Getting the news using The Guardian's API on Windows Phone

The Guardian newspaper provides an open platform API so that the published articles (after 1999) can be accessed and used in any application. This article explains how to get latest news and articles using the API.

Introduction

The Guardian API has four building blocks which form the basis of how to fetch published articles.

- **Content Search** - The basic search based on keywords. Results/News are provided in the form of articles lists.
- **Section Search** - All the articles published in The Guardian lie under some section. We can choose section(s) under which we would like to do basic content search.
- **Tag Search** - Articles which are tagged by some keyword can be search by this kind of search operation.
- **Item Search** - Item search gives the detail information available for a single content.

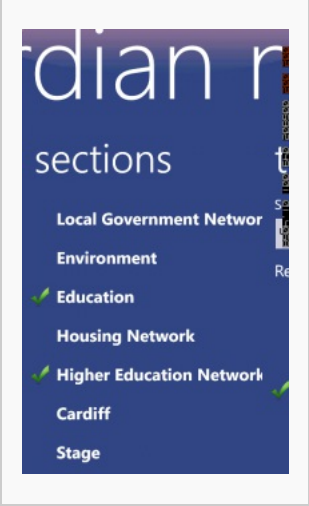
In this article, we will make use of first three to get news based on various parameters. We will make a panorama page having three portions for the basic search/content search, sections search and tags search. On the basis of these filters, we will open our listing page which will show searched articles. Then we can view the detail of any selected item using Web Browser control.

The screens in this project are as shown below:

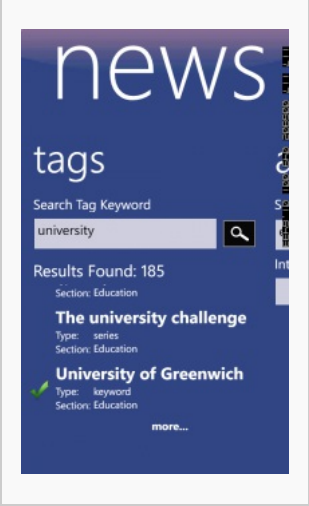
Default view of screen to search Contents

Default view of screen to search Sections

Default view of screen to search Tags



Some Sections selected



Some tags selected

Screen from where content search is performed

(C) Copyright Nokia 2014. All rights reserved.

Search results listing screen

Article's detail screen

Prerequisites

- For this particular article, it is not mandatory to register for an application key for *The Guardian* API. A key is required in order to get detailed information related to the searched items - it is available from here.
- To use this article, we need to have a reference to **The Windows Phone Toolkit** which can be installed by NuGet Manager. To do so, in the Solution Explorer | Manage Nuget Packages >> search for **wptoolkit** and install it.

Note: The Guardian API provides all data free of cost. The detailed amount of data we can fetch depends on whether we are using application key or not. The API does has a limit on API calls per month but if you are willing to have an app based on it, then the limits can be increased for individual publishers.

References Required

Following references to the DLLs are required.

- Microsoft.Phone.Controls.Toolkit
- PhonePerformance
- System.Runtime.Serialization
- System.Servicemodel
- System.Servicemodel.web

UI

- HomePage - A panorama control having three panorama items will be used. In the first one, a text box and couple of date pickers would be placed. Any item can be searched by writing some text only. To reduce the search results, the time intervals can be mentioned. In the second one, a simple list of all the sections available will be placed. On selecting among the items, an icon of tick mark will be visible. In the third and last panorama item, a text box will be placed to search related tags. The successful results will be shown in the list box beneath it.
- CommonList- To make all the lists (tags/sections etc.), we have used User Control defined in this file.
- ArticlesListingPage- In this page, we will show total number of results received and initial 10 results in a list by default. At the bottom of the list, we have an option *more*, by pressing which next 10 items will be fetched and added in the current list. This process can be repeated itself until all the received results aren't viewed.
- ArticleDetailPage- In this page, we will simply show the detail of any article using a web browser control.

Note: Detailed code snippet can be checked in the attached source code file.

Parsing the JSON response

This section demonstrates how to parse the JSON response received by the *content search*. *Tags* and *Sections* search are parsed similarly (see the attached source code for detail).

