Overview

Introduction

Collections combine large groups of similar items on the internet, so we can begin viewing the Web as a "web" rather than a series of isolated pages. As a developer, you can create collections of your own. New collections can be created with no code – they’re only data. This page contains an overview of the collection-building process, as well as links to more detailed pages on specific topics. This overview contains the following sections:

* [Creating a Collection](#Creating_a_Collection)
* [Kinds of Collections](#Kinds_of_Collections)
* [Architecture](#Architecture)

Creating a Collection

Collections are composed of two parts:

* **XML** – The items in the collection are described in XML, in a ".cxml" file. For details, see: [Collection XML Schema](#XML_Schema).
* **Images** – The images in the collection are described in Deep Zoom format. For details, see: [Collection Image Content](#Image_Content).

Creating a collection has four distinct steps:

1. **Pick your data** – First, pick a set of data to turn into a collection and decide how you want to present it. For tips, see [Collection Design](#Collection_Design).
2. **Create XML and images** – Once you have your data sources, you’ll need to describe it in Collection XML (CXML) and transform your images to the Deep Zoom format. See [Collection XML Schema](#XML_Schema), [Collection Image Content](#Image_Content), and [Collection Design](#Collection_Design) for detailed information.
3. **Host it** – To share your collection with others, host it on a web server. For more information, see [Collection Hosting](#Collection_Hosting).
4. **Share it** – When it’s hosted, share it with the world!

Kinds of Collections

There are three primary kinds of collections. They differ primarily in size and their ability to respond to custom user queries. Structurally, they are composed of either previously generated (or *static*) XML, or XML generated *dynamically* in response to a query. See the following figure for more information on the three kinds of collections.

|  |  |  |
| --- | --- | --- |
| Simple Collections | Linked Collections | Dynamic Collections |
| http://ll-pivotweb03/getpivot/images/collections/image009.png | http://ll-pivotweb03/getpivot/images/collections/image010.png | http://ll-pivotweb03/getpivot/images/collections/image011.png |
| **Difficulty:** Easy  **Size:** Up to 3,000 items | **Difficulty:** Medium  **Size:** Limited by storage complexity | **Difficulty:** Hard  **Size:** Unbounded |
| **Details:**   * The most common type of collection. * Data is static and loaded all at once. * Visuals are static and contained in one DZC (see: [Collection Image Content](#Image_Content)). | **Details:**   * Generally used for collections in the several thousands of items. * Collection is stored as a composition of inter-linked *simple* collections. * Data is static and loaded one *simple* collection at a time. * Visuals are static and contained in multiple DZCs (see: [Collection Image Content](#Image_Content)). | **Details:**   * Used for very large data sets (hundreds of thousands of items or more). * Data is dynamic (see [Collection Hosting](#Collection_Hosting)) and loaded in response to a query. * Visuals are partially dynamic and contained in dynamic DZCs (see: [Collection Image Content](#Image_Content)). |

For more details, see: [Collection Hosting](#Collection_Hosting).

Collection Design

Introduction

At its root, designing a collection means deciding how to present a large amount of information at once. The key elements to designing a great collection are relatively simple: pick a subject matter with great images and interesting details, choose a small number of properties to filter on, and add links to related content. This page provides guidelines and details for each of the following:

* [Design Process](#Design_Process)
* [Facets and Facet Categories](#Facet_Categories)
* [Organizing Facet Categories in Pivot](#Organizing_Filter)
* [Facet Naming](#Facet_Naming)
* [Facet Values and Formatting](#Facet_Values)
* [Imagery](#Imagery)
* [Design Checklist](#Design_Checklist)

For documentation on the mechanics of creating collections, please see: [Collection XML Schema](#XML_Schema)

Design Process

Designing a collection is an iterative process. Start with something simple, then build it and try it. Add complexity gradually.

The basic steps are:



1. **Define the Subject Matter**  
   First, decide what subject your collection will cover. Great collections typically involve subjects that someone would want to spend lots of time exploring, rather than a subject matter where the user is looking to pin-point data about a particular topic. They also enable the user to discover trends across all items using different views.
2. **Define the User**   
   Second, define the user of your collection and their needs. For example, if you were building a collection about cars, a car racing enthusiast would want different content than a person shopping for a car.
3. **Define the Key Tasks**  
   Third, define the key tasks the user will perform with the collection. Tasks can be both about specific goals and browsing. Decide on the key views and kinds of information you want the user to be able to see.
4. **Choose Content and Data**  
   Fourth, choose the facet categories, details, links, default view, etc. that will help the user accomplish the key tasks. Edit or prune your data as necessary to get the best experience.
5. **Evaluate and Iterate**   
   Finally, experiment with your collection to evaluate it. Step through the key tasks from step three. If they can't be accomplished easily, go back to step four. Finally, go through the [Collection Design Checklist](#Design_Checklist).

Facets and Facet Categories

Once you have chosen your subject matter, the most important step in collection design is choosing which *facet categories* and *facets* to use, and where to display them. *Facets* are similar to property values on an item, and *facet categories* are groups of facets. For example: if a collection had a facet category called "U.S. State," then "Georgia" could be a facet in that category. Facets can take several different forms in the Pivot client. Most notably, they appear as filters over a collection’s items, detailed information about an item, and/or links to related content. For more information on facet categories and facets, see [Collection XML Schema](#XML_Schema).

The following diagram illustrates where facet categories and facets appear in Pivot.



The key points of control to keep in mind when designing facet categories (and thus the facets that will appear in them) are:

* **Naming and formatting** – Appropriate naming, formatting, and ordering of facet categories improve readability and comprehension. For details, see the [Naming](#Facet_Naming) and [Values & Formatting](#Facet_Values) sections below.
* **Presence as a filter** – You can choose whether the category appears in the filter panel. If a category appears in the filter panel, it is also available as a sort option in the view.
* **Presence in the info panel** – You can also choose whether the category appears in the info panel, as part of the details of an item.
* **Inclusion in keyword filtering** – Keyword filtering allows the user to enter keywords to use as filters in a collection. It is generally recommended that only facet categories that help identify the item (ex: actor name) be included in keyword filtering, as opposed to more tangential properties (ex: an actor’s list of co-stars).

For more information, see the detailed sections below and [Collection XML Schema](#XML_Schema).

Organizing Facet Categories in Pivot

When you are designing a collection, consider the different parts of Pivot one at a time and decide what you want them to contain. Use the following guidelines to help you.

Filter Panel

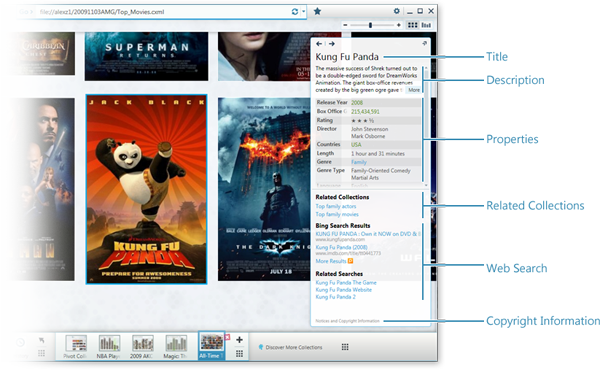
Generally, collections are best with 5 to 8 categories in the filter panel. Although it may be tempting to include a large number of facets, this can often overwhelm the user.

To give the user lots of opportunity for exploration, try to find interesting combinations of categories that will occur when the user filters with facets from different categories at once.

