mp3scraper User's Guide

Free and open-source program (GPL v3) for finding MP3 links and generating RSS files.
This program does not download MP3 files and is not a podcatcher.

CONTENTS

THE PROBLEM

Some web sites offer MP3 downloads, and may call themselves podcasts, but don't provide an RSS or iTunes feed. The MP3s can be manually downloaded, but most podcast apps and podcatchers can't parse the web pages to automate access to new MP3s.

THE SOLUTION

mp3scraper loads a web page and parses, or "scrapes" the markup, i.e., the HTML code of the page. **mp3scraper** then creates an RSS file to be fed to your podcatcher or app.

QUICK START

Extract the files from the ZIP file into a folder. In the same folder as "`mp3scraper.exe`", you'll also find "`mp3scraper.exe.config`". Open this file in a plain text editor, such as Notepad. Microsoft Word and other word processing programs will work, but be sure to save changes as plain text.

mp3scraper has no user interface, i.e., no buttons or text boxes. Web page URLs and other settings are provided by editing the config file. The initial settings will work, but probably do not refer to your personal favorites. Edit the config file, adding and removing settings for the sites you wish to scrape.

Once the config file is ready, you can run the program by double-clicking the `mp3scraper.exe` icon from a file explorer or the desktop. You can run the program from a command prompt. Progress messages and error messages are displayed on the console window. If some sort of error occurs, you can direct all message to a text file by running from a command prompt:

```
C:\folder> mp3scraper.exe > mp3scraperoutput.txt
```

If files do not appear in the expected location, rerun the program watching for error messages. Make sure you spelled everything correctly, such as web site names.

TEST FILES

The program is delivered with test files. You can find these also at http://www.gocek.org/software/mp3scraper.htm .

mp3scraper.exe.config in the ZIP files is provided with test settings to scrape the test pages on gocek.org. Note the destination folder is c:\podcasts which must exist before you run the program, and that can be changed in the config file. RSS files will be generated but no FTP destination is specified, so you can simply look at the RSS results. If RSS files are not generated, then review the console output for errors. The test files and MP3s are boring, simply a few types of web pages available for your initial testing. Going forward, you need to adjust the config file to point to the web pages that have the MP3s you really care about, set up FTP destinations, etc.

DETAILS

A "podcast", technically, is more than a collection of media files. In order for end users to subscribe to the podcast, the publisher needs to provide a "feed" file that can be used by a podcatcher program to locate and download the media files. A "podcatcher" is a program that manages a user's podcast subscriptions. Most smartphone operating systems support podcast applications that act as podcatchers. You plug in the subscription address and the device periodically downloads the latest content.

Let's say you want to listen to an hourly news summary. National Public Radio provides a summary. Start at the NPR home page,
http://www.npr.org/podcasts/500005/hourly-news-summary

But that's not the subscription page. For that, use the menus or links on that page to find the subscription addresses for iTunes and RSS link. The RSS link is:
http://www.npr.org/rss/podcast.php?id=500005

If you open that in your browser, you'll see the hourly news item. In this case, the podcast only has one item at a time. Most podcasts offer old episodes, and you'd see a bunch of items, usually listed in reverse chronological order (the newest episode is listed at the top).

Some podcasts, however, are publicized via a web page with links to media files such as MP3 files, but provide no subscription feed file. You can manually download the media files and listen, but that's annoying. You just want your podcatcher to kick off every once in a while and do the work. Run **mp3scraper** and your podcacther will use the results.

mp3scraper specifically looks only for MP3 links. If you need to scrape, say, video content, **mp3scraper** can't help.

PREREQUISITES

The target .NET framework for m3scraper is 4.5.2, so you'll need a fairly recent version of Windows with the latest .NET framework installed. If you're running Windows 7, 8 or 10, you should be all set.

mp3scraper generates RSS files which need to be placed in a location your podcatcher or app can access. If your podcatcher can read feed files directly from your hard drive, that may be the easiest way to go. The RSS files can also be uploaded to a web site.

You need the URL (web address) for a web page with MP3 links. Most podcasts provide feed files and **mp3scraper** is not needed. **mp3scraper** is useful when a web page has download links for MP3 files but provides no feed file. The page doesn't need to be an episodic podcast; it can be just a page with a bunch of song links.