Data Structures

When we make a hit for *content search*, the structure of the response we get starts from the root node named `response` (declared in **ContentMainResponse.cs**):

```
[DataContract]
   public class ContentMainResponse
   {
```

 (C) Copyright Nokia 2014. All rights reserved.

```
        [DataMember(Name = "response")]
        public ContentResponse Response { get; set; }
    }
```

The `ContentResponse` used in `response` (declared in **ContentResponse.cs**) has setters and getters for the main elements returned by the query (status, number of pages, current page, etc.):

```
  [DataContract]
    public class ContentResponse
    {
        [DataMember(Name = "status")]
        public string Status { get; set; }

        [DataMember(Name = "userTier")]
        public string UserTier { get; set; }

        [DataMember(Name = "total")]
        public string Total { get; set; }

        [DataMember(Name = "startIndex")]
        public string StartIndex { get; set; }

        [DataMember(Name = "pageSize")]
        public string PageSize { get; set; }

        [DataMember(Name = "currentPage")]
        public string CurrentPage { get; set; }

        [DataMember(Name = "pages")]
        public string Pages { get; set; }

        [DataMember(Name = "orderBy")]
        public string OrderBy { get; set; }

        [DataMember(Name = "results")]
        public List<ContentObject> ContentObjList { get; set; }
    }
```

Note that the results data member is a list of `ContentObject` which define each individual article (defined in **ContentObject.cs**):

```
  [DataContract]
    public class ContentObject
    {
        [DataMember(Name = "id")]
        public string Id { get; set; }

        [DataMember(Name = "sectionId")]
        public string SectionId { get; set; }

        [DataMember(Name = "sectionName")]
        public string SectionName { get; set; }

        [DataMember(Name = "webPublicationDate")]
        public string WebPublicationDate { get; set; }

        [DataMember(Name = "webTitle")]
        public string WebTitle { get; set; }

        [DataMember(Name = "webUrl")]
        public string WebUrl { get; set; }

        [DataMember(Name = "apiUrl")]
        public string ApiUrl { get; set; }
```


(C) Copyright Nokia 2014. All rights reserved.

```
        [DataMember(Name = "fields")]
        public ContentThumbnail Fields { get; set; }
    }
```

Last of all, each article (`ContentObject`) has a thumbnail (`ContentThumbnail`), which is declared in **ContentThumbnail.cs**:

```
[DataContract]
    public class ContentThumbnail
    {
        [DataMember(Name = "thumbnail")]
        public string ThumbnailUrl { get; set; }
    }
```

Files Used

All the files used for the different types of web services hits in this app are listed below:

- ContentObject.cs
- ContentResponse.cs
- ContentThumbnail.cs
- SectionMainResponse.cs
- SectionObject.cs
- SectionResponse.cs
- TagObject.cs
- TagsMainResponse.cs
- TagsResponse.cs

API Usage

Sections Search

As soon as the Home page loads, we are fetching the list of all available sections. The code snippet is as below.

```
public void getSectionsList()
        {
            HttpWebRequest httpReq = (HttpWebRequest)HttpWebRequest.Create(new
Uri(AppConstants.sectionsBaseUri));
            httpReq.BeginGetResponse(HTTPWebRequestSectionsCallBack, httpReq);
        }

        private void HTTPWebRequestSectionsCallBack(IAsyncResult result)
        {
            string strResponse = "";
            try
            {
                Dispatcher.BeginInvoke(() =>
                {
                    HttpWebRequest httpRequest = (HttpWebRequest)result.AsyncState;
                    WebResponse response = httpRequest.EndGetResponse(result);

                    Stream stream = response.GetResponseStream();
                    StreamReader reader = new StreamReader(stream);
                    strResponse = reader.ReadToEnd();
                    if (strResponse.Trim() != "")
                    {
                        parseSectionsResponseData(strResponse);
                    }
                });
            }
            catch
            { }
        }
```

After receiving and saving the Section's data we have to parse it. After parsing Sections data, we are setting it to the target list box. On tapping the sections, we need to save their texts as multiple sections names can be used to search any content on The

 (C) Copyright Nokia 2014. All rights reserved.

