoryStream">Stream: enthält Xml einer Show mit beliebig vielen Episoden</param>
* <returns>Sammlung aller "Episoden" in einem MemoryStream</returns>

private List<Episode> CreateEpisodeListObject(MemoryStream memoryStream)

* EpisodeDeserializer episodeDeserializer = new EpisodeDeserializer();

List<Episode> deserializedEpisodeList = episodeDeserializer.XmlToDeserializedEpisode(memoryStream);

return deserializedEpisodeList;

}

}

}

* **Episoden Deserializer**  
  Die Klasse EpisodeDeserializer besitzt drei Aufgaben:
* Xml-Knoten, die Daten einer "Episode" enthalten, an Properties mappen.
* Eine Xml deserialisieren und die Werte an die Properties binden.
* Die Properties an ein "Episode"-Listenobjekt und zurückgeben.
* Die Klasse wird mit Aufruf der Methode "XmlToDeserializedEpisode(MemoryStream xmlStream)" angesteuert. Der MemoryStream enthält eine Rss-Feed-Xml eines Podcasts.
* Anhand den gemappten Properties kann die Xml deserialisiert werden.

Code:

namespace RssFeedProcessor

{

[XmlRoot("rss")]

public class EpisodeDeserializer

{

* Interne Liste der Klasse "DeserializedEpisode". Nicht fähig für übergreifenen Datentransfer. Ist an eine Listenobjekt des Typs "Episode" gebunden werden.

[XmlIgnore]

private List<DeserializedEpisode> AllDeserializedEpisodes { get; set; }

//DataTransferObjekt der Klasse "Episode" welches zurückgegeben werden soll.

[XmlIgnore]

public List<Episode> EpisodeListDTO { get; set; }

//Map Properties

#region MappedProperties

* **Hinweis: Um Attribute eines Xml-Elements zu deserialisieren muss eine nested-Class verwendet werden.**

*(Beispiel: die Property "FileInfo" des Typs "FileData" zeigt das XmlElement an das gelesen werden muss.*

* **Die Definition der Klasse "FileData" zeigt das XmlAttribute an (url), das gelesen werden soll.**

[XmlElement("channel")]

public ChannelNode Channel { get; set; }

public class ChannelNode

{

[XmlElement("item")]

public List<DeserializedEpisode> DeserializedEpisodeList { get; set; }

}

public class DeserializedEpisode

{

[XmlElement("title")]

public string Title { get; set; }

[XmlElement("pubDate")]

public string PublishingDate { get; set; }

[XmlElement("keywords", Namespace = "http://www.itunes.com/dtds/podcast-1.0.dtd")]

public string Keywords { get; set; }

[XmlElement("description")]

public string Summary { get; set; }

[XmlElement("image", Namespace = "http://www.itunes.com/dtds/podcast-1.0.dtd")]

public ImageLink LinkToImage { get; set; }

[XmlElement("enclosure")]

public FileData FileInfo { get; set; }

}

public class ImageLink

{

[XmlAttribute("href")]

public string Link { get; set; }

}

public class FileData

{

[XmlAttribute("url")]

public string PodcastUri { get; set; }

[XmlAttribute("length")]

public int Length { get; set; }

[XmlAttribute("type")]

public string Type { get; set; }

}

#endregion MappedProperties

Helfermethode regelt den Methodenfluss. Diese Methode wird von außen angesteuert und erhält einen MemoryStream mit geladener Xml.

* **Sie ruft eine Methode auf, welche die vielzahl an "Episoden" aus der Xml deserialisiert.**
* **Das deserialisierte Objekt wird durch eine zweite Methode an ein DTO-Listenobjekt des Typs "Episode" gebunden und zurückgegeben.**

/// </summary>

/// <param name="xmlStream">Stream: enthält Xml einer Show mit beliebig vielen Episoden</param>

/// <returns></returns>

public List<Episode> XmlToDeserializedEpisode(MemoryStream xmlStream)

{

XmlLoader deserializingProcessor = new XmlLoader();

DeserializeXmlToMappedPodcastEpisode(xmlStream);

SerializedShowToDataTransferObject(AllDeserializedEpisodes);

return EpisodeListDTO;

}

Instanziert einen XmlSerializer. Der XmlSerializer wird mit den gemappten Properties der Klasse ShowDeserializer geladen.

* **Anhand den gemappten Properties werden nun die Knotenwerte der Xml an die übereinstimmenden Properties gebunden.**
* **Jede einzelne deserialisierte Episode wird derselben klasseneigenen Property-Liste "AllDeserializedEpisodes" hinzugefügt.**
* **Für einen xmlStream enstehen so viele Listeneinträge wie die Xml-Datei Episodenknoten hat.**

/// <param name="xmlStream">Stream: enthält Xml einer Show mit beliebig vielen Episoden</param>

private void DeserializeXmlToMappedPodcastEpisode(MemoryStream xmlStream)

{

XmlSerializer deserializer = new XmlSerializer(typeof(EpisodeDeserializer));

DeserializedEpisode serializedShow = new DeserializedEpisode();

EpisodeDeserializer episodesCollection = new EpisodeDeserializer();

episodesCollection = (EpisodeDeserializer)deserializer.Deserialize(xmlStream);

AllDeserializedEpisodes = episodesCollection.Channel.DeserializedEpisodeList;

}

Bindet die klasseneigenen Properties an eine Instanz der allgemeinen "Show"-Klasse.

* **Mit dem konditionellen Operator "?:"**

**wird a) ein default-Wert an eine non-nullable Property zugewiesen.**

**oder b) eine alternativer Property-Wert zugewiesen.**

**Es werden so viele Listeneinträge initialisiert wie es Listeneinträge im übergebenen Parameter gibt.**

* **<param name="deserializedShow">Deserialisierte Liste mitEpisodeneinträgen.**
* **Nicht fähig für übergreifenen Datentransfer. Muss an eine Listenobjekt des Typs "Episode" gebunden werden.</param>**

private void SerializedShowToDataTransferObject(List<DeserializedEpisode> deserializedShow)

{

EpisodeListDTO = new List<Episode>();

foreach (DeserializedEpisode item in deserializedShow)

{

Episode newEpisode = new Episode

{

Title = item.Title,

PublishDate = ConvertDateTime(item.PublishingDate),

Keywords = item.Keywords,

Summary = item.Summary,

ImageUri = item.LinkToImage != null ? item.LinkToImage.Link : "",

FileDetails = new FileInformation(item.FileInfo.PodcastUri, item.FileInfo.Length, item.FileInfo.Type)

};

EpisodeListDTO.Add(newEpisode);

}

AllDeserializedEpisodes = null;

}

Initialisiert den Zugriff auf einen DateTimeParser.

/// <param name="dateTimeForParsing">string, der zu DateTime geparsed werden soll</param>

/// <returns>DateTime Objekt</returns>

private DateTime ConvertDateTime(string dateTimeForParsing)

{

DateTimeParser dateParser = new DateTimeParser();

return dateParser.ConvertStringToDateTime(dateTimeForParsing);

}

}

}