Umbraco Courier 2.5

Data Resolvers and event providers

Per Ploug Hansen

10/3/2011

Table of Contents

Introduction

This document outlines how 2 central components in Courier works:

1. Its data resolvers and how they can change and process data
2. Its item event providers which can trigger and queue events

These concepts are useful for people who wish either extend or change the way Courier works with the website content.

Data resolvers will show a developer how Courier can understand your data

Item event handlers enables a developer to trigger and queue events, such as Lucene Indexing, triggering workflows and so on.

Intended audience

Developers who understands .net, c# and has a clear idea of how Umbraco works, and what components in Umbraco does what.

These concepts are targeted developers who wish to add support for 3rd party components such as Data Types, or change or extend the way Courier handles current built-in components.

Revision History

- Version 1, 25/9/2011 Outline

Data Resolvers

A data resolver is a custom model built into Courier 2, to handle 3rd party data types, storing data in a custom way, or other custom components in your website, where Courier doesn't understand the stored data.

In short, a Data resolver is simply a .net class, which inherits from a specific base class, which allows the developer to hook into different events during the data packaging and extraction.

Out of the box, Courier 2, can understand all standard data-types in umbraco. This means that Courier knows that a Content picker contains a ID, pointing at a document, which then becomes a dependency, and the ID gets translated into a Guid which can safely be deployed to another location. It also knows that a template might contain references to javascript files or internal links, using the `[locallink:]` syntax. Or a a lot of other cases where data have a special meaning.

This is what dataresolvers do, add special meaning to specific data that matches certain criteria, for instance properties using a specific datatype, templates containing a certain keyword and so on.

Sample data resolver

To show some code as fast as possible here is a commented code sample which outlines a simple resolver:

```
//inherite from ItemDataResolver and implement ResolvableTypes and ShouldExecute
public class test : ItemDataResolverProvider
{
    //resolvableTypes are the types of content which can be processed by this resolver
    //all built-in classes are available under Umbraco.Courier.ItemProviders
    //this class handles all templates and nothing else
    public override List<Type> ResolvableTypes
    {
        get { return new List<Type> { typeof(Template) }; }
    }

    //ShouldExecute, a fast way to determine if the provider should trigger or not, under a specific event
    //in this case the resolver will only trigger during Packaging Event and if the template has any
    //javascript resources packaged
    public override bool ShouldExecute(Item item, Core.Enums.ItemEvent itemEvent)
    {
        if (itemEvent == Core.Enums.ItemEvent.Packaging)
        {
            var t = (Template)item;
            return t.Resources.Where(x => x.ExtractToPath.EndsWith(".js")).Count() > 0;
        }

        return false;
    }

    //Implements the packaging event, here we can change the content of the item being processed
    //we have access to all the data  and can replace anything, which will then be saved to the revision.
    public override void Packaging(Item item)
    {
        //here we simply just fetch the javascript resource and could then do something with those
        foreach (var jsFile in item.Resources.Where(x => x.ExtractToPath.EndsWith(".js")))
        {
            //do something with that jsFile
        }
    }
}
```

The above data resolver simply hooks into Courier item providers and targets all items with type Template, and performs an action during the Packaging event.

Where can you use a data resolver?

Data resolvers can hook into several events during the extraction, packaging and post-processing of any data:

Event	Description
Packaging	Happens before data is packaged into the revision folder
Packaged	Happens after data is packaged and values have been replaced
Extracting	Happens before extraction starts
Extracted	Happens when the item extraction has been completed
PostProcessing	Only happens if the item is marked for Postprocessing, happens after extracted
PostProcessed	Happens after postprocessing is completed

A data resolver is also able to hook into the processing of resources attached to a item, this can be handy to for instance handle the .master file associated with an umbraco template.

The data resolvers has an event model for this as well:

Event	Description
PackagingResource	Happens before data is packaged into the revision folder
PackagedResource	Happens after data is packaged and values have been replaced
ExtractingResource	Happens before extraction starts
ExtractedResouce	Happens when the item extraction has been completed

All these events provides a developer with all the possible extension points during the entire life-cycle of a piece of umbraco data in Courier, from the time Courier packages it into xml, to its done extracting it into another installation.