MORE DETAILS ON WHAT MP3SCRAPER TRIES TO DO

RSS Generation and Regeneration

The first time a host page, named by the "`url`" config file setting, is scraped, there is no pre-existing RSS feed that had been generated by **mp3scraper**. The program scrapes all the MP3s it can find and generates the RSS. On subsequent runs, **mp3scraper** first read in the existing feed. The item with the most recent date is used to determine whether the configured refresh period (via the "`refreshDays`" setting) has elapsed. For example, if `refreshDays` is "7" but an existing feed has an item that is only four days old, then the web page is skipped.

Note that once `refreshDays` has elapsed, **mp3scraper** will refresh that feed every time it is run until the host web page finally offers a new MP3 with a recent modification date. For example, if `refreshDays` is "7" and this period elapses with no new MP3s, **mp3scraper** does NOT wait another seven days; **mp3scraper** refreshes the feed every time it is run until new MP3s are found. In order for **mp3scraper** to find out if there are newer MP3s, it has to scrape the links and query the remote files' properties. **mp3scraper** tries to avoid that network activity via `refreshDays`, so you want `refreshDays` to be as long as possible to keep runtime to a minimum. If the publisher is offering dozens of MP3s, it really does take some time to scrape and query. If you KNOW that the MP3s are only updated once a week, it is helpful to you and to the publisher to avoid the work when possible. If a web page becomes inactive, with no new offerings (ever), you should set the feed to INACTIVE in the config file (or remove it altogether). Otherwise, the old files will be continually checked.

Ordering of MP3 Links

An RSS file (for the purposes of **mp3scraper**) is a plain-text XML file with a few properties to describe the overall feed and then a series of items to describe each MP3 link. It is helpful to podcatchers if the items are ordered in reverse chronological order, i.e., with the most recent item listed first. **mp3scraper** does not actually sort the items based on the MP3 file modification date. **mp3scraper** assumes that the links are ordered either chronologically or reverse-chronologically, depending on the value of the "`sortDescending`" configuration setting. If `sortDescending` is true, **mp3scraper** assumes the first MP3 link found in the web page is the most recent (and vice-versa when `sortDescending` is false). You (the user of **mp3scraper**) must look at the web page and figure out how to set `sortDescending`. If the web page is in some sort of random order, then you'll just have to take your best guess and hope that your podcatcher can figure it out. ORPHANED MP3 ITEMS: After a feed has been generated from a scraped page, an MP3 link might disappear from the page. Usually, this means that the MP3 file itself has been deleted and it no longer makes sense to maintain that link in the feed. However, it is possible the file still exists, but without a link on the web page, and the RSS item could be maintained. The user can control this with the `retainOrphans` config file setting.

Saving the RSS File

The generated RSS feed file will be saved on in the folder named by the "`destFolderName`" config file setting, with a file name corresponding to the "`destBase`" setting. Typically, the file extension should be ".xml" but **mp3scraper** does not require that. This file can be uploaded to a web address with a name corresponding to the "`existingFeedFolder`" setting. If the file is not uploaded or if `existingFeedFolder` is blank, **mp3scraper** will refresh the feed every time it is run because the previous MP3 mod date will not be determined and `refreshDays` will be ignored. **mp3scraper** does not read the existing file from a hard drive (only from a web location).

Uploading the RSS file

mp3scraper will upload the feed file to the remote location via FTP if credentials are provided. Note this is not a secure way to implement FTP operations. It doesn't use SFTP and the credentials are stored and passed to the network in the clear. If you're just running this at home on a secure connection, then there's no problem. I recommend against running this on an open, public wi-fi connection. If you really care about the security of an FTP connection, the workaround is to blank out the FTP settings in the config file, allow **mp3scraper** to write the files to your hard drive, and finally use a secure method (not **mp3scraper**) to upload the files to the remote location.

Non-obvious Scrapes

Some web sites provide media content that is easy to consume while using a web browser opened to the provider's web page, but the underlying MP3 links are hard to scrape. You can use your browser, mouse and keyboard to download individual MP3 files one at a time, but the web page is coded to allow that without exposing the full MP3 links. **mp3scraper** can work around some of these obfuscation methods, but not all. The reason it's hard to scrape these pages is because the providers want you to use the web page and view advertisements while consuming the media content. They don't want you to periodically scrape the page without seeing the ads and without manually visiting the site.