Info Panel

When the user zooms in on a collection item, detailed information about it appear in the info panel. This part of Pivot serves two purposes:

* **Providing Detail** – Rich information allows the user to read details about the item they are looking at.
* **Encouraging Exploration** – Links help the user find similar or related items, or jump to other web content.

  
**Title**

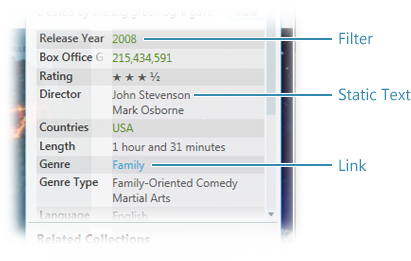
This is the name of the item.

**Description**

This is a special field that is displayed above the property section of the info panel.

**Properties**

The properties section contains *facets* on this item. Along with dates and numbers, there are three different kinds of text that can appear here. See the following figure for reference:



* **Filters** - When clicked, these filter the collection to see other items that have the same facet (in the example above, the movie release year "2008"). If a facet category appears in the filter panel, it will also appear as a filter in the info panel.
* **Static Text** - If a textual category does not make sense as a filter or link, use static text.
* **Links** - These are normal hyperlinks. They may be used for creating linked collections (in the example above, a link to a collection of movies in the "Family" genre).

In some cases, you may want a facet category to appear differently in the filter panel from the info panel. For example, you may want to use a different format or type in each, even though the data is the same. In these cases you can create two facet categories: one to appear in the filter panel and another to appear in the info panel.

Other Links

The bottom section of the info panel contains a series of links to related content. They are:

* **Related Collections** – These are links to other collections that are in some way related to this item. For example, when looking at a collection of actors, there might be a link to a collection of movies this actor has starred in.
* **Notices and Copyright Information** - The copyright attribution text (which can be a link) may be specified in the CXML. This attribution text appears at the very bottom of the item info panel.

Facet Naming

In general, you want concise and clear names for facets and facet categories that don’t get truncated in common usage. Below are some guidelines:

* **Capitalization** – Facet categories should follow [headline/title style conventions](http://www.chicagomanualofstyle.org/ch08/ch08_sec167.html) for capitalization. Facet values are case-sensitive, so make sure the values use consistent casing throughout.
* **Plurality** - Use the singular form of the category name. For example: Rather than naming a category "Countries," use "Country."
* **Brevity and Redundancy** - Use as few words as possible to name categories. Keep in mind that the user is already browsing within the context of your collection, so any concepts or words which inherently make sense in that context can be taken as implied. For example: in a collection named "Major League Soccer Players," a facet named "Player Position" could be shortened to "Position." A facet named "Average Points Scored per Game" could be "Average Points / Game," or simply "Average Points." Finally, use the active voice to avoid unnecessarily long phrases.
* **Acronyms** - While they help with brevity, acronyms may add confusion. Avoid them where possible, unless you know that the target audience will be inherently familiar with them. Try using abbreviations instead.
* **Numbers** - Use Arabic numerals ("1, 2, 3") rather than spelling them out ("one, two, three"). They’re shorter and easier to read quickly.
* **String vs. LongString** - LongString is a special type only used for info panel content to be wrapped onto multiple lines. LongString should be used for any text longer than a few words.
* **Sorting Behavior** - Sorting and grouping is strictly alphabetical. That means that when values starting with "The" appear in Pivot, they will be grouped with the T’s.

Facet Values and Formatting

The following guidelines concern facet values, as well as facet formatting:

* **Single Item Values** - Avoid categories that have a very large number of facets, or categories where clicking a facet will frequently result in only one item being in view ("Social Security Number," for example). These categories provide a poor exploration experience, and are generally better in the info panel only. See illustration:

|  |  |
| --- | --- |
| http://ll-pivotweb03/getpivot/images/collections/image035.png | http://ll-pivotweb03/getpivot/images/collections/image036.png |
| Recommended | Not Recommended |

* **Compound Category** - You may encounter situations where you need to create a "compound category." Creating a compound category is a best practice for handling values which you want to display together in the info panel, but that would also make good filters if displayed separately. In this case, create special categories for each part of the UI. As an example, suppose you want to allow users to filter both by city and by state, but would like to display them in the form "city, state" in the info panel. In this case, you can create separate categories to put in the filter pane for City (ex. facet: "Atlanta") and State (ex. facet: "Georgia"), and a third for the info panel called Location (ex: facet "Atlanta, Georgia").
* **Optimize Distribution for Interest** – When examining each category, consider how they will appear in the graph view. If the data has large gaps or is very skewed, it will probably not provide a great distribution when sorted by that category. In this case, you should consider how omitting some facet categories, pruning whole outlier items, or bucketing your values could enhance the experience and keep the focus on the dense part of the distribution. If you choose to bucket the values, you might create a compound facet, as described above, where the actual value for an item is displayed in the info panel but the range buckets are displayed as filters.
* **Consistent Weights and Measures** - Use a consistent system (such as metric or English) across the collection.
* **Currency** - If the facet value represents a quantity of money, always use an appropriate [currency format string](http://msdn.microsoft.com/en-us/library/0c899ak8.aspx).
* **Number Formatting** – In some cases, you may need to use a format string containing the "#" character. In this case, be sure to include a zero character in the format (at the end). See also the [string format reference](http://msdn.microsoft.com/en-us/library/0c899ak8.aspx).
* **Units** - Include units in your number formats where appropriate, rather than in the facet names.

Imagery

Given the visual nature of the Pivot experience, quality imagery is of utmost importance. Below are some guidelines for great imagery:

* **High Image Quality** - Images of at least 500px on the long side are recommended, but always use the highest resolution images available. Because of Pivot’s usage of Deep Zoom technologies, there is no extra waiting or other cost for the user if a collection contains big images. The higher resolution the image, the better!
* **Consistent Aspect Ratio** - The views in Pivot look best when the items are all similar aspect ratios. Consider pruning or replacing imagery for items with very divergent aspect ratios, as they will affect the layout and spacing of the entire view.

|  |
| --- |
| http://ll-pivotweb03/getpivot/images/collections/image037.png |
| Using a consistent aspect ratio ensures a regular, dense collection view |

* **Source Image Format** - It is recommended (but not required) that source imagery be in the JPG format (PNG if requiring transparency), and that compression be kept low when creating them (quality settings should be in the 85%-95% range). Images must then be converted to Deep Zoom format, [Collection Image Content](#Image_Content) pages for more information.
* **Edge Treatments** - If your imagery is white or very light on the edges of the images, consider adding a darker border to give it more contrast with the Pivot background.
* **Enhanced Collection Imagery** - Some collections may benefit from adding graphics and text to the items. Adding some simple layout and information design to the items can produce an even richer and more compelling collection. Graphical treatments work well to show higher level statistics, trends or classifications or label details that are very useful to have in-context with the image. Having this information in context allows the user to scan across multiple items quickly without having to look in the info panel for each one in turn. As such, these graphics should be easy to scan when zoomed out, but also provide additional detail when zoomed in.

|  |  |  |
| --- | --- | --- |
| http://ll-pivotweb03/getpivot/images/collections/image038.png | http://ll-pivotweb03/getpivot/images/collections/image039.png | http://ll-pivotweb03/getpivot/images/collections/image040.png |
| Enhanced Collection Imagery | | |

Design Checklist

* Basics
  + Is the collection named well? Would the target user understand what it is about?
  + Good quality iconography, branding and attribution?
  + Is the default view interesting given the subject matter and target user?
* Images
  + Good enough quality?
  + Would it help to add text or graphics to the items?
* Facet Categories and Facets
  + Are the categories in the filter pane the right ones? How many do you have? (5-8 is ideal)
    - Should any be added?
    - Are there any categories a user might need not in the collection currently?
    - Are there any categories with a long list of facets? If so, can the values be collapsed?
    - Are the categories in the right order?
  + Are the categories named properly? Do they use the correct conventions and will your target user be familiar with the terms used?
  + Are the facet values formatted properly?
  + Are the right categories in the info panel?
  + Should this collection be linked to other collections?
  + Are you happy with the search results?
* Can you complete all of the key tasks that you defined in the planning phase?