Built-in item types the resolves can target

Data resolvers filters by the types of items they can process, the list of built-in types are below:

Name	Fully qualified name and description
ContentPropertyData	`Umbraco.Courier.Core.ItemProviders.ContentPropetyData` Handles propertydata on both media and content
DataType	`Umbraco.Courier.Core.ItemProviders.DataType` Handles datatypes
DictionaryItem	`Umbraco.Courier.Core.ItemProviders.DictionaryItem` Handles dictionary items
Document	`Umbraco.Courier.Core.ItemProviders.Document` Handles basic document data, structure and paths NOT the property itself
DocumentType	`Umbraco.Courier.Core.ItemProviders.DocumentType` Handles document type and propert types
File	`Umbraco.Courier.Core.ItemProviders.File` Handles individual files
Folder	`Umbraco.Courier.Core.ItemProviders.Folder` Handles folders
Language	`Umbraco.Courier.Core.ItemProviders.Language` Handles languages
Macro	`Umbraco.Courier.Core.ItemProviders.Macro` Handles macros
MacroPropertyType	`Umbraco.Courier.Core.ItemProviders.MacroPropertyType` Handles macro property types
Media	`Umbraco.Courier.Core.ItemProviders.Media` Handles basic document data structure and paths, NOT the media property data
MediaType	`Umbraco.Courier.Core.ItemProviders.MediaType` Handles media type and property types
StyleSheet	`Umbraco.Courier.Core.ItemProviders.Stylesheet` Handles stylesheets and individual stylesheet properties
TagRelations	`Umbraco.Courier.Core.ItemProviders.TagRelations` Handles tags and their relations
Template	`Umbraco.Courier.Core.ItemProviders.Template` Handles tags and the .master files

Notice: Courier 2 is not limited to use the classes in the `Umbraco.Courier.Core` namespace. It can use any type that inherites from `Umbraco.Courier.Core.Item`.

You can therefore without issues create your own Item provider and associate a custom Data Resolver to such a provider.

Syntactic sugar for data types

Handling data types, their configuration and the stored data is a bit more complex as it involves several moving pieces:

- The data types item provider containing the data type and its configuration
- The property data which contains the actual stored data from the data type

So using the normal data resolver model, you would need to hook into 2 different providers and process the data.

However, from version 2.5 Courier supports a data resolver specifically targeted at simplifying this: the "`PropertyDataResolverProvider`".

PropertyDataResolverProvider provides a more specialized event model to hook into:

Event	Description
PackagingProperty	Happens before property data is packaged into the revision folder
PackagedProperty	Happens after property data is packaged and values have been replaced
ExtractingProperty	Happens before property extraction starts
ExtractedProperty	Happens when the property data extraction has been completed
PackagingDataType	Happens before the datatype and its configuration is packaged into the revision folder
PackagedDataType	Happens after datatype is packaged and values have been replaced
ExtractingDataType	Happens before datatype extraction starts
ExtractedDatatype	Happens when the datatype extraction has been completed

Besides this simplified event model, the Resolver matching is purely done based on the datatype GUID, which is then able to match both data type and propertye data, based on this GUID. Making the code much more transparent

Sample PropertyDataResolverProvider

This sample goes through processing a datatype which stores a list of images in a custom format like so: “Name|image1.gif” “Name2|image2.gif “, “Nameshdshd|image45.png” in its configuration.

It looks at that configuration and tells courier the files it can find so courier remembers to transfer them, and finally this sample changes some data on the media / document properties using this data type.

```
//inherit from PropertyDataResolverProvider
public class ImageDropdownlist : PropertyDataResolverProvider
{
    //the GUID of the datatype
    public override Guid DataTypeId
    {
        get { return new Guid("a4ca44c9-ebb6-48e8-8d39-96bfdf619825"); }
    }

    //happens while we package the data type and it's configuration
    //as well as prevalues
    public override void PackagingDataType(ItemProviders.DataType item)
    {
        //we go through the settings/prevaues and save references to images stored in the datatype
        foreach (var setting in item.Prevalues.Where(x => x.Value.Contains("|") ))
        {
            //split the settings on the | char
            var currentSetting = setting.Value.Split('|');
            var file = currentSetting[1];

            //simply add to the item.Resources to store and transfer as part of the revision
            item.Resources.Add(file);
        }
    }

    //here we intercept the actual data and replace any unicorn mention with "horse"
    public override void PackagingProperty(Core.Item item, ItemProviders.ContentProperty propertyData)
    {
        //get the refence to the property data object the data is part of
        var properties = (ContentPropertyData)item;

        if (propertyData.Value.ToString() == "unicorn")
            propertyData.Value = "Horse";
    }
}
```

Helpers for working with custom XML data

A common use-case is a data type storing node ID references in an xml structure. Due to umbracos history and xml usage, this is a common road for data type developers to take.

For instance storing data like so:

```
<nodes>
  <nodeid>1627</nodeid>
  <nodeid>27282</nodeid>
</nodes>
```

Courier can without issues transfer this, but does not know that the data contains node Ids, so on the other end, this data will break.

