FARSIGHT SECURITY

Working With The SIE Batch API: A Command Line Client In Ruby, Perl, Python and C

Version 1.0.1
March 26, 2020

Remember how it was with you?
Remember how you pulled me through?
I remember
I remember

"I Remember", deadmau5 and Kaskade
https://www.youtube.com/watch?v=vUzVCw8BEXA
[10,108,796 views as of 2020-02-23]

Overview/FAQ

"What's SIE?" The Farsight Security Information Exchange ("SIE") is a way to see live passive DNS data feeds and other cybersecurity data feeds "as they happen." Think of SIE as being like having your own network collecting cyber security data all over the Internet -- but with none of the hassles of running that infrastructure yourself.

"What Data Is Available Via SIE?" SIE data is organized into "channels" (similar to cable TV channels). A guide to available channels is at https://www.farsightsecurity.com/assets/media/download/fsi-sie-channel-guide.pdf

"How Do I Get Access To Data on a Particular SIE Channel?" Until recently, subscribers needed to monitor channels of interest in real time, either from a server colocated at SIE or through use of an encrypted network tunnel. This worked, but was like collecting data from a flowing river -- if you didn't grab something as it went by, it was likely gone forever.

The New Alternative -- SIE-Batch: SIE-Batch offers a new alternative to watching SIE channels continuously. SIE Batch buffers and remembers SIE data for you and other subscribers. This means that subscribers can use SIE-Batch to pull data for select SIE channels from a buffer of data going back at least half a day. This means that if you just want to grab some SIE data, you can do so now with no waiting, and no fretting over recent data you might otherwise have missed!

"How Do I Mechanically Retrieve SIE-Batch Data?" The easiest way for subscribers to download SIE data via SIE-Batch is via an easy-to-use point-and-click web page offered by Farsight. Contact Farsight Security Sales for more information or to arrange access at sales@farsightsecurity.com or 1-650-489-7919.

"Is SIE-Batch Free?" No. SIE and SIE-Batch are commercial product offerings. For information about purchasing access or arranging for a trial, contact our sales team at sales@farsightsecurity.com or call 1-650-489-7919. (Farsight also supports the academic research community and individual sworn law enforcement officers via a program of grants and discounts, see https://www.farsightsecurity.com/grant-access/ for more on that option.)

"What If I Want To Integrate SIE-Batch Into My Own Custom Security Framework?" You can! Developers can directly access the SIE-Batch API from their own custom applications. You'll receive a pointer to the API documentation with your trial or subscription.

"So What's THIS Document About Then?" This document provides a brief intro to SIE and answers some of the more common questions users have. **This document is meant to help bootstrap developers by showing examples of how to use SIE-Batch in Ruby, Python, Perl and C.** For each language, we build a simple command line client that let's you:

- Verify that your SIE-Batch API key is OK
- List the channels you've got available
- Find the range of dates available for a specific channel (and how many bytes that data represents)
- Request data for a specific channel for a user-specified number of minutes (going back from now)
- Request data for a specific channel from a user-specified starting datetime to a user-specified ending datetime

These examples should give you a "quick start" when it comes to using SIE-Batch and building your own custom apps.

What Else Can I Do With These "Proof of Concept" Applications? If you like to work at the command line/from the Un*x shell prompt, you may find these proof of concept applications are just what you need for:

- Quickly verifying that your SIE-Batch API key is okay and that the channels you expect have been provisioned
- Routinely downloading SIE data (perhaps from a cron job)
- Making *ad hoc* SIE data requests to support one-off data requests for investigations

"I Have Other Questions!" Feel fee to drop us a note at support@farsightsecurity.com -- we'd be glad to help.

Contents

Section I. INTRODUCTION

I-1. Overview

The Farsight Security SIE Batch service allows Security Information Exchange (SIE) subscribers to conveniently download recent SIE data for select subscribed SIE channels. This normally happens via an interactive web client, as described in:

> "What's SIE Batch? Why Might I Be Interested In It?"
> https://www.farsightsecurity.com/txt-record/2020/02/20/stsauver-siebatch/

However, the SIE Batch **API** that underlies the SIE Batch web client is also available for subscriber use. It is documented on a customer-only web site:

> https://batch.sie-remote.net/apidoc/

The purpose of this report is to help subscribers begin working with that API. We will be providing example implementations of a simple command line client in four popular languages: Ruby, Perl, Python and C.

I-2. Context: Why This Whitepaper? Why Build An Actual Sample Client?

We know that it can sometimes be tricky to begin working with a new API, no matter how experienced you may be, and no matter how clean the API itself may be. Having a working example can be a huge boon. Having an example in the *specific language you actually develop in* can put your productivity into overdrive.

In the past, we'd published the blog article "Making Programmatic DNSDB Queries With libcurl"
(see https://www.farsightsecurity.com/txt-record/2016/11/04/stsauver-dnsdb-libcurl/).
That article was meant to help people begin to call ***DNSDB***'s API from C with libcurl. It has been quite popular.

Paralleling that earlier DNSDB-focused article, shortly after SIE-Batch was announced we published another blog article showing how to make simple programmatic queries using ***SIE-Batch's*** API from C with libcurl:
https://www.farsightsecurity.com/txt-record/2020/03/23/stsauver-siebatchc/

The primary issue with that article is that we wanted to keep it short, so it really only barely scratched the surface of what the SIE-Batch API can do, and it didn't deliver a true fully-featured SIE-Batch API client. It was also written around libcurl and C. While C remains a popular choice among professional developers, C is a language that's particularly cumbersome to use for string-handling. We believe most users will prefer to use a string-manipulation-friendly scripting language such as Ruby, Perl or Python. In this whitepaper we're providing full examples for all three of those scripting languages, while also providing a full version implemented in C.

Finally, when it comes to the question of "Why actually build a full client?", there can be many subtle nuances to using an API that you might not notice unless/until you're actually developing a complete application. Hence, we've gone ahead and done so for this whitepaper, thereby effectively "eating our own dog food."

I-3. Client Design and Desired Features: Client Use Cases

Use cases tend to drive client features. We had three main use cases in mind for this set of programs.

Use Case #1: Testing One of the simplest use cases for a command line client is to confirm that we can reach the SIE-Batch API server, or confirm that our SIE-Batch API key is valid, or check to see the range of dates for which data is available (and how much data that represents). A command line client should make it easy to do those things, and for us to get a list of the channels the client supports.

We envisioned something like:

```
$ sie_get_rb checkkey   ← to check connectivity and verify key validity
$ sie_get_rb channels   ← to get a list of channels
$ sie_get_rb 212        ← get available dates and volume for ch212
```

Use Case #2: Ad Hoc/One-Off Analyses Moving beyond simple testing, we believe SIE-Batch is perfect for an analyst who wanted to get an *ad hoc* sample of data, perhaps covering a particular UTC datetime range for a particular SIE channel (such as SIE Channel 212 from February 13th, 2020 at 18:42 UTC to February 13th, 2020 at 18:47). We wanted the analyst to be able to simply say:

```
$ sie_get_rb 212 "2020-02-13 18:42:00" "2020-02-13 18:47:00"
```

The output from that command would then go to an output file such as:

```
sie-ch212-{2020-02-13@18:42:00}-{2020-02-13@18:47:00}.jsonl
```

That file's name may initially seem "exotic" or "intimidating," but it actually is pretty straight forward when decoded:

Filename element:	Connotes:
sie	source of this data
ch212	specific channel
{2020-02-13@18:42:00}	starting datetime
{2020-02-13@18:47:00}	ending datetime
jsonl	file format

Having this sort of systematic file naming structure is important if you end up with many files – you want the files to be easily sortable and "self-documenting" if at all possible, so there's no need to produce and retain separate "metadata" about the files.

Use Case #3: Routinely Scheduled Downloads As we worked on the client and talked with colleagues, we also came to understand that another popular use case would consist of customers **routinely pulling data for a channel from cron.** This might be done to:

- Collect data in support of an actual investigation
- As a way to routinely download and save data, or
- As part of an ongoing quality control testing program.

This use case drive us to support a simplified query specification, namely:

```
$ sie_get_rb channelNumber now minutesBack
```

For example:

```
$ sie_get_rb 212 now 5
```

That is, "retrieve the last five minutes of ch212."

While this request format is simplified, the emitted output filenames are written in the same format as the explicitly-specified datetime queries described previously in this section.

I-4. File Formats

The contents of the output file will be the native format for the channel (e.g., either JSON Lines or binary NMSG format).

Some channels mentioned in the Farsight Security Channel Guide[1] may **not be available** via SIE Batch API at this time. Currently the channels NOT available via SIE Batch are Ch14 (Darknet), Ch220 (DNS Errors), and Ch255 (Heartbeat).

The channels that are currently delivered in **binary NMSG format** are Ch204, Ch206, Ch207, and Ch208.

The remaining channels (Ch25, Ch27, Ch42, Ch211, Ch212, Ch213, and Ch214) are all delivered in **JSON Lines format**.[2]

a) JSON Lines Format Files: Let's look at an example to make this "concrete." We'll start with JSON Lines format, as used by Channel 212. We can "pretty print" JSON Lines output using the popular JSON formatting client "jq" (see https://stedolan.github.io/jq/):

```
$ jq '.' < sie-ch212-{2020-02-13@18:42:00}-{2020-02-13@18:47:00}.jsonl
{
  "time": "2020-02-13 18:42:02.823029041",
  "vname": "SIE",
  "mname": "newdomain",
  "source": "a1ba02cf",
  "message": {
    "domain": "xmzpknm.cn.",
    "time_seen": "2020-02-13 18:40:51",
    "bailiwick": "cn.",
    "rrname": "xmzpknm.cn.",
    "rrclass": "IN",
    "rrtype": "NS",
    "rdata": [
      "ns3.dnsdun.com.",
      "ns3.dnsdun.net."
    ],
    "keys": [],
    "new_rr": []
  }
}
[etc]
```

Some JSON Lines files, such as the output from Ch27, may include base64-encoded fields. For example:

```
{"time":"2020-02-13 20:37:14.523360013","vname":"base","mname":"encode",
"source":"a1ba02cf","message":{"type":"JSON","payload":"[encoded payload blob
here]"}}
[etc]
```

To decode those payloads, you can say:

```
$ jq -r '.message.payload' <
sie-ch27-{2020-02-13@20:36:00}-{2020-02-13@20:38:00}.jsonl | base64 -d | jq '.'
{
  "reported_url": "[URL elided here]",
  "domain_malicious": "0",
  "falsepositive": "0",
  "url_type": "Phishing redirect",
```

[1] See the SIE Channel Guide that's available at https://www.farsightsecurity.com/assets/media/download/fsi-sie-channel-guide.pdf
[2] http://jsonlines.org/

```
      "shutdown_time": "[datetime elided here]",
      "ip_addresses": "[IP elided here]",
      "normalized_url": "[URL elided here]",
      "domain": "[domain elided here]",
      "detected_time": "[datetime elided here]",
      "target_brand": "[brand elided here]"
    }
        [etc]
```

b) NMSG Format Files: If you have an NMSG binary format file (rather than a JSON Lines file), use **nmsgtool** to process it. If you're using Debian Linux, you can install **nmsgtool** as a package, see https://www.farsightsecurity.com/technical/SIE-user-guide/sie-debian/

Source code is also available for those who prefer to build from source, or for use on systems other than Debian Linux. See https://github.com/farsightsec/nmsg and Appendix I for information on building **nmsgtool** for Mac OS Catalina.

Once we have the software we need installed, we can retrieve some data. Let's retrieve a minute's worth of ch204 with:

```
$ sie_get_rb 204 now 1
```

That yielded the binary format NMSG output file:

```
sie-ch204-{2020-02-13@20:54:00}-{2020-02-13@20:55:00}.nmsg
```

To see data decoded from NMSG format into normal **SIE presentation format:**

```
$ nmsgtool -r sie-ch204-{2020-02-13@20:54:00}-{2020-02-13@20:55:00}.nmsg
[113] [2020-02-13 20:53:59.849148756] [2:1 SIE dnsdedupe] [00000000] [] []
type: EXPIRATION
count: 2
time_first: 2020-02-13 14:51:17
time_last: 2020-02-13 17:21:18
bailiwick: straubing.de.
rrname: bnschkennzeichen-9.straubing.de.
rrclass: IN (1)
rrtype: TXT (16)
rrttl: 3600
rdata: "v=spf1 a:mxout58.expurgate.net ~all"
[etc]
```

OR if you want to decode the file from binary NMSG format into **JSON Lines format:**

```
$ nmsgtool -r sie-ch204-{2020-02-13@20:54:00}-{2020-02-13@20:55:00}.nmsg -J -
{"time":"2020-02-13 20:53:59.849148756","vname":"SIE","mname":"dnsdedupe",
"message":{"type":"EXPIRATION","count":2,"time_first":"2020-02-13
14:51:17","time_last":"2020-02-13 17:21:18","bailiwick":"straubing.de.","rrname":
"bnschkennzeichen-9.straubing.de.","rrclass":"IN","rrtype":"TXT","rrttl":3600,
"rdata":["\"v=spf1 a:mxout58.expurgate.net ~all\""]}}
[etc]
```

I-5. Understanding SIE-Batch API Data Availability and Channel Volumes

SIE-Batch is NOT meant to provide all-encompassing historical archives of SIE data. It just provides a **temporary buffer.** Typically it will have data going back for perhaps half a day, and then only for channels of manageable size. (The volume of data Farsight offers via SIE-Batch may also change over time.) At the time this was written, the length of time the data for each channel was retained, and the volume thereof, looked like the following (the scripts used to produce this little summary are included in Appendix II):

```
$ bash check-volumes.py | ruby read-time-2.rb | sort -k3n
Chan  Minutes         Octets
  27   1165          1768388  ← E.g., ~1.7 meg
  25   1166         18092719
 212   1166         68410653
 211   1166        526179776  ← Over half a gig
 213   1166      15039295360  ← 15 gig
  42   1165      61425710804
 206    735     120317004442
 204    735     143056192588
 207    735     153865819896
 208    735     191942653519  ← Nearly 200 gig
```

1165 minutes equals a buffer or more than 19 hours of data. 735 minutes equals a buffer of more than 12 hours worth of data. As time passes, new data will replace the old data in the buffer, FIFO (first in, first out) style.