Collection XML Schema

Introduction

Collection XML (CXML) is the schema used to describe structured data to be displayed in the Pivot collection experience. This page contains detailed information about this schema in the following sections:

* [Hello World Example](#Hello_World)
* [Schema Structure](#Schema_Structure)
* [Element Overview](#Element_Overview)
* [Schema Details](#Schema_Details)

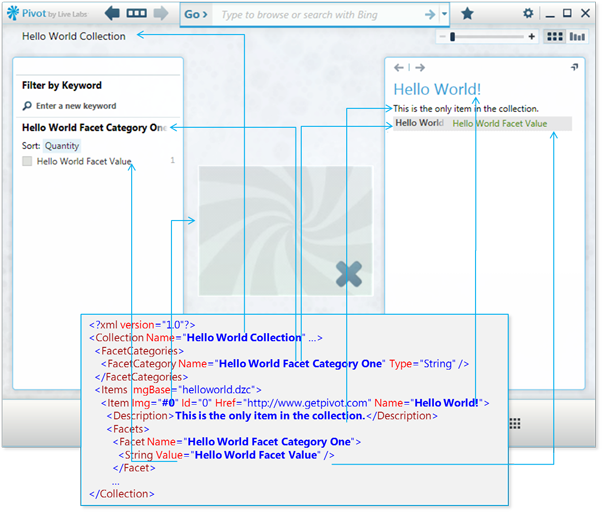
At a high-level, CXML can be thought of as a set of property/value pairings. Facets are like property values on an item, and facet categories are groups of facets. For example: if a collection had a facet category called "U.S. State," then "Georgia" could be a facet in that category. Depending on authoring choices, these facets may be displayed as filters in the Pivot collection experience, or included in the details of an item.

Hello World Example

The following sample XML specifies a simple collection with only one item.

<?xml version="1.0"?>   
<Collection Name="Hello World Collection" SchemaVersion="1.0"   
xmlns="http://schemas.microsoft.com/collection/metadata/2009"   
xmlns:p="http://schemas.microsoft.com/livelabs/pivot/collection/2009"   
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
xmlns:xsd="http://www.w3.org/2001/XMLSchema">   
  <FacetCategories>   
    <FacetCategory Name="Hello World Facet Category One" Type="String"/>   
  </FacetCategories>   
  <Items ImgBase="helloworld.dzc">   
    <Item Img="#0" Id="0" Href="http://www.getpivot.com" Name="Hello World!">   
      <Description> This is the only item in the collection.</Description>   
      <Facets>   
        <Facet Name="Hello World Facet Category One">   
         <String Value="Hello World Facet Value"/>   
        </Facet>         
      </Facets>   
    </Item>   
  </Items>   
</Collection>

Now, let’s look at how each part of the XML is surfaced in the client.



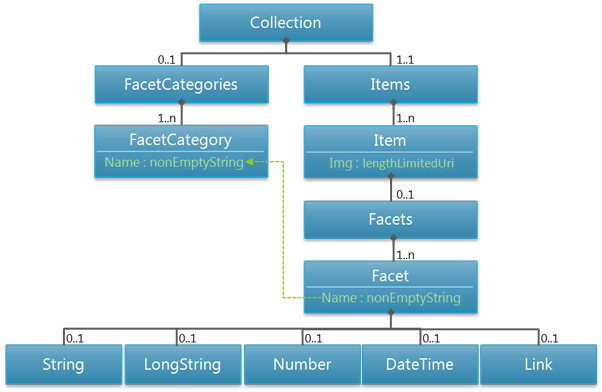
As you can see, a basic collection can be created with a small amount of XML. Much of the work, and much of the schema, revolves around fine-tuning the presentation of the data. Please read below for further details, and also see [Designing Collections](#Design_Process) for tips.

Schema Structure

A collection for the Pivot client consists of CXML and a corresponding image asset collection. The CXML begins with a top-level container called *Collection*. The *Collection* node contains top-level properties and elements which pertain to the entire set of items contained within the collection.

The *FacetCategories* section is a child of the *Collection* node. For each facet attached to an item, a corresponding facet category must exist. Here, the name of the facet, the type, a format string, and various visibility options can be set. The *Type* field can be *Number*, *String*, *LongString*, *DateTime*, or *Link*.

Another child of *Collection* is the *Items* node. Because this is where each individual item in the collection is defined, this node usually contains the majority of the information in the file. Each item can have properties such as a double-click action link, *Href*; a name, *Name*; a description, *Description*; an id, *Id*; an image asset, *Img*; or any number of custom properties declared in the *Facets* section.



Pivot Client Reference

For reference, use the following table to identify the various parts of the Pivot client.



Element Overview

The following table contains a summary of the key elements in CXML, and their attributes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Element Name** | **Type** | **Description** | **Key Attributes** |
| **Collection** | xs:Element | Top-level container for everything in the collection. | * **Name** – Name of the collection which appears in the Pivot title bar. * **Icon** – Icon representing the collection which appears with thumbnails of the collection (in history, for example). * **BrandImage** – Image that appears left-most in the Pivot title bar. Typically used for branding. * **AdditionalSearchText** – Specifies an additional string to append to search queries used for Bing search results in the info panel. E.g. if the item title is "Justin Miller" and the string specified is "baseball player", the Bing query will be "Justin Miller baseball player." To disable Bing queries entirely, set the value to "\_\_block". * **SchemaVersion** – Should match the schema version in the xsd. Optional. |
| **FacetCategory** (child of Collection) | xs:ComplexType | A grouping of facets. Note that all values (see children of *Facet* below) for a *FacetCategory* must be of the same type. | * **Name** – Name of the category which appears in the Pivot filter panel and in the info panel. * **Format** – Optional .NET format string to be used for numeric types. It is recommended that data be rounded/truncated to match the format string to avoid potentially undesired behavior when filtering on that facet in the Pivot Graph View. * **Type** – One of *String*, *LongString*, *Number*, *DateTime*, or *Link*. Note that all values for a *FacetCategory* must be of the same type. * **IsFilterVisible** – Determines whether the category appears in the filter panel. Can only be true for categories of type *String*, *Number*, or *DateTime*. * **IsMetaDataVisible** – Determines whether the category is visible in the info panel. * **IsWordWheelVisible** – Determines whether the category is included in keyword filters over that collection. This attribute can only be true for categories of type *String*, *LongString*, or *Link*. |
| **Items** (child of Collection) | ItemList | Container for all the items in the collection. | **ImgBase** – Specifies a relative path to the Deep Zoom DZC containing this collection’s images. |
| **Item** (child of Items) | xs:ComplexType | An individual item in a collection. | * **Img** – Uri of the image. With Deep Zoom DZCs, uses the URI fragment identifier, "#", as an index into an image collection. E.g. the image with item id x in the DZC specified by the *ImgBase* attribute of *Items* would be accessed with the string "#x." * **Id** – Unique id of this item in the *ItemList*. * **Href** – A URL associated with this item. Double-clicking the item in the Pivot client will navigate the user to this URL. * **Name** – The name of the item which will appear in the info panel. |
| **Description** (child of Item) | xs:String | A text description of an item to be displayed in the info panel. | **Value** - A non-empty string. |
| **Facet** (child of Item) | xs:ComplexType | A container for a value (*String*, *LongString*, *Number*, *DateTime*, or *Link*) of a property (*FacetCategory*) on an item. May be used to filter, sort, or display detailed information about an item. | **Name** – A non-empty string. The name must match the name of the *FacetCategory* for which it contains values. |
| **String** (child of Facet) | StringType | A string facet value. | **Value** – A non-empty string. |
| **LongString** (child of Facet) | StringType | A string facet value that appears in the info panel only. Used for text longer than a short sentence, for optimized multi-line display. | **Value** – A non-empty string. |
| **Number** (child of Facet) | NumberType | A number facet value. | **Value** – An *xs:decimal*. |
| **DateTime** (child of Facet) | DateTimeType | A *DateTime* facet value. | **Value** – An *xs:dateTime*. |
| **Link** (child of Facet) | LinkType | A hyperlink that appears in the info panel only. Used to link to related collection or web-pages. | * **Href** – An URL. Note that this element can be used to achieve the appearance of a mixed-type (strings + links) category by setting some of the *Href* values to "about:none." * **Name** – A non-empty string. |