Although **mp3scraper** defeats certain types of obfuscation, I am not a big fan of ad blockers because they reduce income for providers of "free" content and most legitimate sites provide reasonable ad content. But, I am also not a fan of audio content that is restricted to a web browser

and hard to consume away from my PC (such as in my car). I am particularly annoyed by web sites that claim to be podcasts but provide no feed file and make the page hard to scrape. Sometimes that's because the publisher doesn't know any better, but sometimes it's because the publisher is trying to control my usage. Provide the content or don't, but don't force me to sit at my PC to listen to a song because you think you'll make a few cents by driving me to an Amazon purchase.

mp3scraper scrapes only publicly available MP3 links and does not try to scrape secure (passworded) content.

Relative Addresses

Sometimes the links can be scraped but they're relative, i.e., the fully qualified URL is not present, but part of the URL is present. You might see: /folder1/folder2/abc.mp3 rather than http://www.radiostation.com/ folder1/folder2/abc.mp3 In this case, set `prependRelative` to either http://www.radiostation.com or http://www.radiostation.com/ depending on whether the scraped links start with a slash.

Embedded Players with no Links

Sometimes, there are no MP3 links, and embedded media players (such as Flash players) point to a different folder. **mp3scraper** can't deal with this directly. In some cases, the embedded players (one for each episode) point to different files in the same folder. In this case, you might be able to find the list of MP3s on that other folder and scrape that page instead. This is often the case when the provider uses archive.org for MP3 storage. You need to view the page source and read the HTML to figure this out.

Obfuscated Links

Sometimes the links are obfuscated with HTML escape codes. **mp3scraper** tries to un-escape these types of addresses. If that fails but there are .MP3 links, use `prependRelative` along with stripBaseName, as long as the folder doesn't change for each MP3 file.

Reused Links

Sometimes, the provider changes the content periodically but reuses the same file name. **mp3scraper** will work and the updated file date will be determined, but the resulting RSS file will contain only the one link. In that case, run **mp3scraper** regularly to recreate the RSS file when the provider updates the content and run your podcatcher to download the content. The podcatcher I use allows me to save the MP3 file with a date in the file name, so each download is saved as a different file when I want to save more than one.

Defunct Podcasts

All good thinks come to an end, and that's true for podcasts. In many cases, old episodes are of no value, such as commentary on current events. However, a podcast about "oldies" can be listened to from start to finish again, since the songs were old to begin with. A podcatcher usually can't help, since it will always try to download the latest episodes. If you don't want to download every episode to your phone or USB drive, it takes some manual labor to manage and find old episodes.

mp3scraper provides the `wraparound` attribute to help. A new episode is selected every few days and "wraps around" to the beginning after reaching the end. As long as the source files remain on the remote web site, **mp3scraper** will find them as if the podcast were still active. Note that when a podcast offers an RSS feed file, **mp3scraper** is normally not needed at all, since

mp3scraper just generates an RSS file. However, **mp3scraper** can scrape the remote RSS file in order to step through a defunct podcast.

The `wraparound` attribute is also useful for a directory of MP3 files, allowing each file to be selected, over time.

See Appendix I for details on how often a new link is selected.

APPENDIX I – SETTINGS REFERENCE

mp3scraper has no user interface. Program behavior is controlled by a configuration file. The format of this file may seem unusual, but the format is a standard format for Windows programs.

When the program is extracted from the ZIP file, keep `mp3scraper.exe` and `mp3scraper.exe.config` together, in the same folder. The names of these two files can be changed, but they must be changed together, e.g., if you change `mp3scraper.exe` to `foobar.exe`, change `mp3scraper.exe.config` to `foobar.exe.config`.

There are currently 21 settings. Some are optional, but it is the recommendation of the developer that a value for all settings be specified for every feed, even if blank. This forces the program to behave as intended rather than allowing the program to assume default settings, which could change in future versions.

When editing the file, a user should change only the “add key” lines that look like this:
```
<add key="ftpPath0001" value="ftp://ftp.mydomain.net/podcasts/" />
```

You will change the numeric part at the end of the key (referred to as the *index*), and the value part. Do not change the base key name. In the example above, the base name is “`ftpPath`” and the index is “0001”. You can copy and paste this to something like
```
<add key="ftpPath0031" value="ftp://ftp.anothersite.com/stuff/" />
```
But, don’t change “`ftpPath`”. This part of the string is used by the program to determine runtime behavior. In this example, that’s the path to the remote site for uploading the generated RSS file. If you change the key to “blah1234”, the program won’t know what “blah” is and it will be ignored.