Guardian. If we want to search some text based on section 1 or section 2 then we have to bind them and send in our web request as section 1 | section 2. The code snippet of sections list box selection is as shown below:

```
    private void sectionsList_SelectionChanged(object sender, SelectionChangedEventArgs
e)
        {
              if (sectionsList != null && sectionsList.SelectedIndex >= 0)
              {
                  CommonList data = (sender as ListBox).SelectedItem as CommonList;

                  int listSelIndex = sectionsList.SelectedIndex;
                  Boolean flagIsAlreadySelected = false;

                  // if index is already selected, then deselect it
                  for (int i = 0; i < _prevSelSectionIndexArr.Count; i++)
                  {
                      if (listSelIndex == _prevSelSectionIndexArr.ElementAt(i))
                      {
                          _prevSelSectionIndexArr.RemoveAt(i);
                          _sectionIdsArrToBeSearchIn.RemoveAt(i);
                          data.imgSectionTickIcon.Visibility =
System.Windows.Visibility.Collapsed;
                          flagIsAlreadySelected = true;
                          break;
                      }
                  }

                  // if not selected earlier then select it
                  if (!flagIsAlreadySelected)
                  {
                      flagIsAlreadySelected = false;
                      data.imgSectionTickIcon.Visibility =
System.Windows.Visibility.Visible;
                      _prevSelSectionIndexArr.Add(listSelIndex);
                      _sectionIdsArrToBeSearchIn.Add(data.txtSectionId.Text.Trim());
                  }

                  sectionsList.SelectedIndex = -1; //resetting index to -1
              }
        }
```

Above, we are just checking if the item is already selected. If yes, it is deselected else selected and it's value is saved in a String type array.

Tags Search

To get desired tags, we had created a text box and a search button. On tapping the search button, tag search will start. The code snippet is shown below:

```
         public void getTagsList(String aKeyword, int aPageNumber)
        {
            HttpWebRequest httpReq = (HttpWebRequest)HttpWebRequest.Create(new
Uri(AppConstants.tagsBaseUri + "q=" + aKeyword + "&page=" + aPageNumber));
            httpReq.BeginGetResponse(HTTPWebRequestTagsCallBack, httpReq);
        }

        private void HTTPWebRequestTagsCallBack(IAsyncResult result)
        {
            string strResponse = "";
            try
            {
                Dispatcher.BeginInvoke(() =>
                {
                    HttpWebRequest httpRequest = (HttpWebRequest)result.AsyncState;
                    WebResponse response = httpRequest.EndGetResponse(result);
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    Stream stream = response.GetResponseStream();
                    StreamReader reader = new StreamReader(stream);
                    strResponse = reader.ReadToEnd();
                    if (strResponse.Trim() != "")
                    {
                       parseTagsResponseData(strResponse);
                    }
                });
            }
            catch
            { }
        }
```

By default 10 initial results will be shown. The facility to view *More* items is also available in this list. The code snippet is as below.

```
 private void lst_tagsResult_SelectionChanged(object sender, SelectionChangedEventArgs
e)
        {
            CommonList data = (sender as ListBox).SelectedItem as CommonList;

            if (lst_tagsResult != null && lst_tagsResult.SelectedIndex >= 0)
            {
                try
                {
                    if (data.txtTagWebTitle.Text.Contains("more"))
                    {
                        int tagsListSelIndex = lst_tagsResult.SelectedIndex;
                        try
                        {
                            _tagsResultList.RemoveAt(tagsListSelIndex);
                        }
                        catch
                        { }

                        getTagsList(_aTagWordToSearch, ++_aTagsPageNumber);
                    }
                    else
                    {
                          int listSelIndex = lst_tagsResult.SelectedIndex;
                          Boolean flagIsAlreadySelected = false;

                          // if index is already selected, then deselect it
                          for (int i = 0; i < _prevSelTagsIndexArr.Count; i++)
                          {
                              if (listSelIndex == _prevSelTagsIndexArr.ElementAt(i))
                              {
                                  _prevSelTagsIndexArr.RemoveAt(i);
                                  _tagsIdsArrToBeSearchIn.RemoveAt(i);
                                  data.imgTagTickIcon.Visibility =
System.Windows.Visibility.Collapsed;
                                  flagIsAlreadySelected = true;
                                  break;
                              }
                          }

                          // if not selected earlier then select it
                          if (!flagIsAlreadySelected)
                          {
                              flagIsAlreadySelected = false;
                              data.imgTagTickIcon.Visibility =
System.Windows.Visibility.Visible;
                              _prevSelTagsIndexArr.Add(listSelIndex);
```