Schema Details

When creating CXML, the following namespaces should be used:

* xmlns="http://schemas.microsoft.com/collection/metadata/2009"
* xmlns:p=<http://schemas.microsoft.com/livelabs/pivot/collection/2009>

[The PAuthor tool](http://pauthor.codeplex.com/) is an open source tool created by the Pivot team. It is a command line tool used for converting spreadsheets into Pivot collections.

Compression

Given the large amount of text typical in CXML files, it is recommended that files be compressed before being sent over the wire. As such, the mime type should be set to "text/xml." Note that most web servers will compress this mime type by default (including IIS and Apache).

Image Content

Introduction

Every Pivot collection contains a set of images stored in the Deep Zoom format and rendered using Seadragon technology.

The first section of this document provides a quick start guide to build a Deep Zoom image collection for Pivot. The rest of the document describes the Deep Zoom format, discusses the various tools, technologies and techniques for authoring image content, and provides more detailed information on image encoding considerations and deploying image collections on production servers. Sections:

* [Quick Start](#Quick_Start)
* [Deep Zoom Images and Collections](#Deep_Zoom_Images)
* [Deep Zoom Image and Collection Content Creation](#Deep_Zoom_Image)
* [Image Processing Guidelines](#Image_Content)
* [Deploying Deep Zoom Collections](#Deploying_Deep)

Quick Start

There are a number of ways to create the Deep Zoom images that will be part of a Pivot collection. It depends on how the Pivot collection is being created and whether this is a "one off" collection or is part of a production work flow. All of the procedures summarized in this section are described in more detail later in this document.

Manually Creating a Deep Zoom Image Collection

It’s not uncommon to create the Pivot metadata CXML file from content stored in an existing database by implementing the appropriate XML export. In this case, the Pivot Collection Tool for Excel tool is not required, and other tools are used to build the Deep Zoom images used by the Pivot collection. This can be accomplished with [Deep Zoom Composer](http://www.microsoft.com/downloads/details.aspx?FamilyID=457B17B7-52BF-4BDA-87A3-FA8A4673F8BF&displaylang=en) or the [Deep Zoom Tools Command Line Utilities](http://www.seadragon.com/developer), both available as free downloads from Microsoft.

Deep Zoom Composer

Deep Zoom Composer is a graphical utility for creating Deep Zoom Collections. It provides many advanced features, but only a fraction of its capabilities are required to build the images required for a Pivot collection.

1. Import the source images to be included in the collection.
2. Drag the images to the Layout work area. It doesn’t matter how they are laid out.
3. Use Custom Export, specifying Images as the Output Type, and exporting as a collection.

Deep Zoom Tools Command Line Utilities

Deep Zoom Tools Command Line Utilities are included in the free download of the Deep Zoom Tools Library. These two utilities can be automated using conventional shell scripting techniques:

1. Use DZConvert.exe to convert multiple image files into Deep Zoom Images (DZI).
2. Use DZCollection.exe to convert multiple Deep Zoom images into a Deep Zoom Collection.

Custom Applications to Create Deep Zoom Image Collections.

For production work flows, it’s often desirable to create a custom application to generate or update a Pivot Collection. The Deep Zoom Tools Library (Deep ZoomTools.dll) makes it easy to create Deep Zoom Images (DZI) and Deep Zoom Collections (DZC) as part of any custom Windows application. Deep ZoomTools.dll is available as a free download from Microsoft, and includes sample applications with source code.

Deep Zoom Images and Collections

Overview

The image content for a Pivot collection consists of a Deep Zoom Collection of individual Deep Zoom images, all based on Microsoft Seadragon technology. To enable efficient progressive access to large collections of arbitrarily sized images, the Deep Zoom format consists of pre-rendered, tiled image pyramids for both single images and collections. Images must first be converted to this format, and there are multiple tools and techniques to accomplish this.

Each individual item image is encoded in a tiled pyramid as a Deep Zoom Images (DZI). The complete set of Deep Zoom Images included as part of a Pivot collection are encoded as a Deep Zoom Collection (DZC). The DZI and DZC formats are open, and fully documented online on the [Microsoft Developer Network (MSDN) web site](http://go.microsoft.com/fwlink/?LinkId=164944).

|  |
| --- |
| http://ll-pivotweb03/getpivot/images/collections/image049.png |
| Illustration of DZI (left) and DZC (right). |

The following two sections provide a brief overview of the associated file organization.

Deep Zoom Image (DZI)

A Deep Zoom image consists of an XML file plus a subdirectory at the same location and with the same name, plus the suffix "\_files". For example, a DZI named "foo" will consist of the file "foo.xml" and the subdirectory "foo\_files". The XML file may have either the .xml or .dzi extension. The .xml extension is commonly used to accommodate web server requirements. The subdirectory includes multiple subdirectories for each resolution level of the image. Each of these subdirectories contains the individual tile image files for that resolution level.

Deep Zoom Collection (DZC)

A Deep Zoom Collection consists of an XML file for the collection plus a subdirectory at the same location and with the same name, plus the suffix "\_files". The XML file specifies the names of the individual DZI images within the collection. While it is not mandatory, the default location of the collection’s DZI images is a subdirectory at the same location as the CXML file, with the same name plus the suffix "\_images". This subdirectory contains one or more DZI images (XML files plus subdirectories.)

Sparse Images

Single Deep Zoom images can also be "sparse images." A sparse image has different resolutions in different areas, and is typically created by merging together multiple source photos at different scale and/or resolution. Any number of items within a Deep Zoom Collection may be sparse images. Pivot provides full support for sparse images within a Pivot collection.

Deep Zoom Image and Collection Content Creation

Deep Zoom Composer

Deep Zoom Composer is a free tool from Microsoft for creating complete Deep Zoom experiences. It can create Deep Zoom images (DZI), and Deep Zoom Collections (DZC), as well as complete Silverlight Deep Zoom or Seadragon Ajax viewers and the associated web pages. While not required for creating Pivot collections, Deep Zoom Composer can also create fully interactive Silverlight Deep Zoom images and collections using a variety of customizable and pre-built templates.