Other declaratory lines in the config file such as,
```
<?xml version="1.0" encoding="utf-8" ?>
```
or
```
</appSettings>
```
should not be changed or moved, since these are critical to the format of a config file. You can move the key settings into any order between the `<appSettings>` and `</appSettings>` lines, and there can be any number of keys, but don’t mess with those declaration elements.

When processing a feed, the key names are grouped by the index after the base name. For example. `ftpPassword0003` is the password used with `ftpUserName0003`, after scraping the MP3 links from `url0003`. In the config file, you’ll probably group all the 0001 settings together, and all the 0002 settings, etc., but the order doesn’t matter. You can skip indexes, e.g., you can have 0001 settings and 0003 settings, but no 0002 settings. You can specify up to 9999 groups of settings, 0001 through 9999. Don’t use 0000 (it will be ignored).

Always use 4 digits for the index, never fewer or more.
Key names are case sensitive. For example, you must specify "`testIndex`", not "`testindex`" or "`TESTINDEX`".

There is one setting that does not have an index, "`testIndex`". See the description below.

The Settings

`testIndex`	`testIndex` is the only key name that does not have an index after the name. The possible values are "" or "0001" through "9999". This will usually be "". If `testIndex` is not "", only the feeds for that index will be processed, e.g., set the `testIndex` value to "0002". Set the value to "" to process all feeds. If `testIndex` is set to an index, that feed will be processed even if `refreshDays` is used and the period has not yet passed. `testIndex` overrides `refreshDays` and overrides enabled.

`url0001`
`url0002`
…
`url9999`
	The value is the fully qualified web address of the web page expected to contain links to MP3 files, e.g., `https://archive.org/26/items/otr_cbsradiomysterytheater/`
	If the value is empty or if the page cannot be found, the index will be skipped.
	
	mp3scraper loads this web page and searches the markup for MP3 files, i.e., files with names that end with ".mp3". Not all pages that play audio have links to MP3 files. You may need to view the page source with your web browser to track down a page with MP3 files. Sometimes, the MP3 references are hidden through software tricks and **mp3scraper** will not be able to find them.
	
	mp3scraper can work around some software tricks such as relative addresses. See the settings below for details.
	
	By default, all the MP3 links on the page will be scraped (duplicates will be ignored), but this can be adjusted via the filtering settings.

`channelNotes0001`
`channelNotes0002`
…
`channelNotes9999`
	The value is mainly a note to yourself to remind you of why you scraped a web page instead of using the web site's features for downloading the MP3s. For example. "No RSS feed offered by abc.com."
	
	There are other reasons for scraping a page. I have encountered RSS feed files that cause errors when my podcatcher tries to read them. **mp3scraper** isn't exactly more robust; it doesn't try to do as much as a podcatcher.

`channelTitle0001`
`channelTitle0002`
…
`channelTitle9999`
	The value is used as the channel title in the generated RSS file. I'm not sure if there is a technical limit on the length, but keep it fairly short, such as "My Garage Band Podcast".

`destBase0001`
`destBase0002`
…
`destBase9999`
	The value is the base file name of the generated RSS file. In the URL http://www.abc.com/mypod.xml, the base file name is "mypod.xml". This setting is required, but actually, I haven't tested with this value set to "".

`destFolderName0001`
`destFolderName0002`
…
`destFolderName9999`
	The value is the name of the local destination folder for the generated RSS file. This is a folder name such as "`C:\mypodfolder\`", and does not include the file name. Put a backslash at the end. This setting is required, but actually, I haven't tested with this value set to "".

`enabled0001`
`enabled0002`
…
`enabled9999`
	The values for enabled are "true" or "false".
	
	Optional, defaults to false.
	
	If enabled is false, the feed won't be processed. Sometimes, you set up a feed but decide to stop processing it for a while, such as if it becomes inactive, but you don't want to delete the settings. If you just want to process one feed, such as when you're testing the settings, you can set all the enabled values to false except the one, but it's easier in that case to use the `testIndex` setting.

`existingFeedFolder0001`
`existingFeedFolder0002`
…
`existingFeedFolder9999`
	The value is the web folder (excluding the file name) of the location where the generated RSS file is uploaded (if it is uploaded). This is optional. The file name is taken from the `destBase` setting.
	