While there's currently 12-19 hours of data there, most users won't be downloading anywhere near that much in a single request. SIE-Batch is really meant to handle much smaller requests. For example, maybe you want to download an hour's worth of Ch212 data. That quickly completes over a shared 100Mbps cable modem connection:

```
$ time ./sie_get_rb 212 now 60

real	0m5.044s
user	0m0.212s
sys	0m0.113s
```

Even downloading five minutes worth of Ch208, the busiest channel of those listed above, doesn't take long:

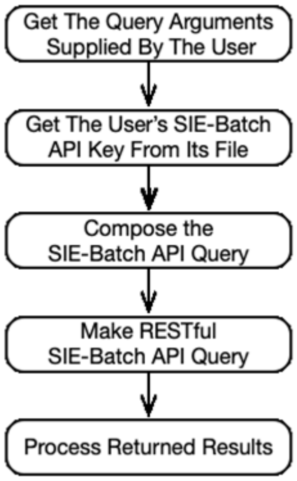
```
$ time ./sie_get_rb 208 now 5

real	3m13.779s
user	0m11.211s
sys	0m12.415s
```

Section II. Our Approach To The Sample Clients

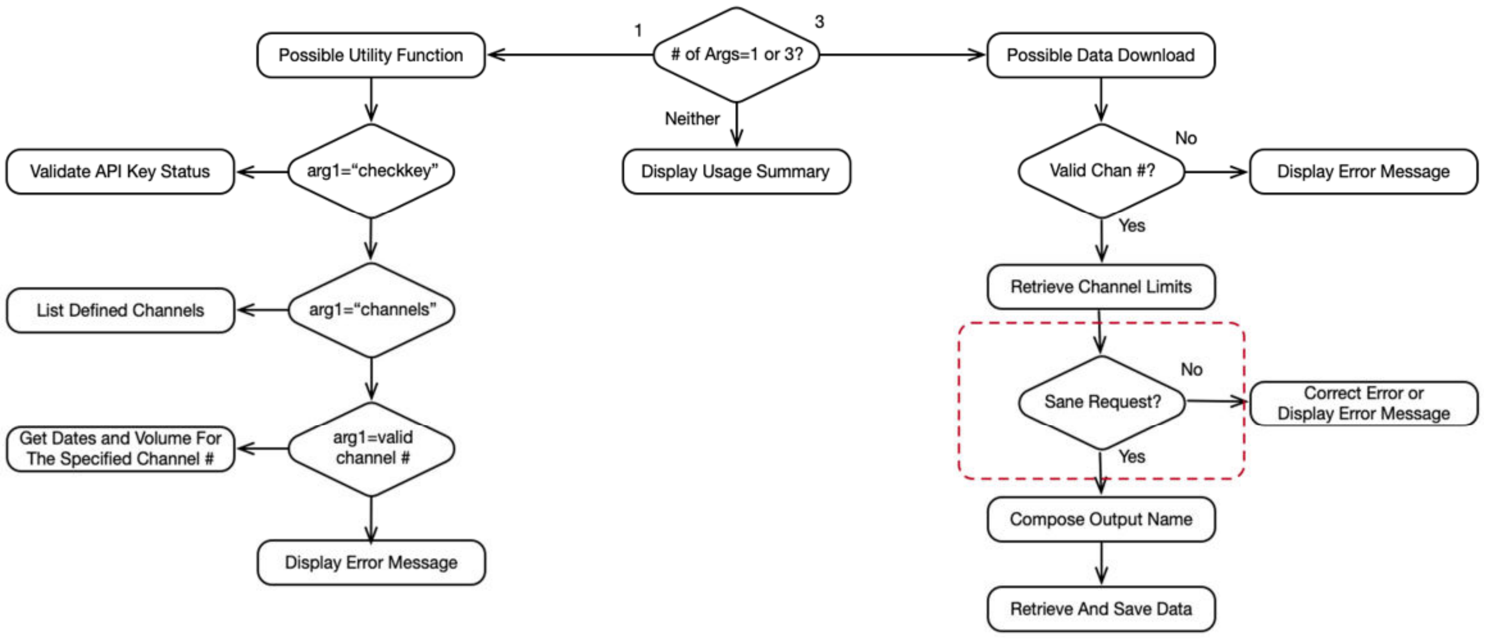
II-1. The Overall Process

We can describe our overall process flow as a five-step process:


Get The Query Arguments Supplied By The User
Get The User's SIE-Batch API Key From Its File
Compose the SIE-Batch API Query
Make RESTful SIE-Batch API Query
Process Returned Results

We'll now explain what we mean by each of those "blocks:"

i) Query arguments will typically be picked up via an "`argc/argv[]`"-like mechanism, to use the "C nomenclature" for accessing command line arguments.[3]
ii) Retrieving the SIE-Batch API key from its file is normally a matter of getting the user's home directory location, opening that file, reading in the SIE-Batch API key, and closing the file.
iii) Before we can send a query to the SIE-Batch API, we need to construct the required query. Those queries will be in JSON format. We can either assemble those queries "manually," or we can use a JSON library to serialize inputs into the required format.
iv) We're then ready to submit our RESTful SIE-Batch API query using an SSL/TLS library (such as libcurl).
v) Finally, we'll take the results and either just save them ("as-is") to a file or process the returned results to extract specific bits of information that are of interest.



[3] https://www.gnu.org/software/libc/manual/html_node/Program-Arguments.html

II-2. Input Validation and Error Handling

When using the sample application, there are a few common potential errors:

(1) **The user may attempt to execute a non-existent command:** Currently the sample client knows about two named commands: `checkkey` and `channels`. Users can also mention a specific channel number to get summary info about that channel (e.g., the earliest-available data, latest-available data, associated volume). However, a user might make a mistake, perhaps asking for checkkey**s** (plural), or **list**channels (instead of just channels). If/when that happens, the user will be shown an error message (and typically a command synopsis to help them out).

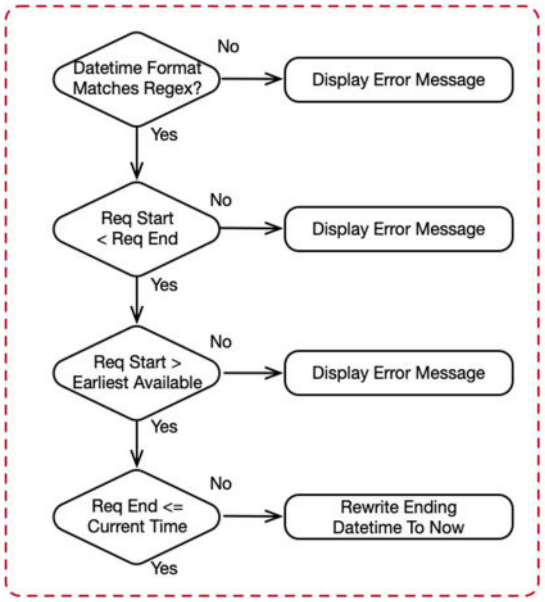
(2) **When the user actually wants to pull data from one of their channels (rather than just execute a utility command) opportunities for errors increase due to the presence of datetimes and other factors:**

- The user might ask for data that's older than what's available in SIE Batch's buffer, or for data that's yet to be collected (e.g., they might try to ask for data "from the future").

- If specifying arbitrary datetimes, we expect the user will supply a quoted and zero-padded YYYY-MM-DD HH:MM:SS datetime such as `"2020-02-24 14:22:35"` The user may fail to do so. Instead…
  - They may enter only a date or only a time (or they may enter both but fail to put quotes around the two values, causing them to be read separately rather than as a single datetime value).
  - They may omit leading zeros on various values. (perhaps entering "2020-2-5 8:20:00" instead of "2020-02-05 08:20:00")
  - The user may write the date in the wrong order or wrong format (we want YYYY-MM-DD but they may try something else, such as MM/DD/YY)
  - When specifying arbitrary starting and ending datetimes, the SIE-Batch API requires seconds be specified (but doesn't actually use them)
  - The user may attempt to specify a non-UTC time zone (such as EDT, PST, etc.)

- The user may ask for a channel that doesn't exist (or at least a channel which they haven't purchased)

Sanity Checking



II-3. Leveraging 3rd Party Libraries For Specialized Capabilities

In order to successfully build our sample clients, there were several capabilities we needed, most notably:

- An ability to make SSL/TLS-encrypted HTTP POST connections to the SIE-Batch API web site
- The ability to extract elements from JSON-encoded objects returned in response to queries

When the programming language or scripting language we used didn't offer those capabilities natively, we used third party libraries. The challenge in doing so is often one of deciding WHICH third party library to use, since often there are multiple competing options. In this case, we've selected:

<u>SSL/TLS Libraries:</u>

C:	**libcurl**	(see https://curl.haxx.se/libcurl/)
Perl:	**LWP**	(see https://metacpan.org/pod/LWP)
Python 3:	**pycurl**	(see http://pycurl.io/)
Ruby	**typhoeus**	(see https://github.com/typhoeus/typhoeus)

We picked libcurl because it is a well established library that we'd used in previous examples (see "Making Programmatic DNSDB Queries With libcurl," https://www.farsightsecurity.com/txt-record/2016/11/04/stsauver-dnsdb-libcurl/).

pycurl and typhoeus wrap libcurl for Python3 and Ruby, respectively, making them natural choices for those implementations.

We picked LWP::UserAgent just to "be a little different" when it came to picking a library for Perl.

There are other alternatives in each case that may also be worth evaluating, such as **LWP::Curl** or **Net::Curl::Easy** for Perl, or **curb** (https://github.com/taf2/curb) for Ruby. Feel free to experiment or use whatever works best for you.

<u>JSON Libraries:</u>

When it comes to working with JSON objects, we similarly had a number of options. We picked:

C:	**parson**	(see https://github.com/kgabis/parson)
Perl:	**JSON**	(see https://metacpan.org/pod/JSON)
Python 3:	**json**	(built-in package)
Ruby:	**json**	(built-in package)

Some alternatives you may want to consider include:

C:	**JSON-C** (see https://github.com/json-c/json-c) **Jsmn** (see https://github.com/zserge/jsmn) or **Jansson** (see https://github.com/akheron/jansson).
Perl:	A list of JSON implementations for Perl can be found at https://perlmaven.com/json
Python 3:	**orjson** (see https://github.com/ijl/orjson) **RapidJSON** (see https://github.com/python-rapidjson/python-rapidjson)
Ruby:	See the assessment of options at https://www.ruby-toolbox.com/categories/JSON_Parsers

II-4. Miscellaneous Program Design Considerations

Command Line Arguments At Program Invocation: Given the simple command line interface, we did not use getopt/getopt_long and named command line arguments, we just pick up positional arguments from ARGV. (If we had a more complex command line grammar, we might have needed to use getopt/getopt_long)

Endpoint and Proxies? We know that some of you need to connect to a non-standard endpoint or connect through a proxy. We've handled that explicitly in our code (note the endpoint and useproxy variables in the source for each implementation). We assume that if you're using a proxy that it is a SOCKS5 proxy at 127.0.0.1 on port 1080, as would typically result from saying:

```
$ ssh –D 127.0.0.1:1080 bastionhost.example.com
```

If need be, you can obviously tailor that (see also https://ec.haxx.se/libcurl/libcurl-proxies)

Section III. Installing The Clients

Note: *The clients all do the same thing. You don't need* ***all*** *of them, just install the client in the language you prefer.*

III-1. Installing the Ruby (sie_get_rb) Client

1) Copy `sie_get_rb` from Appendix IV, or download it from:

https://github.com/farsightsec/blog-code/tree/master/sie_get_clients/sie_get_ruby

2) Put your SIE-Batch API key into `~/.sie-get-key.txt` and make sure it isn't readable by others:

```
$ chmod  u=r,go-rwx  ~/.sie-get-key.txt
```

3) Install Ruby and any required Ruby gems (what you actually need to install will depend on whether you already routinely use Ruby). On the Mac, for example, assuming you use the Homebrew package manager (see https://brew.sh/) you'd say:

```
$ brew install ruby
$ sudo gem install typhoeus
```

4) We assume the IP address of the host you're using to access SIE Batch has been explicitly whitelisted by Farsight.

[**IF NOT**, but you at least have ssh access to a host that has been, you'll need to enable proxy support in the client, then create an ssh tunnel to that bastion host. To enable proxy support in the client, edit the Ruby source code. Find the **`useproxy`** line and change it to be **`true`**, then save the modified file. When you want to use the client via that proxy, you'll need to create an SSH tunnel to it. Assuming the bastion host is `bastionhost.example.com`, create a SOCKS5 ssh tunnel to it by saying (in another window): `$ ssh -D 127.0.0.1:1080 bastionhost.example.com`]

5) Now install the Ruby script to a directory in your path and make it executable:

```
$ sudo cp sie_get_rb.rb /usr/local/bin/sie_get_rb
$ sudo chmod a+rx /usr/local/bin/sie_get_rb
```

6) You're now ready to try the client:

```
$ sie_get_rb                    ← see usage summary
$ sie_get_rb checkkey           ← verify your API key
$ sie_get_rb channels           ← get list of channels
```

Note: to actually retrieve data for a channel, you'll need to be subscribed to that channel, and you'll have to ask for data from a date range that's available.

```
$ sie_get_rb 212                ← details for a channel (e.g., date range available)
$ sie_get_rb 212 now 5          ← grab 5 minutes of ch212
$ sie_get_rb 212 "2020-01-28 19:17:00" "2020-01-28 19:22:00"  ← grab data for a specific time range
```

Output will be to a file in your current directory. That filename will be something like the following:
`sie-ch212-{2020-01-18@21:22:19}-{2020-01-18@21:32:19}.jsonl`

7) Feedback/comments/questions? Please email `<stsauver@fsi.io>`

III-2. Installing the Python 3 (sie_get_py) Client

1) Copy sie_get_py from Appendix V, or download it from:

https://github.com/farsightsec/blog-code/tree/master/sie_get_clients/sie_get_python

2) Put your SIE-Batch API key into `~/.sie-get-key.txt` and make sure it isn't readable by others:

```
$ chmod  u=r,go-rwx  ~/.sie-get-key.txt
```

3) You likely already have Python 3 (in fact, perhaps even *multiple* copies of Python 3). The script will use the version that it finds first in your path. However, you can easily install a new copy should you need or desire to do so. For example, if you're on a Mac and use Homebrew (see https://brew.sh/) to manage your packages, you'd say:

```
$ brew install python3
```

4) Now install the python modules we'll need:

```
$ sudo pip3 install datetime
$ sudo pip3 install pathlib
$ sudo pip3 install pycurl
```

5) We assume the IP address of the host you're using to access SIE Batch has been explicitly whitelisted by Farsight.