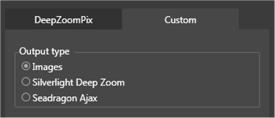
For the purpose of building content for Pivot, only the exported Deep Zoom Collection (DZC) is required.

Deep Zoom Composer can be downloaded for free from the [Microsoft Download Center](http://www.microsoft.com/downloads/details.aspx?FamilyID=457B17B7-52BF-4BDA-87A3-FA8A4673F8BF&displaylang=en).

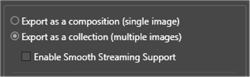
Creating a Deep Zoom Collection

Creating a Deep Zoom Collection for Pivot consists of importing multiple source images, placing the images on the layout panel (the arrangement is completely arbitrary), and then exporting the set of images to a DZC file set.

* Start Deep Zoom Composer and create a new project.
* Select the Import tab and add the image files you want to convert to Deep Zoom Images (DZI).
* Select the Layout tab and drag the source images from the gallery on the left into the layout area. The specific layout is not important; this won’t be used by Pivot.
* Select the Export tab.
* Select the Custom tab on the right panel.
* Select Images as the Output type.



* Specify the name and the directory location for the exported DZI file.
* Select "Export as a collection (multiple images)". This is required to create a Deep Zoom Collection rather than a single Deep Zoom Sparse Image.



* Set the Image Settings as desired.
* Click the Export button.

The specified output directory will contain a subdirectory of the name you specified in the Export options. That directory contains the DZC file named dzc\_output.xml plus the associated collection subdirectories dzc\_output\_files and dzc\_output\_images. You may rename them as desired, as long as they still correspond to required naming relationships. Additionally, the item filename references inside the DZC file will need to reflect the correct location name for the \_images subdirectory.

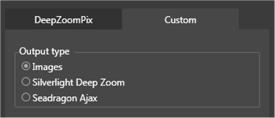
Individual images within a Pivot Collection are referenced by their index within the Deep Zoom Collection. It’s important to review the Deep Zoom Collection (DZC) XML file created by Deep Zoom Composer to insure the image ID references in the Pivot Collection CXML file references the correct image for each collection item.

Creating a Sparse Image

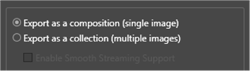
Deep Zoom Composer can also be used to create individual sparse images as DZI files. Other tools are then required to create a Deep Zoom Collection from the individual DZI files.

To create a sparse image DZI file:

* Start Deep Zoom Composer and create a new project.
* Select the Import tab and add the source images that you want to merge together to create the sparse image.
* Select the Layout tab and drag all the desired source images to the layout area. Resize and arrange the images as desired.
* Select the Export tab.
* Select the Custom tab on the right panel.
* Select Images as the Output type.



* Specify the name and the directory location for the exported DZI file.
* Select "Export as a composition (single image)". This is required to create a single DZI image rather than a collection of separate images.



* Set the Image Settings as desired.
* Click the Export button.

The specified output directory will contain a subdirectory of the name you specified in the Export options. That directory contains the DZI file named dzc\_output.xml plus the associated subdirectory dzc\_output\_files. These are the only files required to display the Deep Zoom Image using Seadragon Ajax. You can rename them as desired, as long as the file directory is in the same location as the .xml file and is the same name plus the "\_files" suffix.

Deep Zoom Tools

Deep ZoomTools.dll is installed as part of Deep Zoom Composer. This library can also be downloaded along with sample applications and source code from the [Seadragon developer web page](http://www.seadragon.com/developer).

Deep ZoomTools.dll provides a powerful and simple to use method to incorporate Deep Zoom Collection creation within your own custom content management applications.

In addition to the information on the Seadragon developer’s web page, you can read more about Deep ZoomTools.dll in this [blog post](http://blogs.msdn.com/expression/archive/2008/11/26/hello-DeepZoomtools-dll-deep-zoom-image-tile-generation-made-easy.aspx) from the Expression team. Additionally, a simple [Bing search](http://www.bing.com/search?q=Deep%20Zoom%20tools.dll) will uncover several additional examples of developers using Deep ZoomTools.dll to create Deep Zoom content.

Deep Zoom Tools Command Line Utilities

The Seadragon team has made available a pair of sample applications for Deep ZoomTools.dll, providing the ability to easily build Deep Zoom Images (DZI) and Deep Zoom Collections (DZC) from the command line. The Deep Zoom Tools Command Line Utilities, including the full source code, are available as a free download from the [Seadragon developer’s web page](http://www.seadragon.com/developer).

As a parameter driven command line tools, the Deep Zoom Tools Command Line Utilities can be used to easily automate repetitive content creation work flows.

There are two separate utilities. DZConvert.exe converts one or more image files to Deep Zoom Image (DZI) files. DZCollection.exe converts one or more Deep Zoom Images (DZI) into a Deep Zoom Collection (DZC.)

DZConvert.exe converts single images into the equivalent Deep Zoom Image (DZI). Deep Zoom Composer must be used to create sparse images.

DZCollection.exe provides an extremely efficient method to build or re-build a Deep Zoom Collection when adding or removing images, or when only a subset of the images needs to be updated. It eliminates the bulk of the work generating the individual image tile pyramids.

Because sparse images are typically built as individual Deep Zoom Images (DZI), separately from building a Deep Zoom Collection (DZC), DZCollection.exe is an ideal solution for creating a Deep Zoom Collection from previously created sparse images.

Individual images within a Pivot Collection are referenced by their index within the Deep Zoom Collection. It’s important to insure the Pivot Collection CXML file references the correct image for each collection item. DZCollection.exe builds the collection in the order it retrieves the source Deep Zoom images, typically sorted by filename within the specified wildcard set. DZCollection.exe also provides a verbose report option, listing the pertinent information about each item added to the collection. This report can be saved to a file via command line redirection to aid in subsequent automated processing steps.

http://ll-pivotweb03/getpivot/images/collections/up_arrow.png [Back To Top](http://ll-pivotweb03/getpivot/developer-info/image-content.aspx#top)

Image Processing Guidelines

File Format

The tile pyramids for Deep Zoom images and Deep Zoom collections can be encoded as JPEG, PNG or HD Photo files. Each format has its own advantages and disadvantages, so it’s important to select the most appropriate format based on the source image content.

JPEG

For most scenarios, JPEG is the appropriate file format. JPEG is the de facto standard for web images, as it provides the greatest level of compatibility with the overall image and web ecosystem. JPEG also provides effective compression efficiency for continuous tone images, such as photographs or drawings with shading and gradients.

JPEG provides configurable lossy compression. There will always be some degradation of the source image, but depending on the image compression settings (discussed below) this degradation is often completely undetectable.

PNG

PNG is a lossless image format. It provides very inefficient compression compared to JPEG for continuous tone images, but often provides as good or better compression for simple diagrams, text, or other images with single color backgrounds. For high contrast content (like text and line drawings), PNG eliminates the harsh artifacts that are typical when using JPEG compression for this type of content.

PNG also supports an optional alpha channel, making it possible to present images with transparency. There are limited uses for this in typical Pivot collections, but it can be used to provide non-rectangular or partially transparent items images.

However, adding an alpha channel increases the overall image file sizes, and using PNG for continuous tone images (such as photographs) can significantly degrade the overall collection experience due to the slower performance associated with the larger files. This also increases server storage requirements, deployment time and bandwidth costs.

Image Compression

The overall performance and bandwidth costs and deployment time for any Deep Zoom Collection experience can be optimized by choosing the most appropriate image compression settings. Increasing the amount of compression will produce smaller image tile files that transfer faster and decoder quicker. But too much compression may adversely impact the quality of the displayed images.