	Normally, if you upload the file to a remote site, you would specify `existingFeedFolder` so that the previous version of the file can be loaded while generating a new version. In particular, this is necessary for the `refreshDays` setting to be able to determine if it's time to refresh. But, `refreshDays` is optional.

`ftpPath0001`
`ftpPath0002`
…
`ftpPath9999`
	The value is the FTP path to which the generated RSS file is uploaded.
	
	A simple (not SFTP) connection is used. If you really care about security, set the FTP values to "" and upload with a secure client.

`ftpUserName0001`
`ftpUserName0002`
…
`ftpUserName9999`
	The value is the user name for the FTP upload credentials.

`ftpPassword0001`
`ftpPassword0002`
…
`ftpPassword9999`
	The value is the password for the FTP upload credentials.

`guidPrefix0001`
`guidPrefix0002`
…
`guidPrefix9999`
	The value is inserted into the generated RSS file.
	
	Each RSS item is assigned a GUID unique to the file. **mp3scraper** prefixes the MP3 file name with the value of this setting to create the GUID. Some podcatchers may use this. **mp3scraper** doesn't use the GUID other than to store it in the file. The value may be "".

`mp3Filter0001`
`mp3Filter0002`
…
`mp3Filter9999`
	If this value is not "", the specified value must be part of the scraped file name in order for the link to be saved in the generated RSS file. If the value begins with "!", the file name must not contain the value.
	
	If `mp3Filter` is "`good`", then "`goodvibrations.mp3`" and "`googoodolls.mp3`" will be scraped, but not "`wipeout.mp3`".
	
	If `mp3Filter` is "`!good`", then "`goodvibrations.mp3`" and "`googoodolls.mp3`" will not be scraped, but "`wipeout.mp3`" will be.

`permissionRevoked0001`
`permissionRevoked0002`
…
`permissionRevoked9999`
	The value is simply a description string written into the RSS file as an attribute on the root channel node (keep it fairly short).
	
	If you're generating RSS files and publishing them on your own web site, the MP3 publisher may ask you to stop. The publisher can't actually force you to stop, but you might stop out of courtesy. Set `permissionRevoked` to something like "`2016-06-27 joe@abc.com`" so you can remember the details.

`prependRelative0001`
`prependRelative0002`
…
`prependRelative9999`
	The value is a partial web address to insert before the scraped MP3 links.
	
	In some cases, a web page offers partial links such as "`/folder/song.mp3`", rather than the full path. Often, the path starts with the same folder that the web page is in. So, if you set `prependRelative` to "`http://www.abc.com`", the full path is constructed to be "`http://www.abc.com/song.mp3`".

	If `stripBaseName` is set to true, `prependRelative` needs a value of some sort to result in a full path.

`refreshDays0001`
`refreshDays0002`
…
`refreshDays9999`
	The value is the number of days to wait after generating an RSS file to re-scrape and generate a new version of the RSS file.
	
	When a page is scraped, all the MP3 files available from the feed are tested across the network for file size and modification date. This can take a little while to process, especially if there are multiple MP3s. Most feeds are published only once a week or less often, so use `refreshDays` to avoid processing every feed, every day.
	
	At run time, **mp3scraper** gets the previous generation day from the previously generated RSS file, which is opened using `existingFeedFolder` and `destBase`. If the previous feed can't be found or opened, the page is reprocessed.
	
	More specifically, the date of the last generation is determined by the most recent modification date of all the items in the existing RSS file, which may not be the same as the date the RSS file was created.
	
	If `refreshDays` is "" or "0", the page will be scraped every time.
	
	If `testIndex` is set, that feed will be reprocessed (`refreshDays` ignored).
	
	Note that once the refresh date is reached, the feed will be processed every time **mp3scraper** is run until the publisher finally adds a newer item to the scraped URL.

`retainOrphans0001`
`retainOrphans0002`
…
`retainOrphans9999`
	The value determines whether **mp3scraper** retains links to MP3s from the previous version of a feed even if the links have been removed from the scraped web page.
	
	If the MP3 publisher removes a link, it's usually because the MP3 itself was removed. However, you may find the file still actually exists, but the publisher decided to orphan it by removing the link. Since you previously scraped the link, you can keep it. If you're only saving the most recent few episodes of a podcast, you should set this to "false".

`sortDescending0001`
`sortDescending0002`
…
`sortDescending9999`
	The value is “true” or “false”.
	