[**IF NOT**, but you at least have ssh access to a host that has been, you'll need to enable proxy support in the client, then create an ssh tunnel to a bastion host that has access. To enable proxy support in the client, edit the Python source code. Find the **`useproxy`** line, change it to be **`True`** and save the modified file. When you want to use the client via that proxy, create an SSH tunnel to the proxy Assuming the bastion host is `bastionhost.example.com`, create a SOCKS5 ssh tunnel to it by saying (in another window): `$ ssh -D 127.0.0.1:1080 bastionhost.example.com`]

6) Install the Python script to a directory in your path and make it executable:

```
$ sudo cp sie_get_py.py /usr/local/bin/sie_get_py
$ sudo chmod a+rx /usr/local/bin/si_get_py
```

7) Try the Python client:

```
$ sie_get_py                  ← see usage summary
$ sie_get_py checkkey         ← verify key
$ sie_get_py channels         ← get list of channels
```

Note: to actually retrieve data for a channel, you'll need to be subscribed to that channel, and you'll have to ask for data from a date range that's available.

```
$ sie_get_py 212              ← details for a channel (e.g., date range available)
$ sie_get_py 212 now 5        ← grab 5 minutes of ch212
$ sie_get_py 212 "2020-01-28 19:17:00" "2020-01-28 19:22:00"  ← grab data for a specific time range
```

Output will be to a file in your current directory. That filename will be something like the following:
`sie-ch212-{2020-01-18@21:22:19}-{2020-01-18@21:32:19}.jsonl`

8) Feedback/comments/questions? Please send email to `<stsauver@fsi.io>`

III-3. Installing the Perl (sie_get_pl) Client

1) Copy sie_get_pl from Appendix VI, or download it from:

https://github.com/farsightsec/blog-code/tree/master/sie_get_clients/sie_get_perl

2) Put your SIE-Batch API key into `~/.sie-get-key.txt` and make sure it isn't readable by others:

```
$ chmod  u=r,go-rwx  ~/.sie-get-key.txt
```

3) You likely already have `perl` installed (in fact, perhaps even multiple copies of `perl`). The script assumes you want `/usr/local/bin/perl` , but feel free to change the first line of the script to taste. If you want a more-current version of `perl` than you may otherwise have, assuming you're on the Mac and using brew (see https://brew.sh/), try:

```
$ brew install perl
```

4) Now install/update the `perl` modules we'll be using from CPAN (these are broken onto 3 lines to avoid wrapping):

```
$ sudo cpan -i Carp Data::Structure::Util DateTime DateTime::Format::Strptime
$ sudo cpan -i File::HomeDir JSON LWP LWP::UserAgent Scalar::Util TimeDate
$ sudo cpan -i Time::ParseDate POSIX POSIX::strftime::GNU
```

5) We assume the IP address of the host you're using to access SIE Batch has been explicitly whitelisted by Farsight.

[**IF NOT**, but you at least have ssh access to a host that has been, you'll need to enable proxy support in the client, then create an ssh tunnel to a bastion host that has access. To enable proxy support in the client, edit the Perl source code. Find the `my $useproxy = '0';    # '0' is false` line and change it to be `'1'` instead of `'0'`. Save the modified file. When you want to use the client via that proxy, create an SSH tunnel to the proxy Assuming the bastion host is `bastionhost.example.com`, create a SOCKS5 ssh tunnel to it by saying (in another window):

```
$ ssh -D 127.0.0.1:1080 bastionhost.example.com    ]
```

6) Now install the Perl script to a directory in your path and make it executable:

```
$ sudo cp sie_get_pl.pl /usr/local/bin/sie_get_pl
$ sudo chmod a+rx /usr/local/bin/sie_get_pl
```

7) Try the client

```
$ sie_get_pl                ← see usage summary
$ sie_get_pl checkkey       ← verify key
$ sie_get_pl channels       ← get list of channels
```

Note: to get data for a channel, you need to be subscribed to it, and ask for data from a date range that's available.

```
$ sie_get_pl 212            ← get the details for a channel (e.g., date range available)
$ sie_get_pl 212 now 5      ← grab 5 minutes of ch212
$ sie_get_pl 212 "2020-01-28 19:17:00" "2020-01-28 19:22:00"   ← grab data for a specific time range
```

Output will be to a file in your current directory. That filename will be something like the following:
`sie-ch212-{2020-01-18@21:22:19}-{2020-01-18@21:32:19}.jsonl`

8) Feedback/comments/questions? Please email `<stsauver@fsi.io>`

III-4. Installing the C (sie_get_c) Client

1) Copy the C code for sie_get_c from Appendix VII, or download sie_get_c from:

https://github.com/farsightsec/blog-code/tree/master/sie_get_clients/sie_get_c

2) Put your SIE-Batch API key into `~/.sie-get-key.txt` and make sure it isn't readable by others:

```
$ chmod  u=r,go-rwx  ~/.sie-get-key.txt
```

3) We assume that you have a working C development environment. If not, you'll need to install a C compiler and supporting tools. For example, on a Mac, this normally implies installing Xcode: https://developer.apple.com/xcode/

cd into that directory that has the source code for the client, review the Makefile and tweak any defaults you don't like.

4) We assume the IP address of the host you're using to access SIE Batch has been explicitly whitelisted by Farsight.

[**IF NOT**, but you at least have ssh access to a host that has been, you'll need to enable proxy support in the client, then create an ssh tunnel to a bastion host that has access. To enable proxy support in the client, edit `sie_get_c.h`
Find `char useproxy[] = "no"; /* "yes" or "no" */` and change `'no'` to `'yes'`. Save the modified file. When you want to use the client via that proxy, create an SSH tunnel to the bastion host. Assuming the bastion host is `bastionhost.example.com`, create a SOCKS5 ssh tunnel to it by saying (in another window):

```
$ ssh -D 127.0.0.1:1080 bastionhost.example.com   ]
```

5) Build and install the executable with:

```
$ make
$ sudo make install
$ make clean
```

6) Try the client:

```
$ sie_get_c                ← see usage summary
$ sie_get_c checkkey       ← verify key
$ sie_get_c channels       ← get list of channels
```

Note: to get data for a channel, you need to be subscribed to it, and ask for data from a date range that's available.

```
$ sie_get_c 212            ← details for the channel (e.g., date range available)
$ sie_get_c 212 now 5      ← grab 5 minutes of ch212
$ sie_get_c 212 "2020-01-28 19:17:00" "2020-01-28 19:22:00"   ← grab data for a specific time range
```

Section IV. Conclusion

This project is a bit of an experiment. We hope that providing the sample programs in this document helps bootstrap those who'd like to develop their own code to access SIE-Batch API, and we hope it also proves useful as a simple command-line interface to SIE-Batch.

We've tried to be very candid about our thinking, and hope that you find yourself at least able to "live with" our choices even if you may disagree with some of them (we know, for example, that choice of formatting styles, and choice of third party supporting libraries can be quite contentious, right up there with questions like "what sort of BBQ is best?")

In any event, we welcome feedback "from the field" on these programs, and would love to hear about your experiences with them. We hope that they will provide a painless introduction to working with SIE-Batch API.

Acknowledgements

We'd like to thank our colleagues Mr. Tyler Wood and Mr. Chuq Von Rospach for their helpful comments in reviewing a draft of this document. Any remaining errors are solely the responsibility of the author.

Appendix I. Building nmsgtool from Source For Mac OS Catalina

Building nmsgtool on Mac OS X Catalina from Source… Version 1.0

Before attempting any part of the following installation, please back up your Mac!

1) Creating A Build Environment

You need to have **Xcode command line tools** installed so you'll have access to the C compiler and related software libraries you'll need. You can install Xcode from the Apple App Store, or you can just install the command line tools from the command line. We'll do the latter in this case.

```
$ xcode-select –install
```

Note: please do NOT type the dollar-sign prompt shown in front of any command in this Appendix.

If you see:

xcode-select: error: command line tools are already installed, use "Software Update" to install updates

that means that you've already got the command line tools installed.

You may still need to agree to the license terms if you've not done so already:

```
$ sudo xcodebuild –license
```
[enter your admin password]
[scroll through the license]
```
agree
```

2) Install The Homebrew Package Manager.

Homebrew (or normally just "brew") is a popular Mac package manager, and well help to help bootstrap the installation of some of the other required software packages.

```
$ /usr/bin/ruby -e
"$(curl -fsSL  https://raw.githubusercontent.com/Homebrew/install/master/install )"
```

[the above command is all one line, it just wrapped as shown here due to the limited width of the page]

Ensure that brew is up-to-date (it should already be if you just installed it, but if you've had it around for a while, it's good to double check/get up-to-date).

```
$ brew update
$ brew upgrade
```

3) Install wget

We'll now begin installing the other stuff we'll need. Some of the software we need can be installed via brew.

We need **wget** in order to have a command line web tool to retrieve some packages…

```
$ brew install wget
```

wget also requires an SSL/TLS certificate bundle for its trust anchors. We retrieve a copy of a reasonable cert bundle by saying:

```
$ wget -no-check-certificate https://curl.haxx.se/ca/cacert.pem
```

Now we'll make create the directory /usr/local/share/curl, copy the cert bundle to that directory, and make sure it's readable:

```
$ sudo mkdir /usr/local/share/curl
[enter root password]
$ sudo cp cacert.pem /usr/local/share/curl
$ sudo chmod a+r /usr/local/share/curl/cacert.pem
```

Curl now needs to be told where to find that cert bundle. We can tell the current shell to find them by saying:

```
$ export SSL_CERT_FILE="/usr/local/share/curl/cacert.pem"
```

Do the above command now. That command will be in effect for the duration your current window is open.

To also make your system remember that setting, e.g., to make it "permanent," add that line to your `.profile` file

```
$ cd
$ nano .profile        🡐 if you prefer a different editor, you can use it instead
[scroll to the bottom of that file]
export SSL_CERT_FILE="/usr/local/share/curl/cacert.pem"
[ctrl-O]               🡐 write the file
[ctrl-X]               🡐 exit from nano
```

4) Install Additional Command Line Utilities and Packages

Now let's use brew to install some other **command line utilities and packages** we'll need:

```
$ brew install autoconf
$ brew install automake
$ brew install binutils
$ brew install bison
$ brew install check
$ brew install cmake
$ brew install doxygen
$ brew install ghostscript
$ brew install graphviz
$ brew install libpcap
$ brew install libtool
$ brew install pkgconfig
$ brew install yajl
$ brew cask install mactex
```

5) Install Stuff We Need To Build From Sources (And Required Dependencies)

Some of the rest of the software that we also need, we'll need to build from source.

We'll do that now. **These packages all have dependencies.** This means that we need to build them **in the right order**, or the packages will complain that stuff they need is missing.

In general, at least for this section, we'll be:

-- cloning a Github repository
-- changing down into the directory where those files were copied
-- running autogen to make the configure script
-- running configure to make the make files
-- running make to compile the code
-- using sudo to install the compiled software
-- returning to the default home directory

Exceptions to that general approach will usually be highlighted in red (we'll also try to highlight things that are particularly easy to overlook or get wrong).

The first things we'll try installing that way are `protobuf`, `protobuf-c`, and `wdns`:

```
$ git clone https://github.com/google/protobuf.git
$ cd protobuf
$ sh autogen.sh
$ ./configure
$ make
$ sudo make install
$ cd

$ git clone https://github.com/protobuf-c/protobuf-c.git
$ cd protobuf-c
$ sh autogen.sh
$ ./configure
$ make
$ sudo make install
$ cd

$ git clone https://github.com/farsightsec/wdns.git
$ cd wdns
$ sh autogen.sh
$ ./configure
$ make
$ sudo make install
$ cd
```

We're now ready to install `nmsg` (it requires `protobuf`, `protobuf-c`, and `wdns`)

```
$ git clone https://github.com/farsightsec/nmsg.git
$ cd nmsg
$ sh autogen.sh
$ ./configure -without-libxs
$ make
$ make html
$ sudo make install
$ cd
```

Now let's do `sie-nmsg` (it requires `nmsg`, `protobuf-c`, and `wdns`)

```
$ git clone https://github.com/farsightsec/sie-nmsg.git
$ cd sie-nmsg
$ sh autogen.sh
$ ./configure
$ make
$ sudo make install
$ cd
```

Now install `OpenSSL`:

```
$ wget https://www.openssl.org/source/openssl-1.1.1e.tar.gz
```

OpenSSL is particularly security sensitive, so we encourage you to verify the checksums for the file as shown below, ensuring that the file you downloaded is correct and hasn't been tampered-with.

Always use the most recent version of **`OpenSSL 1.1`**, as shown at **`https://www.openssl.org/source/`**

```
$ shasum -a 256 openssl-1.1.1e.tar.gz
694f61ac11cb51c9bf73f54e771ff6022b0327a43bbdfa1b2f19de1662a6dcbe  openssl-1.1.1e.tar.gz
$ wget https://www.openssl.org/source/openssl-1.1.1e.tar.gz.sha256
$ cat openssl-1.1.1e.tar.gz.sha256
694f61ac11cb51c9bf73f54e771ff6022b0327a43bbdfa1b2f19de1662a6dcbe [confirm this value
matches what shasum reported…]

$ gunzip openssl-1.1.0e.tar.gz
$ tar xfv openssl-1.1.0e.tar
$ cd openssl-1.1.0e
$ ./config     [NOTE: not ./configure ]
$ make
$ make test
[look for a final "Result: PASS"]
$ sudo make install
$ cd
```

Finally, build **axa** (requires `nmsg, protobuf-c, sie-nmsg, wdns`, and `OpenSSL`):

```
$ git clone https://github.com/farsightsec/axa.git
$ cd axa
$ sh autogen.sh
$ ./configure
$ make
$ make doc
$ sudo make install
$ cd
```

That should do it! Congratulations!