PNG offers perfect image reproduction (lossless compression), but is extremely inefficient for photos and other continuous tone images. PNG is typically the best choice for line drawings, text or other images with high contrast edges and monochromatic backgrounds. There are no options to adjust PNG’s compression. It’s a fixed algorithm and the amount of compression depends on the image content.

JPEG offers a range of lossy compression, expressed as a compression setting from 1-100. The higher the value, the smaller the amount of compression and the higher quality the reproduction compared with the original source.

A typical value for JPEG compression is 80. Using a compression setting greater than 90 will invariably create much larger files with no noticeable increase in quality. For many scenarios, values of 70 or lower can be used and still provide perfectly acceptable quality images while significantly reducing the size of the individual image tile files.

The best approach is to experiment with different settings, carefully evaluating the image quality while also comparing the resulting file size. A modest change in the compression parameter may cut the file size by a factor of two or more with no noticeable impact on image quality. There are no absolute rules here. The results depend on the source image content so the only guaranteed methodology is experimentation.

For high contrast content (line drawings, text, etc.) higher quality settings will be required. The compression efficiency may be less than for continuous tone images, but will typically be significantly better than using PNG for the same content.

Again, successfully optimizing the compression settings for any particular content requires experimentation. There is no "magic rule" that will work with all content types. It’s important to try a variety of settings and find the best compromise between file size and image quality for each specific scenario.

Color Profiles

Unless otherwise set by the content author, most image files contain an sRGB color profile definition, or in the absence of a specific color profile definition, are assumed to be rendered in the sRGB color space.

Deep Zoom rendering in Pivot assumes that all image tiles are based on an sRGB color profile. If image or collection tiles are created based on a different color profile, they will not render correctly in Pivot.

Deep ZoomTools.dll as well as Deep Zoom Composer (which uses Deep ZoomTools.dll) will respect the color profile of the source image and correctly convert to sRGB when building a Deep Zoom image or collection.

If any custom tools are used to generate Deep Zoom tile sets rather than using Deep ZoomTools.dll, those tools should also detect and respect the source image color profile and apply the appropriate conversion to sRGB.

DZC Format vs. DZI Image Format

Within a Deep Zoom Collection used in Pivot, it’s technically possible for the image pyramids for the individual Deep Zoom Images (DZI) to use different image file formats (JPEG, PNG) as well as different compression settings.

To maximize compatibility with other implementations of Seadragon Deep Zoom technology, a consistent file format for the image tiles for all Deep Zoom Images (DZI) in the collection, as well as the image tiles for the Deep Zoom Collection (DZC), is recommended.

It is perfectly acceptable, and often desirable, to use different lossy compression settings for individual items in a collection, or for the collection tile pyramid vs. the image tile pyramids. For example, it may be desirable to use a higher quality setting for the collection tile pyramid, minimizing any visible compression artifacts for the low resolution image levels. However, because compression artifacts are typically less visible at higher resolutions, a lower quality setting can be used for the individual image tile pyramids, reducing storage requirements and improving performance. Because the collection level tile set is typically a small percentage of the overall collection size, the cost of higher quality compression at this level is minimal. Just keep in mind that the image tiles for the Deep Zoom collection are typically created from the image tiles of the individual Deep Zoom images, so a huge difference between the quality settings may not offer any benefits. As always, a few experiments can help discover the most effective settings for any particular image collection.

Deploying Deep Zoom Collections

Overview

Deep Zoom Collections consist of nothing more than XML and image files, and are accessed from the client using conventional http protocols. No special web services beyond a simple file store are required to deploy Deep Zoom image collections.

There are some considerations when deploying large Deep Zoom Collections based on the sheer volume of content involved. Building tile pyramids can create a lot of files. A Deep Zoom Collection created from hundreds or thousands of high resolution images can result in tens or hundreds of thousands of individual image tile files. For example, a collection containing 600 six megapixel images will consist of over 140,000 individual files in over 14,000 subdirectories.

While the total disk space consumed by those files is only incrementally larger than the original source images (typically 1.4x, assuming the same compression ratio), moving hundreds of thousands of files can be challenging. There are a number of techniques that can be used to minimize the work required to deploy Deep Zoom Collection content to a production web server.

Collections consisting of lower resolution images will have a much higher ratio of directories to files because there will be fewer directories containing large numbers of image tile files. For example, a collection created from 2,600 ¼ megapixel images will consist of 51,000 files in 32,000 subdirectories.

Generate in Place

One straightforward technique that may be appropriate for certain production scenarios is to avoid the deployment step completely by generating the Deep Zoom Image (DZI) and Deep Zoom Collection (DZC) content directly on the production server. While this approach may not be viable for all scenarios, it can be very efficient for a well-established production configuration to perform the image content creation tasks directly on the origin server, completely avoiding the need for additional steps to move the content to its hosted location.

A typical implementation involves staging and deploying source image content, or an appropriate manifest with links to the source image content, and then running a production server hosted process to create the Deep Zoom Collection in place from the source images.

It may also be advantageous to run the content creation process on local machines, but save directly from the content creation application to the target deployment server. The goal is the same: eliminate unnecessary copies for large numbers of files and directories.

This technique is not appropriate in cases where it is not feasible to run the content creation process on the production server, when it’s required to pre-stage content for test or verification prior to deployment, or when significant replication beyond the origin server is required.

Content Reduction

There are a few techniques that can be performed to reduce the number of files that need to be deployed.

The tip of each Deep Zoom Image (DZI) pyramid duplicates the content that is included in the Deep Zoom Collection (DZC) image pyramid. In cases where the images are always accessed as elements within a collection, the files and directories that make up the tip of each DZI pyramid are redundant and can be eliminated. In a typical collection, this can eliminate up to nine files and nine directories for each item in the collection. While it may be a small number of the total files, this can eliminate a significant number of the directories that must be created and copied. In the example above with ¼ megapixel images, this technique would eliminate over 80% of the required directories and over 40% of the image tile files.

While Seadragon Deep Zoom technology allows arbitrarily large images to be included in Deep Zoom Collections, scenarios that use low resolution images may be able to completely eliminate the individual Deep Zoom Image (DZI) files and store all of the image data in the Deep Zoom Collection.

Various Deep Zoom content creation tools provides flexibility in setting the dimensions for image tiles. Using a larger image tile size can reduce the number of files required. Depending on the content type, this may create an undesirable increase in tile load latency, so this capability should be used carefully. But for content material that provides efficient image compression, using larger tile sizes can significantly reduce the number of individual files and directories required for deployment.

Generate to ZIP

A custom content generation work flow can generate Deep Zoom image and Deep Zoom collection content directly to a ZIP file rather than to individual image tile files. Significantly less overhead is required to deploy a small number of large ZIP files to the production server vs. tens or hundreds of thousands of individual image files. However, this custom work flow would also require a process on the production server to unpack the ZIP file into the required directory structure.

Updating a Collection

A Deep Zoom Collection should not be updated in place. If any users are currently viewing the collection, their current collection navigation is based on information from the CXML file, cached locally by the application. Changing the collection contents will invalidate this cached information, most likely causing the user to experience inconsistent and undesirable results.

To correctly update a Deep Zoom collection, create an entirely new collection, deployed at a different URL. This URL should contain a unique identifier, such as a version number or deployment date. The web page hosting the collection can then be updated to reference the new collection URL. Users currently accessing the web page will continue to be served by the old version. New requests to the web page will receive Deep Zoom content from the new URL. For a Pivot collection, the URL for the Pivot CXML can remain the same; only the Pivot CXML file needs to be updated to reference the URL for the new Deep Zoom collection. Again, users currently accessing the Pivot collection will continue to receive content from the previous Deep Zoom collection, but anyone reloading the Pivot CXML file will now correctly receive the updated Deep Zoom content that matches the updated Pivot CXML file.