	Most podcasts display their episodes with the newest episode at the top of the page. The episodes are sorted in reverse chronological order. Set `sortDescending` to “true”.
	
	The goal is to get the RSS items into a mostly reverse chronological order so that the podcatcher doesn’t have to work too hard to find the most recent episode.
	
	Once in a while, you’ll see a page sorted in chronological order (oldest episode at the top). In this case, set the value of the `sortDescending` key to “false”.
	
	mp3scraper doesn’t sort the scraped MP3 links; it just writes the RSS items into the generated RSS file in the order scraped from the web page. However, **mp3scraper** will reverse that if `sortDescending` is set to “false”.
	
	If the web page order is not chronological, such as by artist name, just leave `sortDescending` as “true”.

`stripBaseName0001`
`stripBaseName0002`
…
`stripBaseName9999`
	The value is “true” or “false”.
	
	If false, the whole MP3 link is returned. If true, only the base file name is returned. Use this in conjunction with the `prependRelative` setting to determine the actual MP3 link.
	
	I have seen web pages that contain MP3 links, but with all sorts of junky characters in the address, usually for use by the host web site for some unknown purpose. The `stripBaseName` setting discards all the junk, but to get a full MP3 link, you need to tell **mp3scraper** (via the `prependRelative` setting) what the address is.
	
	In other cases, you might specify `prependRelative` but with `stripBaseName` set to “false”. Whatever works.
	
	The most common case is to have `stripBaseName` set to false and `prependRelative` set to “”, but if everything was “common”, you wouldn’t need to scrape web pages.

`urlPrefix0001`
`urlPrefix0002`
…
`urlPrefix9999`
	The value is a string that precedes the link to MP3 files within the source feed.
	
	Optional, defaults to “=”.
	
	This is normally “=”, an equals sign. Most of the feeds scraped are HTML or XML files that contain a list of MP3 links. Within HTML, that usually looks like `src=″http://www.abc.com/song.mp3″`

	Within XML, that may look like
	`url="http://www.abc.com/song.mp3"`
	
	You may find a web page that uses some other convention, such as a collection of JSON properties.
	`"audio": "http://www.abc.com/song.mp3"`
	
	Use the `urlPrefix` attribute to direct **mp3scraper** to find MP3 links following that specialized string. In the example just above, specify:
	`<add key="urlPrefix0001" value="&quot;audio&quot;: " />`
	
	Notice that's how to search for a prefix that includes double quotes.

`allowHttps0001`
`allowHttps0002`
…
`allowHttps9999`
	The value is "true" or "false".
	
	Optional, defaults to false.
	
	In many cases, a feed contains SSL (Secure Sockets Layer) links, i.e., links that begin with "https" rather than "http". However, the World Wide Web Consortium (W3C) considers SSL enclosures within an RSS feed to be invalid. If **mp3scraper** writes an RSS feed file with https links as found at the source, the generated feed is technically invalid according to internet standards.
	
	Typically, https links to MP3 files can be changed to http without a problem. The MP3 file can still be downloaded by a podcatcher. Also, internet radios that process podcasts may fail when the MP3 links use HTTPS.
	
	Normally, this attribute should be set to false. If it is found that the http links won't download, then set the value to true, and the https protocol identifier will be retained.

`wraparound0001`
`wraparound0002`
…
`wraparound9999`
	The value is "true" or "false".
	
	Optional, defaults to false.
	
	When a source is not being updated, such as a defunct podcast or a web page that is not being updated, the default behavior of **mp3scraper** will be to generate an RSS feed that lists the same files in the same order every time **mp3scraper** is run.
	
	When the `wraparound` attribute is true, **mp3scraper** will generate an RSS feed with a single MP3 link selected from the source links. The selected link changes to the next source link after a number of days according to the total number of

	source links. After reaching the last link, **mp3scraper** will start from the beginning again. **mp3scraper** selects a new link every seven days when the total number of links is less than 200; 201-300 links: every six days 301-400 links: every five days 401-500 links: every four days 501-600 links: every three days 601-700 links: every two days more than 700 links: every day It is useful to set the `refreshDays` attribute to the same number of days as shown above, but this is not done automatically. If **mp3scraper** is run every day and would select a new link every five days because there are 352 source links (for example), then it's useful to refresh the generated feed every five days. If `refreshDays` is greater, or if **mp3scraper** is run less often that shown above, some links will be skipped over time.