Appendix II. Getting Channel Volume Summaries With sie_get_*

Checking Channel Volumes

sie_get_* can serve as a nice building block for broader analyses. Imagine a command file like:

```
$ cat check-volumes.py
sie_get_py 25
sie_get_py 27
sie_get_py 42
sie_get_py 204
sie_get_py 206
sie_get_py 207
sie_get_py 208
sie_get_py 211
sie_get_py 212
sie_get_py 213
```

We can run it with:

```
$ bash check-volumes.py > temp.txt
$ more temp.txt
  25  "2020-03-01 03:09:00"  "2020-03-01 22:36:00"        21,678,555
  27  "2020-03-01 03:09:00"  "2020-03-01 22:35:03"         1,476,344
  42  "2020-03-01 03:10:00"  "2020-03-01 22:36:03"    48,065,411,292
 204  "2020-03-01 10:21:03"  "2020-03-01 22:36:00"    98,432,360,388
 206  "2020-03-01 10:21:00"  "2020-03-01 22:36:00"    47,700,952,748
 207  "2020-03-01 10:21:03"  "2020-03-01 22:36:00"    99,674,838,790
 208  "2020-03-01 10:21:04"  "2020-03-01 22:36:00"   116,793,747,711
 211  "2020-03-01 03:09:59"  "2020-03-01 22:36:00"     1,284,373,715
 212  "2020-03-01 03:10:00"  "2020-03-01 22:36:03"       213,300,363
 213  "2020-03-01 03:10:00"  "2020-03-01 22:36:03"     8,235,193,961
```

Which channels have the most data available? Sort in descending order by volume:

```
$ sort -field-separator='"' -k5r < temp.txt
 208  "2020-03-01 10:21:04"  "2020-03-01 22:36:00"   116,793,747,711
 207  "2020-03-01 10:21:03"  "2020-03-01 22:36:00"    99,674,838,790
 204  "2020-03-01 10:21:03"  "2020-03-01 22:36:00"    98,432,360,388
  42  "2020-03-01 03:10:00"  "2020-03-01 22:36:03"    48,065,411,292
 206  "2020-03-01 10:21:00"  "2020-03-01 22:36:00"    47,700,952,748
 213  "2020-03-01 03:10:00"  "2020-03-01 22:36:03"     8,235,193,961
 211  "2020-03-01 03:09:59"  "2020-03-01 22:36:00"     1,284,373,715
 212  "2020-03-01 03:10:00"  "2020-03-01 22:36:03"       213,300,363
  25  "2020-03-01 03:09:00"  "2020-03-01 22:36:00"        21,678,555
  27  "2020-03-01 03:09:00"  "2020-03-01 22:35:03"         1,476,344
```

What channels have the highest data ***rate?*** We'll use a little ruby script to see…

```
$ cat read-time.rb
#!/usr/bin/env ruby
require 'date'
print "Chan  Earliest             Latest           ",
  "    Minutes        Octets        Rate\n"
while STDIN.gets
  values = $_.split('"')
  channel = values[0]
  x = DateTime.parse(values[1])
  y = DateTime.parse(values[3])
  duration_in_min = ((y - x) * (24 * 60)).to_f.round(0)
  volume_without_commas = values[4].tr(',' , '')
  rate=volume_without_commas.to_f / duration_in_min.to_f
  print channel, x.strftime('%Y-%m-%d %H:%M:%S'),"  ",
   y.strftime('%Y-%m-%d %H:%M:%S'),"  ",
   '%5.0f' % duration_in_min,"  ",
   '%14.0f' % volume_without_commas, "%12.0f" % rate, "\n"
end

$ ruby read-time.rb < temp.txt | sort -k7n
Chan  Earliest             Latest               Minutes        Octets         Rate
  27  2020-03-01 03:09:00  2020-03-01 22:35:03   1166         1476344         1266
  25  2020-03-01 03:09:00  2020-03-01 22:36:00   1167        21678555        18576
 212  2020-03-01 03:10:00  2020-03-01 22:36:03   1166       213300363       182933
 211  2020-03-01 03:09:59  2020-03-01 22:36:00   1166      1284373715      1101521
 213  2020-03-01 03:10:00  2020-03-01 22:36:03   1166      8235193961      7062774
 206  2020-03-01 10:21:00  2020-03-01 22:36:00    735     47700952748     64899255
  42  2020-03-01 03:10:00  2020-03-01 22:36:03   1166     48065411292     41222480
 204  2020-03-01 10:21:03  2020-03-01 22:36:00    735     98432360388    133921579
 207  2020-03-01 10:21:03  2020-03-01 22:36:00    735     99674838790    135612026
 208  2020-03-01 10:21:04  2020-03-01 22:36:00    735    116793747711    158903058
```

Simplifying that output further:

```
$ cat simplified_output.rb
#!/usr/bin/env ruby
require 'date'
print "Chan  Minutes        Octets\n"
while STDIN.gets
  values = $_.split('"')
  channel = values[0]
  x = DateTime.parse(values[1])
  y = DateTime.parse(values[3])
  duration_in_min = ((y - x) * (24 * 60)).to_f.round(0)
  volume_without_commas = values[4].tr(',' , '')
  print channel, '%5.0f' % duration_in_min,"  ",
   '%14.0f' % volume_without_commas, "\n"
end

$ ruby simplified_output.rb < temp.txt | sort -k3n
Chan  Minutes        Octets
  27   1166         1476344
  25   1167        21678555
 212   1166       213300363
 211   1166      1284373715
 213   1166      8235193961
 206    735     47700952748
  42   1166     48065411292
 204    735     98432360388
 207    735     99674838790
 208    735    116793747711
```

Appendix III. Miscellaneous Program Design Notes

App III-1. String Handling and Other Programmatic Considerations in C

String handling in C is a non-trivial task. Simon Tatham, author of the terminal emulator Putty, wrote:

> *"String handling in C is very, very primitive by the standards of almost any other language."* [* * *] *"You're probably thinking, by now, that C sounds like a horrible language to work in. It forces you to do by hand a lot of things you're used to having done for you automatically; it constantly threatens you with unrecoverably weird 31ehavior, hard-to-find bugs, and dangerous security holes if you put one foot across any of a large number of completely invisible lines that neither the compiler nor the runtime will help you to avoid; and, for goodness' sake, it can't even handle strings properly. How could anyone have designed a language that bad?*
> [explanation/discussion omitted here]"
> (see https://www.chiark.greenend.org.uk/~sgtatham/cdescent/)

We agree that C can be a challenging language to use when working extensively with strings. It is easy to write code that's either insecure or unstable, and some of the fixes for some of those issues may be incompletely-standardized.

In other cases, even basic higher-level string functions (such as extracting a substring, or searching for matches in an array of strings, or replacing characters in a string) may require supplemental routines. We want to be explicit about how we decided to address those challenges in our sample C code:

- **Secure alternatives to strcat and strcpy:** The stock **strcat** and **strcpy** commands are well known for being potentially dangerous. (See chapter 2 of "Secure Coding in C and C++, 2nd Edition," conveniently available as a sample chapter at https://resources.sei.cmu.edu/asset_files/BookChapter/2005_009_001_52710.pdf , 146 pages).

  To help deal with this issue, strcat and strcpy were augmented with **strncat** and **strncpy**. But since strncat and strncpy don't guarantee null termination for strings, they're still "less than ideal" (see for example "Stop using strncpy already!", https://randomascii.wordpress.com/2013/04/03/stop-using-strncpy-already/).

  strlcpy and **strlcat** are generally accepted as being a better option. One discussion: "strlcpy and strlcat—Consistent, Safe, String Copy and Concatenation" by Todd C. Miller (U. Colorado, Boulder) and Theo de Raadt (OpenBSD project) at https://pdfs.semanticscholar.org/f9cb/fd6575f89433f9ad80498a8328e4311ebba3.pdf

  Mac OS X, being rooted in BSD, has strlcpy and strlcat natively. Those of you running in some other environments may not. (If curious why not, see for example "The ups and downs of strlcpy()" at https://lwn.net/Articles/507319/) An easy way to get strlcat and strlcpy (if you don't already have them natively) is by downloading and including:

  -- http://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libc/string/strlcat.c
  -- http://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libc/string/strlcpy.c

- **Other Miscellaneous String Routines: Checking To See If A String Looks Like It Might Be A Number:** We used the Rosetta code implementation of isNumeric.c, see https://rosettacode.org/wiki/Determine_if_a_string_is_numeric#C

- **Finding The Position (Index) of A Complete String In A List ("Array") of String Values:** We're using: https://rosettacode.org/wiki/Search_a_list#C

- **Extract A Substring From A Larger String (Given A Starting Point, Length of Substring):** We're using code shown at https://stackoverflow.com/questions/2114377/strings-in-c-how-to-get-substring for this

- **Replacing A Substring That's Part of a String:** We're using https://creativeandcritical.net/str-replace-c for this

Function Arguments and Returned Values: Some of the languages used in this project, such as C, are only able to return a single value from a function. Yes, you can overcome that limitations by creating data structures and returning just a single combined struct. However, doing so adds complexity and opportunities for error, the opposite of what's intended.

Thus, by way of compromise, in at least some cases we used global variables to ensure some commonly needed bits were consistently available and updateable. Some of you may not like that approach; if so, feel free to refactor out those globals. With this note, we're explicitly alerting you to the fact that they're there.

Dynamic Vs Static Variables: Another area where errors commonly occur in C relates to the use dynamically allocated memory. A nice summary of common errors can be seen at "C/C++ Memory Corruption And Memory Leaks," see http://www.yolinux.com/TUTORIALS/C++MemoryCorruptionAndMemoryLeaks.html

Since these programs are "one-shot" routines, we didn't worry too much about inefficiency or inelegance and have just created static arrays where doing that is feasible and offered opportunities for simplification.

The "Take Away" For This Section: If all of the above seems medieval (and it can be), consider using one of the script-based alternatives to C such as Ruby, Python or Perl. It can be somewhat unfair to compare total lines of code since things like commenting and code structuring can greatly expand (or reduce) the size of raw code, but we'd still note:

```
$ cat *.c | wc -l    (manually subtracting out 2665 lines of reformatted parson.c and 242 lines of parson.h)
4238 [-2665-242=1331]
$ cat *.pl | wc -l
612
$ cat *.rb | wc -l
533
$ cat *.py | wc -l
471
```

App III-2. Code Formatting

Speaking of code formatting, we made a conscious effort to tidy the code for this project. To do that we selected:

Language	Tool	Settings File	Tool Home
• C	**uncrustify**	~/.uncrustify.cfg	https://github.com/uncrustify/uncrustify
	Processed with: `$ uncrustify -no-backup $(find . -name "*.[ch]")`		
• Perl	**perlcritic**	.perlcritic	https://metacpan.org/pod/perlcritic
	Processed with: `$ perlcritic --brutal --profile .perlcritic *.pl`		
• Python	**pylint**	.pylintrc	https://www.pylint.org/
	Processed with: `$ pylint --rcfile=.pylintrc sie_get_py.py`		
• Ruby:	**Rubocob**	.rubocop.yml	https://docs.rubocop.org/en/stable/
	Processed with: `$ rubocop -c .rubocop.yml sie_get_rb.rb`		

Even if your preferred policy doesn't perfectly align with ours, we hope the use of a consistent format is still helpful.