As long as there are no changes to individual Deep Zoom images within a collection, only the Deep Zoom Collection needs to be updated. New Deep Zoom image files can be deployed and others can be deleted, but only the Deep Zoom Collection needs to be updated. As long as existing Deep Zoom images aren’t modified, they do not have to be deployed again. This can be dramatically more efficient than recreating and redeploying all of the individual Deep Zoom images. It also allows Pivot (and other Deep Zoom applications) to reuse any Deep Zoom image tiles that are already cached locally.

If you are ONLY adding new Deep Zoom images to a Deep Zoom collection, not removing or changing any images, or making any changes to the order of the images within the collection, you can simply overwrite the old collection with the new one. As long as the Deep Zoom Collection (DZC) XML file and the Pivot Collection CXML files are the last files updated, this in-place update will be transparent. Existing users will continue to access the previous set of images; newly added images will not interfere. When the collection is reloaded, then the new images will be available.

Caching a Collection

A Deep Zoom collection can be treated as entirely static content by the deployment web server. To maximize efficient access to this content, it’s important to set aggressive caching options. For instance, the full collection of files that make up a Deep Zoom Collection should have a cache setting to never expire. This will significantly reduce repeat content access requests from clients (including Pivot or Silverlight Deep Zoom), as well as intermediate proxy servers. Instead, repeat content requests can largely be served from local and intermediate proxy caches, improving performance while reducing both server load and bandwidth costs.

The XML files used by Deep Zoom and Pivot collections will also benefit from HTTP compression. Most web servers have a way to enable HTTP compression. This compression option should be enabled for any XML file types used for Deep Zoom and Pivot, including .XML, .DZC, .DZI and .CXML. Options should also be enabled to ensure these files are compressed once, on first use, since their content won’t be changing dynamically.

Copying files

When there is no choice but to copy the individual files to a deployment server over a network connection, there are a few techniques that streamline this process.

Windows includes ROBOCOPY, a command line file copy utility that is the recommended tool for copying large numbers of files and directories locally or across network connections. ROBOCOPY takes full advantage of all system and network resources for multiple copy threads, and has great options to mirror entire directory structures and intelligently restart a previously interrupted copy without the need to retransfer unchanged files.

If both the source and destination computers are running Windows Vista (SP1 or later), Windows 7, Windows Server 2008 or Windows Server 2008 R2, full SMB2 support is available. In this environment, the built in file copy features in Windows Explorer are also more efficient than was the case with previous versions of Windows or when copying in a heterogeneous operating system environment. While ROBOCOPY is still the preferred tool, this can be handy for quickly copying smaller Deep Zoom images or collections.

Performance may also be significantly improved by temporarily disabling anti-virus software, eliminating unnecessary processing of the individual files being copied.

Similar to generating content to a ZIP file discussed earlier, another option for deploying an existing Deep Zoom Collection is to compress the directory structure to one or more ZIP files, copy the ZIP files to the server, and then unpack the ZIP files on the server back into the original directory structure. Depending on the deployment server environment and network and computer performance, the end-to-end time for this work flow may be more efficient than copying all the content as individual files and directories.

Summary

Deep Zoom image collections for Pivot are deployed using an open, fully documented format, built on industry standard content elements. This same format is shared across a variety of Seadragon-powered solutions from Microsoft, powering the next generation of web experiences.

Microsoft provides a variety of free tools to author image content for Pivot collections, including an easy-to-use-library for custom work flow solutions. The open nature of the Deep Zoom format makes it straightforward to build Pivot collection authoring applications for any platform or development environment.

# Collection Hosting

Introduction

Once a collection is built, it must be hosted on a server. This page contains information about the various ways to host collections in the following sections:

* [Simple and Linked Collections](#Simple)
* [Dynamic Collections](#Dynamic)
* [Pivot Team's Dynamic Collection Implementation](#Pivot)

There are three primary types of collections:

* **Simple** collections containing up to 3,000 items. In these collections, the user's experience is defined by previously generated (or *static*) XML.
* **Linked** collections, generally containing several thousands of items. These collections consist of multiple inter-linked *simple* collections.
* **Dynamic** collections, unbounded in size. In these collections, the user's experience is defined by XML generated *dynamically* in response to user action.

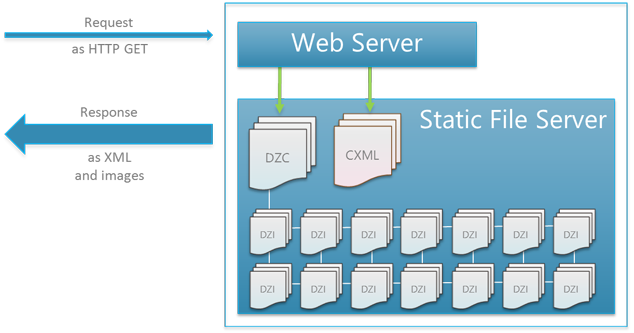
From the hosting perspective, *simple* and *linked* collections are the most straightforward, as they require only a static file server. *Dynamic* collections, on the other hand, require a more complex server structure for generating XML and image content on request. What follows is a brief description of how to host *simple* and *linked* collections, followed by in-depth discussion on the more complex hosting of *dynamic* collections.

Simple and Linked Collections

Serving Files

Because *simple* and *linked* collections are static files, they can be hosted on a simple file server. On each request, the server sends the Pivot client CXML or image files. In both *simple* and *linked* scenarios, the client will only request one CXML at a time. The distinction between the two is only that in the *linked* scenario, a user may navigate between different CXML files via links in the info panel (see [Collection XML Schema](#XML_Schema)). As such, there is no special handling of *linked* collections on the server. The only potential difference is that the file server may need to host multiple CXMLs (one for each *simple* collection composing the *linked* collection).

See the following illustration of a static server:



Tips:

* Due to the large amount of text common in some collections, server-side compression should be enabled and the CXML mime type should be set to "text/xml."
* Server-side caching and CDNs can be leveraged to further optimize performance.
* For tips on optimizing deployment of the large number of image files in a collection, see [Collection Image Content](#Image_Content).

When Is a Simple or Linked Collection Appropriate?

Due to their simplicity, *simple* and *linked* collections are significantly easier to create and host than *dynamic* collections. If a data set is small (3000 items or less) and relatively static, it is probably most appropriate as a *simple* collection. As *simple* collections get larger, however, client performance and bandwidth become limiting factors. These factors eventually provide an upper bound to the size of the collection.

One way to get around collection size limitation is to split a larger data set into a number of inter-linked *simple* collections, thus creating a *linked* collection. With *linked* collections, a large data set can be presented to the user in up to 3000 item segments. Navigation between these segments (each a *simple* collection) happens via the *Link* and *Related Link* facet types (see [Collection XML Schema](#XML_Schema)), or with item click-through links. If a data set numbers in the several to tens of thousands, is relatively static, and lends itself naturally to inter-linking between different parts, it is probably most appropriate as a *linked* collection.