`indirectFilter0001`
`indirectFilter0002`
…
`indirectFilter9999`
	Not currently supported.
	
	For future reference.

`indirect0001`
`indirect0002`
…
`indirect9999`
	Not currently supported.
	
	For future reference.

APPENDIX II – LEGAL ISSUES

I'm not your legal counsel, but here are my thoughts.

A legitimately licensed music feed is probably paying royalty fees for each MP3 download. **mp3scraper** does not download MP3s, but it does query the remote files sizes and modification dates. I don't know if such queries are considered a download. It is a courtesy to the publishers to limit traffic when possible, such as by using the `refreshDays` setting appropriately.

Some audio podcasts and MP3 collections do not provide an RSS subscription file ("feed"). **mp3scraper** scrapes web pages and generate RSS files. The RSS files contain the publishers' original MP3 links. The MP3 files are not downloaded. The original publishers are usually trying to control access to their content, which is their right, but the MP3s, if you can find them, are publicly accessible.

The generation of RSS files containing links to third-party sites and the automated scraping of third-party sites are not illegal nor copyright violations in the United States (fair-use, factual content, etc.).

Some media content publishers also publish web site usage policies that may state restrictions on the way the published content may be used, but these policies are usually not legally binding with respect to public content, i.e., content available without authentication (passwords, etc.).

The generated RSS files do not reuse creative content from the media publishers. Web links, site names and file names are factual information, not subject to copyright protection. The private use of the RSS files to help a podcatcher download the MP3s from their publicly accessible locations is not a violation. If you generate an RSS file and publish it on your own web site, even that would not be a copyright violation. The original publisher might ask you to stop publishing the RSS file, and you might stop out of courtesy, but the original publisher actually has no legal standing to make the request.

By generating RSS files with links to web sites, Gary Gocek (the original developer of **mp3scraper**) does not guarantee, approve, or endorse the information or products available on those sites (not that my disclaimer is any more legally binding than the web site usage policies).

FREQUENTLY ASKED QUESTIONS (FAQ)

Q: I found a web page with links to MP3 files, and it says it's a podcast, but there is no way to "subscribe". There is no feed file to be used by my podcatcher. Can't I just give that web address to my podcatcher and skip all this mp3scraper gobbledygook?

A: Maybe. Some podcatchers can find some MP3 links on raw web pages. Miro seemed to be able to do this, but it didn't provide dates for the episodes or MP3s. I no longer use Miro. I use rcFeedMe, which does not find MP3 links on raw web pages.

Q: Why isn't there a friendlier user interface for managing the settings?

A: Ultimately, the best way to use mp3scraper is to set up the settings and then schedule it to be run from your Windows task scheduler. At most, you just want to double-click the EXE and see that the RSS files have magically appeared a few minutes later so that you can run your podcatcher. I suppose a configuration front-end would be a nice addition.

Q: Any other future plans?

A: Some web sites provide links to folders that each contain an MP3 link, so it would be cool if mp3scraper could recursively traverse the folders.

Q: How long does mp3scraper take to run?

A: mp3scraper queries each remote MP3 file (without downloading it) to get the file size and the date it was last modified. If there are hundreds of MP3 files, which is not unheard of for an old podcast, that can take a while (a second or two per file). And, if you're scraping a dozen sites, well, do the math.

Q: Does all that work happen every time I run mp3scraper?

A: Use the refreshDays setting to control how often an RSS file is regenerated.

Q: How often should I run mp3scraper?

A: Depending on the targeted sites, you'll probably compromise on once a day. Consider that you will run mp3scraper to upload the RSS files, then run your podcatcher to download the MP3s, then sync the MP3s to your listening device. You're probably not going to do all that multiple times per day. There are some sites that offer MP3s more than once per day, such as an hourly news summary site. If you really prefer to download the MP3s rather than to stream the audio directly from the host site, you're going to have to compromise on the frequency.

Q: Does mp3scraper consider every single MP3 link?

A: Use the mp3Filter setting to filter out (or in) files.

Q: Is there a phone app coming, or a Mac version?

A: mp3scraper does not lend itself well to phone apps. I will not develop a Mac version.

Q: Isn't it a copyright violation to scrape a web site?

A: No, as long as the scraped content is not reposted. Web links are just names and are not subject to copyright protection.

Q: Does Gary Gocek eat his own dog food?

A: Yes, you can see my RSS feeds generated by mp3scraper at http://www.gocek.org/podcasts/