Appendix IV. sie_get_rb client source code

```
$ cat sie_get_rb.rb
#!/usr/bin/env ruby

require 'Typhoeus'
require 'json'
require 'date'
require 'set'

# the SIE-Batch API endpoint is NOT routinely globally accessible
#
# your IP must be explicitly whitelisted (or use a proxy server that
# has been authorized access). If using a proxy server, ssh into that
# bastionhost before running this script:
#
#    $ ssh -D 127.0.0.1:1080 bastionhostname

useproxy = true
endpoint = 'batch.sie-remote.net'

# FUNCTIONS
#
# functions appear in order by dependency (to-be called fn precedes calling fn)

# ------------------------------------------------------------------------------
def getkeyfromlocalfile
  # build the filepath pointing at the SIE-Batch API key in the user's homedir
  filepath = Dir.home + '/.sie-get-key.txt'

  # make sure the SIE-Batch API key file exists
  unless File.exist?(filepath)
    abort('\nERROR:\n\n  No SIE-Batch API keyfile at ' + filepath)
  end

  # open that key file and read the SIE-Batch API key
  file = File.open(filepath)
  myapikeyval = file.read.chomp
  file.close

  return myapikeyval
end

# ------------------------------------------------------------------------------
def make_query(useproxy, queryUrl, queryParams)
  # actually handle making the http/https query against the endpoint

  myqueryURL = queryUrl.dup.to_s
  myqueryParams = queryParams.dup.to_s

  # (other Typhoeus options defined:
  # https://github.com/typhoeus/ethon/blob/master/lib/ethon/curls/options.rb )

  if useproxy == true
    request = Typhoeus::Request.new( \
      myqueryURL, \
      method: :post, \
      headers: { 'Application' => 'application/json', \
                 'User-Agent' => 'sie_get_rb/1.0' }, \
      body: myqueryParams, \
      connecttimeout: 10, \
      ipresolve: :v4, \
      proxy: 'http://127.0.0.1:1080', \
      proxytype: 'socks5' \
    )
  else
    request = Typhoeus::Request.new( \
      myqueryURL, \
      method: :post, \
      headers: { 'Application' => 'application/json', \
                 'User-Agent' => 'sie_get_rb/1.0' }, \
      body: myqueryParams, \
      connecttimeout: 10, \
      use_ssl: :all, \
      ssl_verifypeer: true \
    )
  end

  # try validating the API key up to 3 times
  response = nil
  tries = 3
  while tries >= 1
    response = request.run

    # if successful, break out of the attempt loop and proceed
    if response.code == 200
      break
    elsif response.code == 403
      abort("Response Code 403: Bad SIE-Batch API Key?")
    end

    # if not, try again at least two more times
    tries -= 1
    if response.code.zero? || tries.zero?
      abort('HTTP error (retried three times)')
    end

  end
  return response.body
end

# ------------------------------------------------------------------------------
def validateapikeyonline(endpoint, useproxy)
  # after getting the API key from the local key file, is it valid? We'll check
  # online

  myapikeyval = getkeyfromlocalfile

  queryURL = 'https://' + endpoint + '/siebatchd/v1/validate'
  queryParams = { 'apikey' => myapikeyval }.to_json
  response = make_query(useproxy, queryURL, queryParams)
  json_query_object = JSON.parse(response)
  status = json_query_object['_status']
  return status
end

# -------------------------------------------------------------------------------
def format_and_printout_the_chan_list(chan_list)
  # this routine actually formats and writes out the channel listing
  new_hash = {}
  chan_list.each do |key, value|
    tempstring = (key.to_s + value.to_s)
    tempstring = tempstring.sub('ch', '')
    tempstring = tempstring.sub('{"description"=>"', ' ')
    tempstring = tempstring.sub('"}', '')
    delim = tempstring.index(' ')
    strleng = tempstring.length
    # left pad the channel number to a width of 3
    keystring = tempstring[0, delim].rjust(3, ' ')
    valstring = tempstring[delim + 1, strleng]
    new_hash[keystring] = valstring
  end

  new_hash = new_hash.sort
  new_hash.each do |keystring, valstring|
    printf("%s  %s\n", keystring, valstring)
  end
end

# ------------------------------------------------------------------------------
def list_channels(endpoint, useproxy)
  # this routine gets the list of live channels from the API server

  myapikeyval = getkeyfromlocalfile

  queryUrl = 'https://' + endpoint + '/siebatchd/v1/validate'
  queryParams = { 'apikey' => myapikeyval }.to_json
  response = make_query(useproxy, queryUrl, queryParams)
  json_query_object = JSON.parse(response)
  obj = json_query_object['profile']['siebatch']
  format_and_printout_the_chan_list(obj)
  exit(0)
end

# ------------------------------------------------------------------------------
# https://stackoverflow.com/questions/23169510/adding-commas-to-numbers-in-ruby
#
def add_commas(num_string)
  # utility routine to add commas to numeric strings for readability
  num_string.reverse.scan(/\d{3}|.+/).join(',').reverse
end

# ------------------------------------------------------------------------------
#
def format_and_printout_chan_time_limits(chan, earliest_time_string,
  latest_time_string, volume)
  # take those channel status parameters and print them out in a little report

  chan = chan.rjust(4, ' ')
  volume = add_commas(volume.to_s)
  # could add a header, but it's pretty self-obvious, right?
  # printf('chan  earliest datetime    latest datetime       octets\n')
  printf("%s  \"%s\"  \"%s\"  %s\n", chan, earliest_time_string,
    latest_time_string, volume)
end

# ------------------------------------------------------------------------------
def show_intervals(endpoint, useproxy, chan_to_check)
  # each channel is available for a range of dates, and has an associated
  # data volume (so you'll know if you're potentially startng a huge download)

  params = '{"apikey":"' +
    getkeyfromlocalfile +
    '","channels":[' +
    chan_to_check +
    ']}'
  url = 'https://' + endpoint + '/siebatchd/v1/siebatch/chdetails'

  response = make_query(useproxy, url, params)

  json_query_object = JSON.parse(response)

  chan_to_check = 'ch' + chan_to_check
  earliest_time_string =
    json_query_object['channels'][chan_to_check]['earliest']
  latest_time_string = json_query_object['channels'][chan_to_check]['latest']
  chan_size = json_query_object['channels'][chan_to_check]['size']

  # we return results as an array since Ruby is a return-one-arg-only language
  timearray = Array.new(3)
  timearray[0] = earliest_time_string
  timearray[1] = latest_time_string
  timearray[2] = chan_size
  return timearray
end

# ----------------------------------------------------------------------------
def fixup_ending_datetime_in_the_future
  # Stop datetime from being out of range. Replacing with current GMT time.

  format = '%Y-%m-%d %H:%M:%S'

  epochseconds = Time.now
  extraseconds = epochseconds.strftime('%S')
  endingtime_seconds = epochseconds.gmtime.to_i - extraseconds.to_i
  endingtime = Time.at(endingtime_seconds).gmtime.to_datetime
  enddatetime = endingtime.strftime(format)

  return enddatetime
end

# -----------------------------------------------------------------------------
def check_channel(endpoint, useproxy, chanflagstring, startdatetime, \
  enddatetime)
  # make sure that the channel is available and the dates are in-range

  # get the available datetime range for this channel
  (earliest_time_string, latest_time_string) = show_intervals(endpoint, \
    useproxy, chanflagstring)

  # convert the requested and available START datetimes into Un*x seconds
  requested_start_seconds = DateTime.parse(startdatetime).to_time.to_i
  earliest_date_seconds = DateTime.parse(earliest_time_string).to_time.to_i

  # convert the requested and available END datetimes into Un*x seconds
  requested_stop_seconds = DateTime.parse(enddatetime).to_time.to_i
  latest_date_seconds = DateTime.parse(latest_time_string).to_time.to_i

  # stop datetime must be later than start date time
  if (requested_stop_seconds - requested_start_seconds).negative?
    abort('Start datetime must be earlier than stop datetime')
  end

  # requested start datetime must be >= the earliest data available
  if (requested_start_seconds - earliest_date_seconds).negative?
    abort('Start datetime out of range. Must be no earlier than ' + \
      earliest_time_string)
  end

  # end datetime may not be in the future; we fix it up by clamping
  # the end time to now if it is asking for something in the future
  if (requested_stop_seconds - latest_date_seconds).positive?
    enddatetime = fixup_ending_datetime_in_the_future
  end

  # handle return of the (potentially updated) datetimes here
  # another case of needing to pack multiple returns into an array
  timearray = Array.new(2)
  timearray[0] = startdatetime
  timearray[1] = enddatetime
  return timearray
end

# ----------------------------------------------------------------------------
def validate_input_time_date_format(mydatetime)
  # parameter is a datetime that we want to format check
  # if invalid, abort run
  # if valid, return the validated (but unchanged) datetime (could skip
  # doing this for now, but at some point we might decide to fix up bad
  # string formatting as a convenience to the user, so...)

  # check the format with a regex
  if !(mydatetime =~ /\A\d{4}-\d{2}-\d{2}\ \d{2}:\d{2}:\d{2}\Z/).nil?
    # good starting time format
  else
    puts('bad starting time format -- must be "YYYY-MM-DD HH:MM:SS"')
    abort('')
  end

  return mydatetime
end

# -----------------------------------------------------------------------------
def zero_unused_seconds(mydatetime)
  # since SIE-Batch API does not care about seconds, we force them to zero
  mynewdatetime = mydatetime.sub!(/..$/, "00")
  return mynewdatetime
end

# -----------------------------------------------------------------------------
def convert_relative_times_to_real_datetimes(enddatetime)
  format = '%Y-%m-%d %H:%M:%S'

  # the new "real" ending datetime comes from the current GMT time
  epochseconds = Time.now
  extraseconds = epochseconds.strftime('%S')
  # remove any "extra" seconds that would make the formatted time
  # non-zero
  endingtime_seconds = epochseconds.gmtime.to_i - extraseconds.to_i

  # we compute the "real" starting datetime by offsetting backwards
  # we get minutes from the user, but need seconds
  mysecondsback = enddatetime.to_i * 60
  startseconds = endingtime_seconds - mysecondsback.to_i
  # convert from epoch seconds back to datetime object
  stageddatetime = Time.at(startseconds).gmtime.to_datetime
  # format the date time object as a time string
  startdatetime = stageddatetime.strftime(format)

  # repeat conversion and formatting for the ending time
  endingtime = Time.at(endingtime_seconds).gmtime.to_datetime
  enddatetime = endingtime.strftime(format)

  # pass as an array
  timearray = Array.new(2)
  timearray[0] = startdatetime
  timearray[1] = enddatetime
  return timearray
end

# ------------------------------------------------------------------------------
def fix_times
  # starting and ending time arguments come in from the command line
  # so we don't pass them in as parameters when calling fix_times

  # standardize the starting and ending times from the command line
  startdatetime = ARGV[1].dup
  enddatetime   = ARGV[2].dup

  # if relative times, replace the ending time with the current GMT time
  # set the starting time back by the specified number of minutes

  # ensure results returned from convert_relative_times_to_real_datetimes
  # are correctly scoped/available routine-wide
  timearray = Array.new(2)

  if startdatetime == 'now'

    timearray = convert_relative_times_to_real_datetimes(enddatetime)

  else

    # we have real timedate stamps for starting and ending datetimes

    # process the starting datetime value...
    # is the datetime value written in the right format?
    # also zero the seconds if present (SIE-Batch API doesn't use them)
    validate_input_time_date_format(startdatetime)
    startdatetime = zero_unused_seconds(startdatetime)

    # repeat for the ending datetime value...
    validate_input_time_date_format(enddatetime)
    enddatetime = zero_unused_seconds(enddatetime)

    timearray[0] = startdatetime
    timearray[1] = enddatetime
  end

  return timearray
end

# ------------------------------------------------------------------------------
def check_if_chan_is_an_nmsg_chan(chanflagstring)
  # some channels use JSON lines format, some use nmsg
  # this routine defines which is which
  nmsg_channels = ['204', '206', '207', '208', '221'].to_set

  if nmsg_channels.include?(chanflagstring)
```

```
    filetype = '.nmsg'
  else
    filetype = '.jsonl'
  end

  return filetype
end

# ----------------------------------------------------------------------------
def build_filename(startdatetime, enddatetime)
  # construct the filename from the command line arguments and return it

  chanflagstring = ARGV[0].dup
  string1 = startdatetime.dup
  string2 = enddatetime.dup

  # don't want spaces in the filename; could use something else, like 'T'
  # here instead, I suppose
  string1.sub!(' ', '@')
  string2.sub!(' ', '@')

  filetype = check_if_chan_is_an_nmsg_chan(chanflagstring)
  outputfilename = 'sie-ch' +
    chanflagstring.to_s +
    '-{' +
    string1.to_s +
    '}-{' +
    string2.to_s +
    '}' +
    filetype.to_s
  return outputfilename
end

# ---------------------------------------------------------------------------
def print_usage_info
  puts(
    '''
Usage:

  sie_get_rb channel "now" minutesBack
  Example: sie_get_rb 212 now 15

    OR

  sie_get_rb channel "startdatetime" "enddatetime"
  Example: sie_get_rb 212 "2020-01-07 00:13:00" "2020-01-07 00:28:00"

Convenience functions:

  Check SIE-Batch API key:                       sie_get_rb checkkey
  Get a listing of channels:                     sie_get_rb channels
  Get datetime range and volume for a channel:  sie_get_rb 212

Notes:

  Datetimes are UTC and must be quoted. (Current UTC datetime: $ date -u )
  Zero pad any single digit months, days, hours, minutes or seconds.
  Seconds must be entered as part of the UTC datetimes (but are ignored)
  Ending datetime in the future? It will be clamped to current datetime.

  minutesBack must be >= 1 and a whole number
'''
  )
  exit(1)
end

# ----------------------------------------------------------------------------
# https://stackoverflow.com/questions/1235863/how-to-test-if-a-string-is-basically-an-
integer-in-quotes-using-ruby
# note: rubocop complains, but since I've basically taken this code verbatim,
# I'm leaving it as is

# rubocop:disable Naming/PredicateName
# rubocop:disable Style/RedundantSelf
class String
  def is_integer?
    self.to_i.to_s == self
  end
end
# rubocop:enable Naming/PredicateName
# rubocop:enable Style/RedundantSelf

# ----------------------------------------------------------------------------
def one_real_arg(endpoint, useproxy, first_arg)
  # because users can ask for channel info by specifying a channel
  # number, we need to know the list of defined channel numbers
  defined_channels = ['24', '25', '27', '42', '80', '114', '115', \
    '204', '206', '207', '208', '211', '212', '213', '214', '221'].to_set

  if first_arg == 'channels'
    # list channels for the user
    list_channels(endpoint, useproxy)
    exit(0)
  elsif first_arg == 'checkkey'
    # check the user's key for validity
    status = validateapikeyonline(endpoint, useproxy)
    puts('API key status is ' + status)
    exit(0)
  elsif defined_channels.include?(first_arg)
    # list details about the specified channel
    (earliest, latest, datasize) = show_intervals(endpoint, \
      useproxy, first_arg)
    format_and_printout_chan_time_limits(first_arg, earliest, \
      latest, datasize)
    exit(0)
  elsif !defined_channels.include?(first_arg) && first_arg.is_integer?
    ### the requested channel is not one we offer, so...
    print("Channel not offered via this script\n")
    exit(0)
  else
    print_usage_info
  end
  exit(0)
end

# ----------------------------------------------------------------------------
def three_real_args(endpoint, useproxy)
  # We appear to actually be working on retrieving data....
  chanflagstring = ARGV[0]

  # beat the datetimes into shape, if need be
  (startdatetime, enddatetime) = fix_times
  # be sure the requested channel and requested time range are good to go
  # (check the online server to make sure the channel exists, and the
  # start and end datetimes are sane)
  (startdatetime, enddatetime) = check_channel(endpoint, useproxy, \
  chanflagstring, startdatetime, enddatetime)

  # get the output filename
  outputfilename = build_filename(startdatetime, enddatetime)

  # actually get the SIE-Batch data
  param = JSON.generate\
    ({ 'apikey' => getkeyfromlocalfile, \
       'channel' => chanflagstring.to_i, \
       'start_time' => startdatetime, \
       'end_time' => enddatetime })

  url = 'https://' + endpoint + '/siebatchd/v1/siebatch/chfetch'
  response2 = make_query(useproxy, url, param)

  # write the SIE-Batch data out to our file
  File.open(outputfilename, 'wb') do |file|
    file.write(response2)
  end

  exit(0)
end

# ===============================================================================
# main

command_line_arg_count = ARGV.length
if command_line_arg_count >= 1
  first_arg = (ARGV[0]).dup
end

if command_line_arg_count == 1
  one_real_arg(endpoint, useproxy, first_arg)
  exit(0)
elsif command_line_arg_count == 3
  three_real_args(endpoint, useproxy)
  exit(0)
elsif command_line_arg_count.zero? || (command_line_arg_count == 2) || \
  (command_line_arg_count >= 4)
  print_usage_info
  exit(0)
end
```

```
$ cat .rubocop.yml
# get current or go home
AllCops:
  TargetRubyVersion: 2.6

Layout/ArgumentAlignment:
  Enabled: false

Layout/ArrayAlignment:
  Enabled: false

Layout/MultilineOperationIndentation:
  Enabled: false

Layout/ParameterAlignment:
  Enabled: false

Lint/AssignmentInCondition:
  AllowSafeAssignment: false

Lint/ImplicitStringConcatenation:
  Enabled: false

# http://wiki.c2.com/?AbcMetric
Metrics/AbcSize:
  Max: 30

Metrics/CyclomaticComplexity:
  Max: 7

Metrics/MethodLength:
  Max: 45

Metrics/PerceivedComplexity:
  Max: 9

Naming/VariableName:
  Enabled: false

Naming/MethodParameterName:
  Enabled: false

Style/ConditionalAssignment:
  Enabled: false

Style/FormatStringToken:
  Enabled: false

# We prefer to have the interpreter on the first line
Style/FrozenStringLiteralComment:
  Enabled: false

# We use a few intentionally, e.g., for $ENDPOINT and $USEPROXY
Style/GlobalVars:
  Enabled: false

Style/GuardClause:
  Enabled: false

Style/IfUnlessModifier:
  Enabled: false
```

```
Style/ParenthesesAroundCondition:
  AllowSafeAssignment: false

Style/RedundantReturn:
  Enabled: false

Style/StringLiterals:
  Enabled: false

Style/SymbolArray:
  Enabled: true

Style/WordArray:
  EnforcedStyle: brackets
```