While *linked* collections enable a larger data set to be represented, when they grow very large, their complexity can become problematic. Key difficulties encountered when creating *linked* collections of very large data sets include:

* **Storage** - Large *linked* collections can result in very large numbers of image files. Due to likely overlaps among different segments of a *linked* collection, the number of image files can quickly multiply into the many millions.
* **Updates** - If the collection content rarely changes, a large number of image files may not pose a major problem. The addition or removal of an item, however, often requires large portions of the Deep Zoom image tile pyramid to be regenerated (read more about Deep Zoom tile pyramids here: [Collection Image Content](#Image_Content)). As a result, maintaining a large collection that needs to be updated frequently can be cumbersome.
* **User Experience** – Great *linked* collections can be difficult to create when the data set at hand does not lend itself naturally to division into smaller segments. They are also limited in their ability to respond to un-anticipated queries, and do not naturally support search across their segments.

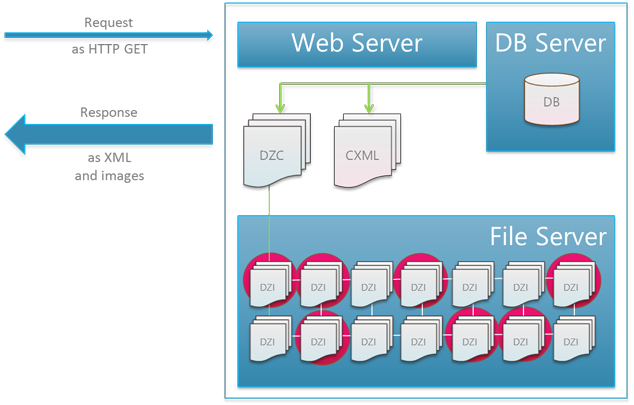
When these difficulties are encountered, *dynamic* collections offer some solutions.

Dynamic Collections

Introduction

To support scenarios beyond those possible with *linked* collections, a *dynamic* collection can be used. Although any number of dynamic architectures is conceivable, the remainder of this document will focus on the approach taken by the Pivot team to create a Wikipedia collection.

Dynamic collections follow a similar sub-collection methodology to *linked* collections, but use a dynamic server structure to mitigate the issues raised in the previous section. In *dynamic* collections, the CXML and corresponding Deep Zoom image collection is generated by a web server at request time. The following figure illustrates a sample architecture:



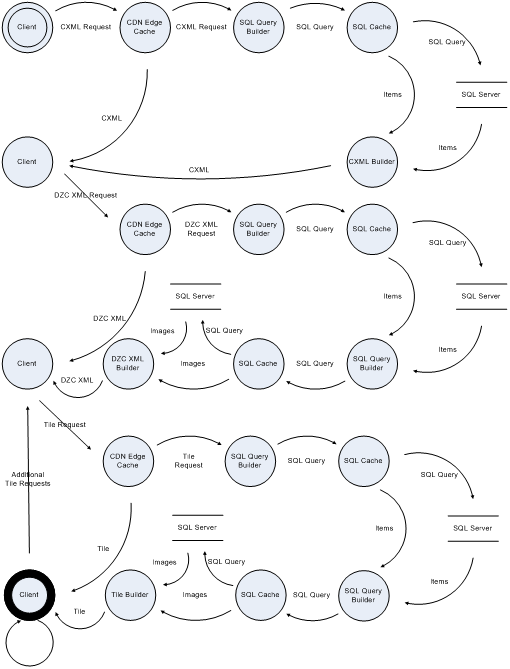
Although more complex to construct, *dynamic* collections have the following advantages:

* **Storage & Updates** - Because much of the CXML and image content is created at request time, file proliferation is less of an issue, and updates can be made by changing only a small number of files (or tables).
* **User Experience** – With a more flexible structure, *dynamic* collections enables scenarios such as search and serving of un-anticipated query patterns. Further, because the cost of duplicating items in various sub-collections is mitigated in this architecture, inter-linking between sub-collections can be much more complex.
* **Enabling Large Existing Data Stores** – Many existing large data sets already have dynamic data stores and pipelines, which can be leveraged to present the data in the Pivot collection format.

Pivot Team's Dynamic Collection Implementation

Overview

At first inspection, building sub-collections in real-time on a web server may seem straightforward. Achieving good performance and scalability properties, however, is not without its nuances. The figure below shows the request flow between the client and web server for the Wikipedia collection.



Request Handlers

The three types of request handlers required are:

* a CXML handler,
* a DZC handler, and
* a tile handler.

The CXML handler is responsible for building the primary CXML with facet definitions, item metadata and a link to the Deep Zoom image collection based on some query which defines the sub-collection. We used querystrings on the CXML files to represent client queries into the system. Note that when handling a tile request, Deep Zoom always asks for a tile in the form "*<DZC name>\_files/<level>/<x>\_<y>.<format>*". A URL rewrite rule is used to map that URL format into a dynamic server request with appropriate query parameters.

One of the most important steps in the process is building collection tiles on demand. The tile building process must determine, based on *level*, *x*, and *y*, what items appear on the requested tile. It must then downsize the raw imagery for those items to the appropriate size for *level* and place them in the tile according to appropriate Morton layout relative to the item’s position in the results set returned during the DZC step. Currently, JPEG’s are the preferred image to use for the raw image, since downsizing and decoding can happen in the same step given the nature of the Discrete Cosine Transforms by which the JPEG is compressed. For more information on image formats, see [Collection Image Content](#Image_Content).

The query often has to be enacted on a data storage system of some kind. In many cases, a relational database such as SQL server will suffice for this purpose. The only caveat is that raw imagery for each item will need to be stored for quick retrieval during the tile stitching phase.

Performance Optimizations - Request Caching

A careful examination of the system will reveal a nuance of great importance. Each request type, or even requests within the same type will often reuse the same piece of data. As an obvious example, consider the CXML and DZC building steps. To construct the CXML, items which match the query are found, their metadata serialized and then returned to the client. To construct the DZC, items which match the query are found, image information examined and then serialized. The step of finding items which match the query is duplicated. Likewise, for building tiles, at different levels, raw imagery is often reused from level to level, so fetching it repeatedly from disk would be wasteful. To prevent executing duplicate operations, request affinity and application server caching can be used, as seen in the figure above. Request hashing is a good choice for affinity, as it has the potential to group like queries together.

Since collection building is not computationally cheap, a content delivery network (CDN) can be used to group like requests and prevent re-computations of the same most visited sub-collections. If usage of a CDN is not available, web server response caching can be used.

Performance Optimizations – Image Tiles

The first few image pyramid tiles have the ironic property of being the smallest information gain for the client and the most difficult for the server to build. However, for client animation reasons, they are important. To circumvent the difficulty associated with building these first few tiles, the dominant color for each item can be pre-computed and then used. This allows the server to avoid loading all images into memory to render the first tile. However, the server should still proactively load the raw source images required to build the tile pyramid into memory. Starting a background operation as early as CXML request time is best.

Another implementation detail involves whether to build the entire image pyramid after the first request, or per tile request. It is advantageous from a server resource standpoint, and ultimately from an end-user standpoint, to build each tile per request. This is because the whole pyramid is rarely requested, so building the extra unused tiles may be a waste. In the event that the whole pyramid is requested, it is often requested in a timeframe slower than what the server is capable of generating the pyramid in, so resources are unnecessarily consumed.

Tile building can also be made much less computationally expensive by creating tiles from raw images, with a technique called macro-blocking. A JPEG image is primarily compressed by a Huffman encoding, followed by a discrete cosine transform of 8x8 portions of the image, referred to as sub-images. Since the images to be placed on the resulting pyramid tile above the 2nd level are 8x8 or some factor greater, it is possible to only undo the Huffman compression on the raw images, place the images accordingly on the resulting pyramid tile, and then re-apply the Huffman encoding.