To make it work, Courier needs to know what IDs the xml contains, convert these ids into GUIDs and finally convert those GUIDs back to the corresponding Node IDs when the item is extracted in another location

To solve this, Courier 2.5 comes with a couple of simple helpers which can help digest this xml

Umbraco.Courier.Core.Helpers.XmlDependencies.ReplaceIds

This simply replace Node Ids with Guids, given a chunk of Valid Xml, and an Xpath Query it will go through the xml and replace ids.

```
ReplaceIds(xml, xpath, attribute, direction, out replaceIds);
```

Parameters

- **Xml**: The xml to search for IDs
- **Xpath**: the query to find the IDs
- **Attribute**: optional, the name of an attribute on the found nodes which contains the ID
- **Direction**; enum, can be either **FromNodeIdToGuid**, or **FromGuidToNodeId**
- **ReplacedIds:** optional, returns a list of NodeIds replace by the method

Returns

The Xml as a string with all IDs replaced

Sample

```
string dataXpath = "//nodeId";
List<string> replacedIds = new List<string>();
propertyData.Value = XmlDependencies.ReplaceIds(
                        propertyData.Value.ToString(),
                        dataXpath,  IdentifierReplaceDirection.FromNodeIdToGuid,
                        out replacedIds);

//these are the IDs we found in the picker, those documents are a dependency
foreach (string guid in replacedIds)
{
    item.Dependencies.Add(guid, ProviderIDCollection.documentItemProviderGuid);
}
```

Umbraco.Courier.Core.Helpers.XmlDependencies.FindResources

Searches xml for resource paths, using an Xpath Query

```
FindResources(xml, xpath, attribute);
```

Parameters

- **Xml**: The xml to search for resource paths
- **Xpath**: the query to find the resource paths
- **Attribute**: optional, the name of an attribute on the found nodes which contains the path

Returns

A List<string> containing paths to all found resources

Sample

```
string resourceXpath = "//url";
foreach (var resource in XmlDependencies.FindResources(propertyData.Value.ToString(), resourceXpath, null)
)
{
    item.Resources.Add(resource);
}
```

PersistenceManager.Default.GetNodeId, GetUniqueId

If the built-in replaces doesn't work for your data, you can access node ids. And Unique Ids in the database through the Persistence Manager. This enables you to translate the Node ID => Guid or Guid => Node ID.

As an optional parameter, you can pass the Umbraco NodeObjectType to this method to filter the type of node you wish to retrieve the id/guid of.

A refence to all NodeObjectTypes is located in `Umbraco.Courier.ItemProviders.NodeObjectTypes`

```
int nodeId = PersistenceManager.Default.GetNodeId(docGUID);
int nodeId2 = PersistenceManager.Default.GetNodeId(docGUID, NodeObjectTypes.Document);
```

Or the other way around:

```
Guid nodeGuid = PersistenceManager.Default.GetUniqueId(docID, NodeObjectTypes.Media);
Guid nodeGuid = PersistenceManager.Default.GetUniqueId(docID);
```

ItemEventProvider

An Item event provider allows a developer to attach events to items during an extraction. Exemples of usage could be:

- Triggering Lucene indexing
- Publishing items
- Refreshing caches
- Rebooting application pool

It is basicy events you would like to postpone / queue until after all data has been safely extracted and the database is done with its transaction, unlocking the inserted and updated data.

The ItemEventProviders is a replacement of the standard umbraco event system, as Courier does not use the standard umbraco API, and can therefore not trigger the standard event handlers, which is usually a good thing. We won't trigger any 3rd party handlers or accidently cause a endless loop, everything is isolated which also increases the success rate.

But in some cases, it is needed to trigger an event as soon as Courier 2 is done extracting its items. Which is why the ItemEventProviders has been added.

A sample item event provider

This is a very simple provider, as it simply has an alias and a execute mehthod which sends a message to twitter:

```
public class test : ItemEventProvider
{
    public override string Alias
    {
        get { return "TweeTOnDeploy"; }
    }

    public override void Execute(ItemIdentifier itemId,SerializableDictionary<string, string> Parameters)
    {
        My.Custom.TweetLibrary("woah, I just deployed some stuff");
    }
}
```

Triggering the item event provider

Item event providers are triggered from either the Item provider itself during extraction, or from an added data resolver.

For both of them, you call the Courier Execution Context, and tell it to either queue the event for later or trigger it now.

```
//execute the event code now
ExecutionContext.ExecuteEvent("TweeTOnDeploy", item.ItemId, null);

//execute the event when Deployment is completed
ExecutionContext.QueueEvent("TweeTOnDeploy", item.ItemId, null, Umbraco.Courier.Core.Enums.EventManagerSys
temQueues.DeploymentComplete);
```

Queues

Item event providers are added to built-in queues which triggers at different times. The standard queues are:

- **ExtractionComplete**
  Triggers when all items have been extracted
- **PostProcessingComplete**
  Triggers when all items marked for postprocessing has been processed
- **DBTransactionComplete**
  Happens just after the database transaction is committed and the Database frees the locks
- **DeploymentComplete**
  Happens after all built-in processes has been run