Appendix V. sie_get_py client source code

```
$ cat sie_get_py.py
#!/usr/bin/env python3
"""This script demonstrates use of the SIE-Batch API from Python3"""

# Lint with $ pylint sie_get_py.py      (assumes .pylintrc file in dir)

import calendar
import datetime
from datetime import datetime
from io import BytesIO
import json
from os import path
from pathlib import Path
import re
import sys
import time
from time import strftime
import pycurl

endpoint = 'batch.sie-remote.net'
useproxy = False    # note: 'false' will not work

# ---------------------------------------------------------------------------
#
def getkeyfromlocalfile():
    """Retrieves the SIE-Batch API key"""

    filepath = str(Path.home()) + "/.sie-get-key.txt"

    if not path.exists(filepath):
        print("\nERROR:\n\n  No SIE-Batch API keyfile at "+filepath)
        sys.exit(1)

    with open(filepath) as stream:
        myapikey = stream.read().rstrip()

    return myapikey

# ---------------------------------------------------------------------------
#
def make_query(url, useproxy, params, outputfilename):
    """make query"""

    if outputfilename != '-999':
        try:
            f = open(outputfilename, "wb")
        except IOError:
            sys.exit("error opening output file for results")
    else:
        buffer = BytesIO()

    c = pycurl.Curl()
    c.setopt(pycurl.URL, url)
    c.setopt(pycurl.HTTPHEADER, ['Content-Type: application/json'])
    c.setopt(pycurl.POST, True)
    c.setopt(pycurl.POSTFIELDS, params)
    c.setopt(pycurl.FOLLOWLOCATION, 1)
    c.setopt(pycurl.CONNECTTIMEOUT, 300)
    c.setopt(pycurl.TIMEOUT, 86400)
    c.setopt(pycurl.USERAGENT, 'sie_get_py/1.0')
    # we're going to write the actual data files directly to the outputfile
    # other stuff (apikey check, channel listing, etc.) we're just going
    # to write to a buffer (and then read that))
    if outputfilename == '-999':
        c.setopt(pycurl.WRITEDATA, buffer)
    else:
        c.setopt(pycurl.WRITEDATA, f)

    if useproxy:
        c.setopt(pycurl.PROXY, '127.0.0.1')
        c.setopt(pycurl.PROXYPORT, 1080)
        c.setopt(pycurl.PROXYTYPE, pycurl.PROXYTYPE_SOCKS5)

    tries = 3
    rc = ''

    while tries >= 1:
        c.perform()
        rc = c.getinfo(c.RESPONSE_CODE)

        # if writing to a buffer, we need to extract results and change
        # from a bytestring to a string
        if outputfilename == '-999':
            body = buffer.getvalue()
            content = body.decode('iso-8859-1')

        # successful transfer? if so, break out of the loop
        # if not, try it again
        #pylint: disable=no-else-break
        if rc == 200:
            break
        else:
            print('Problem in make_query: response code='+str(rc))

    #pylint: disable=no-else-return
    if outputfilename == '-999':
        return content
    else:
        sys.exit(0)
    #pylint: enable=no-else-break

# ------------------------------------------------------------------------------
def validateapikeyonline(endpoint, useproxy):
    """ check the API key for validity on the live SIE-Batch API server """
    myapikeyval = getkeyfromlocalfile()
    params = {'apikey' : myapikeyval}
    params2 = json.dumps(params)
    queryURL = 'https://' + endpoint + '/siebatchd/v1/validate'
    returned_content = make_query(queryURL, useproxy, params2, '-999')
    returned_content_json_format = json.loads(returned_content)
    status = returned_content_json_format['_status']
    return status

# ------------------------------------------------------------------------------
#
def format_and_printout_the_chan_list(chan_list):
    """ we have the channel data, now format and print it out in a report """
    chan_list_json_format = json.loads(chan_list)

    new_hash = {}
    #pylint: disable=unused-variable
    for k, v in chan_list_json_format.items():
        #pylint: enable=unused-variable
        keystring = k.replace('ch', '')
        keystring = keystring.rjust(3, ' ')
        actual_val = chan_list_json_format[k]['description']
        new_hash[keystring] = actual_val

    for k, v in sorted(new_hash.items()):
        print(k, new_hash[k])

    sys.exit(0)

# ------------------------------------------------------------------------------
#
def list_channels(endpoint, useproxy):
    """ retrieve a list of channels from the server """
    myapikeyval = getkeyfromlocalfile()
    params = {'apikey' : myapikeyval}
    params2 = json.dumps(params)
    queryURL = 'https://' + endpoint + '/siebatchd/v1/validate'
    returned_content = make_query(queryURL, useproxy, params2, '-999')
    returned_content_json_format = json.loads(returned_content)
    extract_bit = returned_content_json_format['profile']['siebatch']
    json_query_object = json.dumps(extract_bit)
    format_and_printout_the_chan_list(json_query_object)
    sys.exit(0)

# ------------------------------------------------------------------------------
#
def format_and_printout_chan_time_limits(chan, earliest_time_string, \
    latest_time_string, volume):
    """ print a summary of available channel date range and volume """

    # take the channel status parameters and print them out in a little report
    chan = chan.rjust(4)
    if int(volume) >= 4:
        volume = '{:,d}'.format(int(volume))
        volume = volume.rjust(16)

    # could add a header, but it's pretty self-obvious, right?
    # printf('chan  earliest datetime    latest datetime       octets\n')
    print(chan+'  "'+earliest_time_string+'"  '+\
                 '"'+latest_time_string+'"  '+volume)
    sys.exit(0)

# ------------------------------------------------------------------------------
#
def show_intervals(endpoint, useproxy, chan_to_check):
    """ get the starting and stopping date range and volume """
    # with square brackets
    newchan_to_check = '[' + chan_to_check + ']'
    # no brackets, but with ch literal prefix
    chan_with_prefix = 'ch' + chan_to_check

    myapikeyval = getkeyfromlocalfile()
    params = {'apikey': myapikeyval, 'channels': newchan_to_check}
    params2 = json.dumps(params)

    # {"channels":"[212]","apikey":"blah"}  needs to become
    # {"channels":[212],"apikey":"blah"}    (e.g., no quotes around [chan])
    params2 = params2.replace('"[', '[')
    params2 = params2.replace(']"', ']')

    url = 'https://' + endpoint + '/siebatchd/v1/siebatch/chdetails'
    response = make_query(url, useproxy, params2, '-999')
    decoded_results = json.loads(response)

    earliest_time_string = \
        decoded_results['channels'][chan_with_prefix]['earliest']
    latest_time_string = \
        decoded_results['channels'][chan_with_prefix]['latest']
    size_string = \
        decoded_results['channels'][chan_with_prefix]['size']

    return (earliest_time_string, latest_time_string, size_string)

# ---------------------------------------------------------------------------
#
def fixup_ending_datetime_in_the_future():
    """ if the ending date is in the future, reel it back in! """
    # Replace future times with the current GMT time.
    # The following returns a datetime structure
    epochseconds = time.gmtime()
    enddatetime = time.strftime('%Y-%m-%d %H:%M:%S', epochseconds)
    enddatetime2 = re.sub(r'..$', '00', enddatetime)
    return enddatetime2

# ---------------------------------------------------------------------------
#
def string_fmt_time_to_seconds(string_format_time):
    """ utility function to convert a string format time to epoch seconds """
    dt = datetime.strptime(string_format_time, "%Y-%m-%d %H:%M:%S")
    epoch_seconds = calendar.timegm(dt.utctimetuple())
    return epoch_seconds

# ----------------------------------------------------------------------------
#
def check_channel(endpoint, useproxy, chanflagstring, startdatetime, \
    enddatetime):
    """ make sure that the channel is available and the dates are in-range """

    # get the available datetime range for this channel
    #pylint:disable=unused-variable
    (earliest_time_string, latest_time_string, chan_to_check) = \
        show_intervals(endpoint, useproxy, chanflagstring)
    #pylint:enable=unused-variable

    # convert the requested and available start datetimes into Un*x seconds
    requested_start_seconds = string_fmt_time_to_seconds(startdatetime)
    earliest_date_seconds = string_fmt_time_to_seconds(earliest_time_string)
    requested_stop_seconds = string_fmt_time_to_seconds(enddatetime)
    latest_date_seconds = string_fmt_time_to_seconds(latest_time_string)

    # start datetime must be earlier than stop date time
    if (requested_stop_seconds - requested_start_seconds) < 0:
        sys.exit('Start datetime must be earlier than stop datetime')

    # start datetime may not be earlier than earliest data available
    if (requested_start_seconds - earliest_date_seconds) < 0:
        sys.exit('Start datetime out of range. Must be no earlier than ' + \
            earliest_time_string)
    # end datetime may not be in the future
    if (requested_stop_seconds - latest_date_seconds) > 0:
        enddatetime = fixup_ending_datetime_in_the_future()

    return (startdatetime, enddatetime)

# -------------------------------------------------------------------------------
#
def validate_input_time_date_format(mydatetime):
    """ make sure the user has followed the required datetime format """
    # parameter is datetime to format check. if invalid, abort run.
    # if valid, return the validated (but unchanged) datetime (could skip
    # doing this for now, but at some point we might decide to fix up bad
    # string formatting as a convenience to the user, so...)

    # check the format with a regex
    if not(re.match(r'/\A\d{4}-\d{2}-\d{2}\ \d{2}:\d{2}:\d{2}\Z/', \
        mydatetime), mydatetime):
        print("bad starting time format -- must be \"YYYY-MM-DD HH:MM:SS\"\n")
        sys.exit(1)

    return mydatetime

# --------------------------------------------------------------------------------
#
def zero_unused_seconds(mydatetime):
    """ if seconds are non-zero in the time stamps, zero them out """
    # since SIE-Batch API does not care about seconds, we set them to zero
    mydatetime2 = re.sub(r'..$', '00', mydatetime)
    return mydatetime2

# --------------------------------------------------------------------------------
#
def convert_relative_times_to_real_datetimes(minutesback):
    """ one option is relative times; if we get one, make it a real time """
    # in relative format, the initial "ending time" is actually the minutes
    # worth of data we want to retrieve
    # the "real" ending datetime will be created from the current GMT time
    # we will be doing math on the epoch seconds

    myformat = '%Y-%m-%d %H:%M:%S'

    endingtime = time.gmtime()
    epochseconds = calendar.timegm(endingtime)

    # now compute the formatted ending date time in standard YYYY-MM-DD HH:MM:SS
    enddatetime = strftime(myformat, endingtime)

    # find just the seconds from that string
    extraseconds = int(enddatetime[-2:])
    # subtract the seconds from the full datetime to end up with 00 seconds
    endingtime_seconds = int(epochseconds) - extraseconds

    # let's now work on the starting time
    # we compute the "real" starting datetime by offsetting backwards
    # our to-be-modified datetime is in epoch seconds, so convert min to seconds
    mysecondsback = int(minutesback) * 60
    startseconds = endingtime_seconds - mysecondsback
    startdatetime = strftime(myformat, time.gmtime(startseconds))
    enddatetime = strftime(myformat, time.gmtime(endingtime_seconds))
    return (startdatetime, enddatetime)

# -----------------------------------------------------------------------------
#
def fix_times():
    """ handles calling the rest of the routines to fix up times """
    # arguments come in from the command line so we don't pass them in

    # chanflagstring = str(sys.argv[1])
    startdatetime = str(sys.argv[2])
    enddatetime = str(sys.argv[3])

    # if relative times, replace the ending time with the current GMT time
    # set the starting time back by the specified number of minutes
    if startdatetime == 'now':
        (startdatetime, enddatetime) = \
            convert_relative_times_to_real_datetimes(enddatetime)
    else:
        # we have real timedate stamps for starting and ending datetimes
        # process the starting datetime value...
        # correctly written datetime value?
        # also zero the seconds if present (SIE-Batch API doesn't use them)
        validate_input_time_date_format(startdatetime)
        startdatetime = zero_unused_seconds(startdatetime)

        # repeat for the ending datetime value...
        validate_input_time_date_format(enddatetime)
        enddatetime = zero_unused_seconds(enddatetime)
    return (startdatetime, enddatetime)

# ----------------------------------------------------------------------------
#
# https://stackoverflow.com/questions/1265665/how-can-i-check-if-a-string-represents-an-
int-without-using-try-except
#
def isInt_try(v):
    """ convenience function to see if a string might be integer-ish """
    # pylint: disable=unused-variable,multiple-statements,bare-except
    try: i = int(v)
    except: return False
    return True

# -----------------------------------------------------------------------------
#
def build_filename(chanflagstring, startdatetime, enddatetime):
    """construct the filename from the command line arguments and return it"""

    string1 = startdatetime.replace(' ', '@')
    string2 = enddatetime.replace(' ', '@')

    nmsgchannels = ["204", "206", "207", "208", "221"]
    if chanflagstring in nmsgchannels:
        filetype = ".nmsg"
    else:
        filetype = ".jsonl"

    outputfilename = "sie-ch" + chanflagstring + "-{" + string1 + \
        "}-{" + string2 + "}" + filetype

    return outputfilename
```

```
# ----------------------------------------------------------------------------
#
def print_usage_info():
    """ deliver a succinct usage summary if needed """
    print('''
Usage:

  sie_get_py channel "now" minutesBack
  Example: sie_get_py 212 now 15

OR

  sie_get_py channel "startdatetime" "enddatetime"
  Example: sie_get_py 212 "2020-01-07 00:13:00" "2020-01-07 00:28:00"

Convenience functions:

  Check SIE-Batch API key:                     sie_get_py checkkey
  Get a listing of channels:                   sie_get_py channels
  Get datetime range and volume for a channel:  sie_get_py 212

Notes:

  Datetimes are UTC and must be quoted. (Current UTC datetime: $ date -u )
  Zero pad any single digit months, days, hours, minutes or seconds.
  Seconds must be entered as part of the UTC datetimes (but are ignored)
  Ending datetime in the future? It will be clamped to current datetime.
         ''')
    sys.exit(1)

# ----------------------------------------------------------------------------
#
def one_real_arg(endpoint, useproxy, first_arg):
    """ sometimes we only see one option on the command line; process it """
    defined_channels = {'24', '25', '27', '42', '80', '114', '115', \
        '204', '206', '207', '208', '211', '212', '213', '214', '221'}

    if first_arg == 'channels':
        # list channels for the user
        list_channels(endpoint, useproxy)
        sys.exit(0)

    elif first_arg == 'checkkey':
        # check the user's key for validity
        status = validateapikeyonline(endpoint, useproxy)
        print("API key status is "+status)
        sys.exit(0)

    elif (isInt_try(first_arg) and (first_arg in defined_channels)):
        # list details about the specified channel
        (earliest, latest, datasize) = show_intervals(endpoint, \
            useproxy, first_arg)
        format_and_printout_chan_time_limits(first_arg, earliest, \
            latest, datasize)
        sys.exit(0)

    elif (not(first_arg in defined_channels) and (isInt_try(first_arg))):
        # the requested channel is not one we offer, so...
        print("Channel not offered via this script")
        sys.exit(0)
    else:
        print_usage_info()
        sys.exit(0)

# ------------------------------------------------------------------------------
#
def three_real_args(endpoint, useproxy):
    """ other times we may see three arguments on the command line... """
    chanflagstring = str(sys.argv[1])
    (startdatetime, enddatetime) = fix_times()
    (startdatetime, enddatetime) = check_channel(endpoint, useproxy, \
        chanflagstring, startdatetime, enddatetime)

    outputfilename = build_filename(chanflagstring, startdatetime, enddatetime)
    myapikey = getkeyfromlocalfile()
    params = {"apikey": myapikey, "channel": int(chanflagstring), \
        "start_time": startdatetime, "end_time": enddatetime}
    params2 = json.dumps(params)
    queryURL = "https://" + endpoint + "/siebatchd/v1/siebatch/chfetch"
    make_query(queryURL, useproxy, params2, outputfilename)
    sys.exit(0)

# ==============================================================================
# main

if len(sys.argv) == 1:
    print_usage_info()
    sys.exit(0)

elif len(sys.argv) >= 2:
    first_arg = sys.argv[1]

command_line_arg_count = len(sys.argv)-1

if command_line_arg_count == 1:
    one_real_arg(endpoint, useproxy, first_arg)
    sys.exit(0)

elif command_line_arg_count == 3:
    three_real_args(endpoint, useproxy)
    sys.exit(0)

elif (command_line_arg_count <= 0) or (command_line_arg_count >= 4) or \
    (command_line_arg_count == 2):
    print_usage_info()
    sys.exit(0)
```

```
$ cat .pylintrc
[MASTER]

# A comma-separated list of package or module names from where C extensions may
# be loaded. Extensions are loading into the active Python interpreter and may
# run arbitrary code.
extension-pkg-whitelist=