 (C) Copyright Nokia 2014. All rights reserved.

```
                                _tagsIdsArrToBeSearchIn.Add(data.txtTagId.Text.Trim());
                            }

                            lst_tagsResult.SelectedIndex = -1; //resetting index to -1
                        }
                    }
                    catch
                    { }
                }
            }
```

Now the code of panorama item 2 and 3 is explained. We will now move to panorama item 1. Here, we can select the time duration to filter down our search results. The code snippet is shown below.

```
   private void datePicker_To_ValueChanged(object sender, DateTimeValueChangedEventArgs
e)
        {
            if (e.NewDateTime != null)
            {
                DateTime dateTo = (DateTime)e.NewDateTime;
                DateTimeOffset dateOffset = new DateTimeOffset(dateTo,
                                            TimeZoneInfo.Local.GetUtcOffset(dateTo));

                _searchToDate = dateTo.ToString("o") + "\n";
                _searchToDate = dateOffset.ToString("o") + "\n";
                _searchToDate = trimDateFromDateTime(_searchToDate);
            }
        }

        private void datePicker_From_ValueChanged(object sender,
DateTimeValueChangedEventArgs e)
        {
            if (e.NewDateTime != null)
            {
                DateTime dateFrom = (DateTime)e.NewDateTime;
                DateTimeOffset dateOffset = new DateTimeOffset(dateFrom,
                                            TimeZoneInfo.Local.GetUtcOffset(dateFrom));

                _searchFromDate = dateFrom.ToString("o") + "\n";
                _searchFromDate = dateOffset.ToString("o") + "\n";
                _searchFromDate = trimDateFromDateTime(_searchFromDate);
            }
        }

 private String trimDateFromDateTime(String aFullDate)
        {
            String resultantString = null;
            try
            {
                int posSpace = aFullDate.IndexOf('T');
                resultantString = aFullDate.Substring(0, posSpace);
            }
            catch (Exception e)
            {
            }
            return resultantString;
        }
```

On tapping the search button on the first panorama item, we are redirecting to the ArticlesListingPage by supplying the required values. The *Content Search* server hit will be made on that page.

Content Search

We can request for the content search by passing some parameters like keyword to be search, page number, section name, tag name, time duration etc.

(C) Copyright Nokia 2014. All rights reserved.

```
        public void getArticlesContents(String aKeyword, int aPageNumber)
         {
            HttpWebRequest httpReq = (HttpWebRequest)HttpWebRequest.Create(new
Uri(AppConstants.searchBaseUri + "q=" + aKeyword + "&page=" + aPageNumber +
                "&from-date=" + _searchFromDate + "&to-date=" + _searchToDate +
"&format=" + "json" + "&show-fields=thumbnail" + "&section=" + _sectionNameToBeSearchIn
+
                "&tag=" + _tagNameToBeSearchIn));
             httpReq.BeginGetResponse(HTTPWebRequestCallBack, httpReq);
         }
```

In the callback, the response is saved and parsed.

Note: Data can be fetched from The Guardian API by either JSON format or XML format. We have used JSON format in this article, you may use XML format as well. To do so, simply send the value of *format parameter* as **xml** rather than **json**. In that case, XML parsing will have to be implemented as well.

In the `OnNavigatedTo()`, we are saving the received values from the previous page and making the server request to search the articles based on our criteria. After receiving the successful response the response needs to be parsed.

In this search list, 10 items will be visible by default. To view rest of the items, *more option* is available here as well. By tapping *more option*, new server request will be made which will fetch next 10 items. On tapping other than the item having *more option*, the detail page will be opened showing the article's detail in a web browser.

Build and Run

Now you may build the app and try to run it.

Summary

This way we can get latest news in the form of articles from The Guardian's site using its API.

References

- http://www.guardian.co.uk/open-platform

(C) Copyright Nokia 2014. All rights reserved.