# Add files or directories to the blacklist. They should be base names, not
# paths.
ignore=CVS

# Add files or directories matching the regex patterns to the blacklist. The
# regex matches against base names, not paths.
ignore-patterns=

# Python code to execute, usually for sys.path manipulation such as
# pygtk.require().
#init-hook=

# Use multiple processes to speed up Pylint. Specifying 0 will auto-detect the
# number of processors available to use.
jobs=0

# Control the amount of potential inferred values when inferring a single
# object. This can help the performance when dealing with large functions or
# complex, nested conditions.
limit-inference-results=100

# List of plugins (as comma separated values of python module names) to load,
# usually to register additional checkers.
load-plugins=

# Pickle collected data for later comparisons.
persistent=yes

# Specify a configuration file.
#rcfile=

# When enabled, pylint would attempt to guess common misconfiguration and emit
# user-friendly hints instead of false-positive error messages.
suggestion-mode=yes

# Allow loading of arbitrary C extensions. Extensions are imported into the
# active Python interpreter and may run arbitrary code.
unsafe-load-any-extension=no

[MESSAGES CONTROL]

# Only show warnings with the listed confidence levels. Leave empty to show
# all. Valid levels: HIGH, INFERENCE, INFERENCE_FAILURE, UNDEFINED.
confidence=

# Disable the message, report, category or checker with the given id(s). You
# can either give multiple identifiers separated by comma (,) or put this
# option multiple times (only on the command line, not in the configuration
# file where it should appear only once). You can also use "--disable=all" to
# disable everything first and then reenable specific checks. For example, if
# you want to run only the similarities checker, you can use "--disable=all
# --enable=similarities". If you want to run only the classes checker, but have
# no Warning level messages displayed, use "--disable=all --enable=classes
# --disable=W".
disable=print-statement,
        parameter-unpacking,
        unpacking-in-except,
        old-raise-syntax,
        backtick,
        long-suffix,
        old-ne-operator,
        old-octal-literal,
        import-star-module-level,
        non-ascii-bytes-literal,
        raw-checker-failed,
        bad-inline-option,
        locally-disabled,
        file-ignored,
        suppressed-message,
        useless-suppression,
        deprecated-pragma,
        use-symbolic-message-instead,
        apply-builtin,
        basestring-builtin,
        buffer-builtin,
        cmp-builtin,
        coerce-builtin,
        execfile-builtin,
        file-builtin,
        long-builtin,
        raw_input-builtin,
        reduce-builtin,
        standarderror-builtin,
        unicode-builtin,
        xrange-builtin,
        coerce-method,
        delslice-method,
        getslice-method,
        setslice-method,
        no-absolute-import,
        old-division,
        dict-iter-method,
        dict-view-method,
        next-method-called,
        metaclass-assignment,
        indexing-exception,
        raising-string,
        reload-builtin,
        oct-method,
        hex-method,
        nonzero-method,
        cmp-method,
        input-builtin,
        round-builtin,
        intern-builtin,
        unichr-builtin,
        map-builtin-not-iterating,
        zip-builtin-not-iterating,
        range-builtin-not-iterating,
        filter-builtin-not-iterating,
        using-cmp-argument,
        eq-without-hash,
        div-method,
        idiv-method,
        rdiv-method,
        exception-message-attribute,
        invalid-str-codec,
        sys-max-int,
        bad-python3-import,
        deprecated-string-function,
        deprecated-str-translate-call,
        deprecated-itertools-function,
        deprecated-types-field,
        next-method-defined,
        dict-items-not-iterating,
        dict-keys-not-iterating,
        dict-values-not-iterating,
        deprecated-operator-function,
        deprecated-urllib-function,
        xreadlines-attribute,
        deprecated-sys-function,
        exception-escape,
        comprehension-escape,
        c-extension-no-member,
        redefined-outer-name

# Enable the message, report, category or checker with the given id(s). You can
# either give multiple identifier separated by comma (,) or put this option
# multiple time (only on the command line, not in the configuration file where
# it should appear only once). See also the "--disable" option for examples.
# enable=c-extension-no-member

[REPORTS]

# Python expression which should return a score less than or equal to 10. You
# have access to the variables 'error', 'warning', 'refactor', and 'convention'
# which contain the number of messages in each category, as well as 'statement'
# which is the total number of statements analyzed. This score is used by the
# global evaluation report (RP0004).
evaluation=10.0 - ((float(5 * error + warning + refactor + convention) / statement) * 10)

# Template used to display messages. This is a python new-style format string
# used to format the message information. See doc for all details.
#msg-template=

# Set the output format. Available formats are text, parseable, colorized, json
# and msvs (visual studio). You can also give a reporter class, e.g.
# mypackage.mymodule.MyReporterClass.
output-format=text

# Tells whether to display a full report or only the messages.
reports=no

# Activate the evaluation score.
score=yes

[REFACTORING]

# Maximum number of nested blocks for function / method body
max-nested-blocks=5

# Complete name of functions that never returns. When checking for
# inconsistent-return-statements if a never returning function is called then
# it will be considered as an explicit return statement and no message will be
# printed.
```

```
never-returning-functions=sys.exit

[LOGGING]

# Format style used to check logging format string. `old` means using %
# formatting, `new` is for `{}` formatting,and `fstr` is for f-strings.
logging-format-style=old

# Logging modules to check that the string format arguments are in logging
# function parameter format.
logging-modules=logging


[SPELLING]

# Limits count of emitted suggestions for spelling mistakes.
max-spelling-suggestions=4

# Spelling dictionary name. Available dictionaries: none. To make it work,
# install the python-enchant package.
spelling-dict=

# List of comma separated words that should not be checked.
spelling-ignore-words=

# A path to a file that contains the private dictionary; one word per line.
spelling-private-dict-file=

# Tells whether to store unknown words to the private dictionary (see the
# --spelling-private-dict-file option) instead of raising a message.
spelling-store-unknown-words=no


[MISCELLANEOUS]

# List of note tags to take in consideration, separated by a comma.
notes=FIXME,
      XXX,
      TODO


[TYPECHECK]

# List of decorators that produce context managers, such as
# contextlib.contextmanager. Add to this list to register other decorators that
# produce valid context managers.
contextmanager-decorators=contextlib.contextmanager

# List of members which are set dynamically and missed by pylint inference
# system, and so shouldn't trigger E1101 when accessed. Python regular
# expressions are accepted.
generated-members=

# Tells whether missing members accessed in mixin class should be ignored. A
# mixin class is detected if its name ends with "mixin" (case insensitive).
ignore-mixin-members=yes

# Tells whether to warn about missing members when the owner of the attribute
# is inferred to be None.
ignore-none=yes
```

```
# This flag controls whether pylint should warn about no-member and similar
# checks whenever an opaque object is returned when inferring. The inference
# can return multiple potential results while evaluating a Python object, but
# some branches might not be evaluated, which results in partial inference. In
# that case, it might be useful to still emit no-member and other checks for
# the rest of the inferred objects.
ignore-on-opaque-inference=yes

# List of class names for which member attributes should not be checked (useful
# for classes with dynamically set attributes). This supports the use of
# qualified names.
ignored-classes=optparse.Values,thread._local,_thread._local

# List of module names for which member attributes should not be checked
# (useful for modules/projects where namespaces are manipulated during runtime
# and thus existing member attributes cannot be deduced by static analysis). It
# supports qualified module names, as well as Unix pattern matching.
ignored-modules=

# Show a hint with possible names when a member name was not found. The aspect
# of finding the hint is based on edit distance.
missing-member-hint=yes

# The minimum edit distance a name should have in order to be considered a
# similar match for a missing member name.
missing-member-hint-distance=1

# The total number of similar names that should be taken in consideration when
# showing a hint for a missing member.
missing-member-max-choices=1

# List of decorators that change the signature of a decorated function.
signature-mutators=


[VARIABLES]

# List of additional names supposed to be defined in builtins. Remember that
# you should avoid defining new builtins when possible.
additional-builtins=

# Tells whether unused global variables should be treated as a violation.
allow-global-unused-variables=yes

# List of strings which can identify a callback function by name. A callback
# name must start or end with one of those strings.
callbacks=cb_,
          _cb

# A regular expression matching the name of dummy variables (i.e. expected to
# not be used).
dummy-variables-rgx=_+$|(_[a-zA-Z0-9_]*[a-zA-Z0-9]+?$)|dummy|^ignored_|^unused_

# Argument names that match this expression will be ignored. Default to name
# with leading underscore.
ignored-argument-names=_.*|^ignored_|^unused_

# Tells whether we should check for unused import in __init__ files.
init-import=no

# List of qualified module names which can have objects that can redefine
# builtins.
redefining-builtins-modules=six.moves,past.builtins,future.builtins,builtins,io


[FORMAT]

# Expected format of line ending, e.g. empty (any line ending), LF or CRLF.
expected-line-ending-format=

# Regexp for a line that is allowed to be longer than the limit.
ignore-long-lines=^\s*(# )?<?https?://\S+>?$

# Number of spaces of indent required inside a hanging or continued line.
indent-after-paren=4

# String used as indentation unit. This is usually "    " (4 spaces) or "\t" (1
# tab).
indent-string='    '

# Maximum number of characters on a single line.
max-line-length=100

# Maximum number of lines in a module.
max-module-lines=1000

# List of optional constructs for which whitespace checking is disabled. `dict-
# separator` is used to allow tabulation in dicts, etc.: {1  : 1,\n222: 2}.
# `trailing-comma` allows a space between comma and closing bracket: (a, ).
# `empty-line` allows space-only lines.
no-space-check=trailing-comma,
               dict-separator

# Allow the body of a class to be on the same line as the declaration if body
# contains single statement.
single-line-class-stmt=no

# Allow the body of an if to be on the same line as the test if there is no
# else.
single-line-if-stmt=no

[SIMILARITIES]

# Ignore comments when computing similarities.
ignore-comments=yes

# Ignore docstrings when computing similarities.
ignore-docstrings=yes

# Ignore imports when computing similarities.
ignore-imports=no

# Minimum lines number of a similarity.
min-similarity-lines=4

[BASIC]

# Naming style matching correct argument names.
argument-naming-style=any

# Regular expression matching correct argument names. Overrides argument-
# naming-style.
```

```
#argument-rgx=

# Naming style matching correct attribute names.
attr-naming-style=any

# Regular expression matching correct attribute names. Overrides attr-naming-
# style.
#attr-rgx=

# Bad variable names which should always be refused, separated by a comma.
bad-names=foo,
          bar,
          baz,
          toto,
          tutu,
          tata

# Naming style matching correct class attribute names.
class-attribute-naming-style=any

# Regular expression matching correct class attribute names. Overrides class-
# attribute-naming-style.
#class-attribute-rgx=

# Naming style matching correct class names.
class-naming-style=PascalCase

# Regular expression matching correct class names. Overrides class-naming-
# style.
#class-rgx=

# Naming style matching correct constant names.
const-naming-style=any

# Regular expression matching correct constant names. Overrides const-naming-
# style.
#const-rgx=

# Minimum line length for functions/classes that require docstrings, shorter
# ones are exempt.
docstring-min-length=-1

# Naming style matching correct function names.
function-naming-style=any

# Regular expression matching correct function names. Overrides function-
# naming-style.
#function-rgx=

# Good variable names which should always be accepted, separated by a comma.
good-names=i,
           j,
           k,
           ex,
           Run,
           _

# Include a hint for the correct naming format with invalid-name.
include-naming-hint=no

# Naming style matching correct inline iteration names.
```

```
inlinevar-naming-style=any

# Regular expression matching correct inline iteration names. Overrides
# inlinevar-naming-style.
#inlinevar-rgx=

# Naming style matching correct method names.
method-naming-style=any

# Regular expression matching correct method names. Overrides method-naming-
# style.
#method-rgx=

# Naming style matching correct module names.
# module-naming-style=any

# Regular expression matching correct module names. Overrides module-naming-
# style.
#module-rgx=

# Colon-delimited sets of names that determine each other's naming style when
# the name regexes allow several styles.
name-group=

# Regular expression which should only match function or class names that do
# not require a docstring.
no-docstring-rgx=^_

# List of decorators that produce properties, such as abc.abstractproperty. Add
# to this list to register other decorators that produce valid properties.
# These decorators are taken in consideration only for invalid-name.
property-classes=abc.abstractproperty

# Naming style matching correct variable names.
variable-naming-style=any

# Regular expression matching correct variable names. Overrides variable-
# naming-style.
#variable-rgx=

[STRING]

# This flag controls whether the implicit-str-concat-in-sequence should
# generate a warning on implicit string concatenation in sequences defined over
# several lines.
check-str-concat-over-line-jumps=no

[IMPORTS]

# List of modules that can be imported at any level, not just the top level
# one.
allow-any-import-level=

# Allow wildcard imports from modules that define __all__.
allow-wildcard-with-all=no

# Analyse import fallback blocks. This can be used to support both Python 2 and
# 3 compatible code, which means that the block might have code that exists
# only in one or another interpreter, leading to false positives when analysed.
analyse-fallback-blocks=no
```

```
# Deprecated modules which should not be used, separated by a comma.
deprecated-modules=optparse,tkinter.tix

# Create a graph of external dependencies in the given file (report RP0402 must
# not be disabled).
ext-import-graph=

# Create a graph of every (i.e. internal and external) dependencies in the
# given file (report RP0402 must not be disabled).
import-graph=

# Create a graph of internal dependencies in the given file (report RP0402 must
# not be disabled).
int-import-graph=

# Force import order to recognize a module as part of the standard
# compatibility libraries.
known-standard-library=

# Force import order to recognize a module as part of a third party library.
known-third-party=enchant

# Couples of modules and preferred modules, separated by a comma.
preferred-modules=

[CLASSES]

# List of method names used to declare (i.e. assign) instance attributes.
defining-attr-methods=__init__,
                      __new__,
                      setUp,
                      __post_init__

# List of member names, which should be excluded from the protected access
# warning.
exclude-protected=_asdict,
                  _fields,
                  _replace,
                  _source,
                  _make

# List of valid names for the first argument in a class method.
valid-classmethod-first-arg=cls

# List of valid names for the first argument in a metaclass class method.
valid-metaclass-classmethod-first-arg=cls

[DESIGN]

# Maximum number of arguments for function / method.
max-args=5

# Maximum number of attributes for a class (see R0902).
max-attributes=7

# Maximum number of boolean expressions in an if statement (see R0916).
max-bool-expr=5

# Maximum number of branch for function / method body.
max-branches=15
```

```
# Maximum number of locals for function / method body.
max-locals=15

# Maximum number of parents for a class (see R0901).
max-parents=7

# Maximum number of public methods for a class (see R0904).
max-public-methods=20

# Maximum number of return / yield for function / method body.
max-returns=6

# Maximum number of statements in function / method body.
max-statements=50

# Minimum number of public methods for a class (see R0903).
min-public-methods=2

[EXCEPTIONS]

# Exceptions that will emit a warning when being caught. Defaults to
# "BaseException, Exception".
overgeneral-exceptions=BaseException,
                       Exception
```

Appendix VI. sie_get_pl client source code

```
$ cat sie_get_pl.pl
#!/usr/local/bin/perl

# Check with:  perlcritic --profile ./.perlcritic sie_get_pl.pl

use strict;
use warnings;

use Carp qw( croak );
use Data::Printer;
use Data::Structure::Util qw( unbless );
use DateTime;
use DateTime::Format::Strptime;
use File::HomeDir;
use JSON;
use LWP;
use LWP::UserAgent;
use TimeDate;
use Time::ParseDate;
use POSIX::strftime::GNU;
use POSIX qw(strftime);

my $NL = "\n";

my $endpoint = 'batch.sie-remote.net';
my $useproxy = '0';   # '0' is false

# FUNCTIONS
#
# functions appear in order by dependency (to-be called precedes calling)

# ------------------------------------------------------------------------------
sub getkeyfromlocalfile
  {
  # build the filepath pointing at the SIE-Batch API key
  my $filepath = File::HomeDir->my_home . '/.sie-get-key.txt';

  # make sure the SIE-Batch API key file exists
  if ( !-e $filepath )
  {
    croak( "ERROR:\n\n  No SIE-Batch API keyfile at \n", $filepath );
  }

  # open that key file and read the SIE-Batch API key
  my $fh;
  open ($fh, '<:encoding(UTF-8)', $filepath) or
    croak ("Could not open SIE-Batch API keyfile $fh\n");
  my $myapikey = <$fh>;
  chomp $myapikey;
  close($fh) or croak("Could not close SIE-Batch API keyfile $fh\n");

  return ($myapikey);
  }

# ------------------------------------------------------------------------------
sub make_query
  {
  my $url = $_[0];
  my $useproxy = $_[1];  # we get $useproxy as a global
  my $params = $_[2];
  my $outputfilename = $_[3];
  my $content;
  my $ua;

  $ua = LWP::UserAgent->new();
    $ua->requests_redirectable(['POST',]);
    $ua->default_header('Content-Type' => 'application/json');
    $ua->default_header('Accept' => 'application/json');
    $ua->timeout(86_400);
    $ua->agent('sie_get_pl/1.0');

  if ($useproxy) {
    $ua->proxy([qw(http https)] => 'socks://127.0.0.1:1080');
    $ua->protocols_allowed(['https','http',]);
  } else {
    $ua->protocols_allowed(['https',]);
    $ua->ssl_opts( verify_hostname => 'true');
  }

  # try validating the API key up to 3 times
  # my @response;

  my $tries = 3;
  my $rc;
  my $msg;

  while ($tries >= 1) {
    my $response = $ua->post( $url, content => $params);
    unbless($response);
    $rc = $response->{'_rc'};
    $msg = $response->{'_msg'};
    $content = $response->{'_content'};

    if ( $rc == 200 ) {
      last;
    } else {
      print "Problem in make_query: $rc\n";
      print "Message: $msg\n";
    }

    # if not, try again at least two more times
    $tries = $tries - 1;
    if ($tries == 0) {
      croak('HTTP error (retried three times)');
    }
  }

  return($content);
  }

# ---------------------------------------------------------------------------
sub validateapikeyonline
  {
  # query the live SIE-Batch API server to confirm the key status

  $endpoint = $_[0];
  $useproxy = $_[1];

  my $queryURL = 'https://' . $endpoint . '/siebatchd/v1/validate';

  my $myapikeyval = getkeyfromlocalfile;
  my $queryParams = { 'apikey' => $myapikeyval };
  my $json = encode_json $queryParams;
  my $response = make_query($queryURL, $useproxy, $json, '-999');
  unbless($response);
  my $decoded_response = decode_json $response;
  my $status = $decoded_response->{'_status'};
  return($status);
  }

# ---------------------------------------------------------------------------------
# left pad a value with spaces
# https://www.perlmonks.org/?node_id=7462
sub pad
  {
        my ($string, $length)=@_;
        my $padlength=$length-length($string);
## no critic
        return " "x$padlength.$string;
## use critic
  }

# ---------------------------------------------------------------------------------
sub format_and_printout_the_chan_list
  {
  my $chan_list = $_[0];
  # my %json_decoded_stuff;
  my %new_hash;

  my $json_decoded_stuff = decode_json $chan_list;
  foreach my $key (keys %{$json_decoded_stuff}) {

    my $actual_val=%$json_decoded_stuff{$key}->{'description'};

    # left pad the channel number to a width of 3
    my $key2 = $key;
    $key2 =~ s/ch//s;
    my $keystring = pad($key2,3);
    $new_hash{$keystring} = $actual_val;
  }

  foreach my $keystring (sort keys %new_hash) {
    printf("%s  %s\n", $keystring, $new_hash{$keystring});
  }
  return;
  }

# ---------------------------------------------------------------------------------
sub list_channels
  {

  my $endpoint = $_[0];
  my $useproxy = $_[1];

  my $myapikeyval = getkeyfromlocalfile;
  my $queryURL = 'https://' . $endpoint . '/siebatchd/v1/validate';
  my $queryParams = to_json{ 'apikey' => $myapikeyval };
  my $response = make_query($queryURL, $useproxy, $queryParams, '-999');

  unbless($response);
  my $decoded_response = decode_json $response;
  my $extract_bit = $decoded_response->{'profile'}->{'siebatch'};
  my $json_query_object = to_json($extract_bit);
  format_and_printout_the_chan_list($json_query_object);
  return;
  }

# ------------------------------------------------------------------------------
# https://www.oreilly.com/library/view/perl-cookbook/1565922433/ch02s18.html
#
sub add_commas
  {
  my $num_string = $_[0];
  my $text = reverse $num_string;
  $text =~ s/(\d\d\d)(?=\d)(?!\d*\.)/$1,/gsmx;
  return scalar reverse $text;
  }

# ------------------------------------------------------------------------------
#
sub format_and_printout_chan_time_limits
  {
  my $chan = $_[0];
  my $earliest_time_string = $_[1];
  my $latest_time_string = $_[2];
  my $volume = $_[3];

  my $chan_with_prefix = 'ch' . $chan;  # needed to correspond to the API :-(

  # take the channel status parameters and print them out in a little report
  $chan = sprintf '%4s', $chan;
  if (int($volume) >= 4) {
    $volume = add_commas($volume);
  }

  # could add a header, but it's pretty self-obvious, right?
  # printf('chan  earliest datetime    latest datetime       octets\n');
  printf("%s  \"%s\"  \"%s\"  %s\n", $chan, $earliest_time_string,
    $latest_time_string, $volume);
  return;
  }

# ------------------------------------------------------------------------------
sub show_intervals
  {
  my $endpoint = $_[0];
  my $useproxy = $_[1];
  my $chan_to_check = $_[2];

  # with square brackets
  my $newchan_to_check = '[' . $chan_to_check . ']';
  # no brackets, but with ch literal prefix
  my $chan_with_prefix = 'ch' . $chan_to_check;

  my $myapikeyval = getkeyfromlocalfile;
  my $params  = encode_json{ 'apikey'   => $myapikeyval,
                             'channels' => $newchan_to_check };

  # fix up a bit of awkwardness
  # {"channels":"[212]","apikey":"blah"}  needs to become
  # {"channels":[212],"apikey":"blah"}    (e.g., no quotes around [chan])
  $params =~ s/"\[/[/s;
  $params =~ s/]"/]/s;
  my $url = 'https://' . $endpoint . '/siebatchd/v1/siebatch/chdetails';
  my $response = make_query($url, $useproxy, $params, '-999');
  my $decoded_results=from_json($response);
  unbless($decoded_results);

  my $earliest_time_string =
    $decoded_results->{'channels'}{$chan_with_prefix}{'earliest'};
  my $latest_time_string =
    $decoded_results->{'channels'}{$chan_with_prefix}{'latest'};
  my $size_string =
    $decoded_results->{'channels'}{$chan_with_prefix}{'size'};

  return ($earliest_time_string, $latest_time_string, $size_string);
  }

# ------------------------------------------------------------------------------
sub fixup_ending_datetime_in_the_future
  {
  # Stop datetime out of range. Replacing with current GMT time.

  my $epochseconds = parsedate(gmtime());
  my $enddatetime = strftime('%Y-%m-%d %H:%M:%S', gmtime());
  $enddatetime =~ s/..$/00/s;
  return($enddatetime);
  }

# -------------------------------------------------------------------------------
sub check_channel
  {
  # make sure that the channel is available and the dates are in-range

  my $endpoint = $_[0];
  my $useproxy = $_[1];
  my $chanflagstring = $_[2];
  my $startdatetime = $_[3];
  my $enddatetime = $_[4];

  # get the available datetime range for this channel
  (my $earliest_time_string, my $latest_time_string, my $channel_octers) =
     show_intervals($endpoint, $useproxy, $chanflagstring);

  # convert the requested and available start datetimes into Un*x seconds
  my $requested_start_seconds = parsedate($startdatetime);
  my $earliest_date_seconds =   parsedate($earliest_time_string);

  # convert the requested and available end datetimes into Un*x seconds
  my $requested_stop_seconds = parsedate($enddatetime);
  my $latest_date_seconds =    parsedate($latest_time_string);

  # start datetime must be earlier than stop date time
  if (($requested_stop_seconds - $requested_start_seconds) < 0) {
    croak('Start datetime must be earlier than stop datetime');
  }

  # start datetime may not be earlier than earliest data available
  if (($requested_start_seconds - $earliest_date_seconds) < 0) {
    croak('Start datetime out of range. Must be no earlier than ' . \
      $earliest_time_string);
  }
  # end datetime may not be in the future
  if (($requested_stop_seconds - $latest_date_seconds) > 0) {
    $enddatetime = fixup_ending_datetime_in_the_future();
  }

  # handle return of the (potentially updated) datetimes here
  my @timearray;
  $timearray[0] = $startdatetime;
  $timearray[1] = $enddatetime;
  return (@timearray);
  }

# ----------------------------------------------------------------------------
sub validate_input_time_date_format
  {
  # parameter is datetime to format check
  # if invalid, abort run
  # if valid, return the validated (but unchanged) datetime (could skip
  # doing this for now, but at some point we might decide to fix up bad
  # string formatting as a convenience to the user, so...)

  my $mydatetime = $_[0];

  # check the format with a regex
  if (($mydatetime =~ /\A\d{4}-\d{2}-\d{2}\ \d{2}:\d{2}:\d{2}\Z/smx)){
    # good starting time format
  } else {
    print("bad starting time format -- must be \"YYYY-MM-DD HH:MM:SS\"\n");
    exit(1);
  }

  return ($mydatetime);
  }

# ----------------------------------------------------------------------------
sub zero_unused_seconds
  {
  # since SIE-Batch API does not care about seconds, we set them to zero

  my $mydatetime = $_[0];
  $mydatetime =~ s/..$/00/s;
  return ($mydatetime);
  }

# ----------------------------------------------------------------------------
sub convert_relative_times_to_real_datetimes
  {
  my $minutesback = $ARGV[2];
  my $format = '%Y-%m-%d %H:%M:%S';

  # in relative format, the initial "ending time" is actually the minutes
  # worth of data we want to retrieve
  # the "real" ending datetime will be created from the current GMT time
  # we will be doing math on the epoch seconds

  my $endingtime = gmtime();
  my $epochseconds = parsedate($endingtime);

  # now compute the formatted ending date time in standard YYYY-MM-DD HH:MM:SS
  my $enddatetime = strftime $format, localtime $epochseconds;
  # find just the seconds from that string
  my $extraseconds = strftime '%S', localtime $epochseconds;
  # subtract the seconds from the full datetime to end up with 00 seconds
  my $endingtime_seconds = int($epochseconds) - int($extraseconds);

  # let's now work on the starting time
  # we compute the "real" starting datetime by offsetting backwards
  # our to-be-modified datetime is in epoch seconds, so convert min to seconds
  my $mysecondsback = int($minutesback) * 60;
  my $startseconds = $endingtime_seconds - int($mysecondsback);
  my $startdatetime = strftime $format, localtime $startseconds;

  $enddatetime = strftime $format, localtime $endingtime_seconds;

  my @timearray;
  $timearray[0] = $startdatetime;
  $timearray[1] = $enddatetime;

  return(@timearray);
  }

# ------------------------------------------------------------------------------
sub fix_times
  {
  # arguments come in from the command line so we don't pass them in

  # standardize the starting and ending times from the command line
  (my $chanflagstring, my $startdatetime, my $enddatetime) = @ARGV;

  # if relative times, replace the ending time with the current GMT time
  # set the starting time back by the specified number of minutes
  my @timearray;
  if ($startdatetime eq 'now') {
    @timearray = convert_relative_times_to_real_datetimes($enddatetime);
  } else {
    # we have real timedate stamps for starting and ending datetimes

    # process the starting datetime value...
    # correctly written datetime value?
    # also zero the seconds if present (SIE-Batch API doesn't use them)
    validate_input_time_date_format($startdatetime);
    $startdatetime = zero_unused_seconds($startdatetime);

    # repeat for the ending datetime value...
    validate_input_time_date_format($enddatetime);
    $enddatetime = zero_unused_seconds($enddatetime);

    $timearray[0] = $startdatetime;
    $timearray[1] = $enddatetime;
  }
  return (@timearray);
  }

# ------------------------------------------------------------------------------
sub check_if_chan_is_an_nmsg_chan
  {
  my $chanflagstring = $_[0];

  my %keys = (204 => 1,
              206 => 1,
              207 => 1,
              208 => 1,
            221 => 1,);

  my $filetype = undef;
  # my $keys;
  if ($keys{$chanflagstring}) {
    $filetype = '.nmsg';
  } else {
    $filetype = '.jsonl';
  }

  return($filetype);
  }

# ------------------------------------------------------------------------------
sub build_filename
  {
  # construct the filename and return it
  my $string1 = $_[1];
  my $string2 = $_[2];
  my $chanflagstring = $ARGV[0];

  $string1 =~ s/ /@/s;
  $string2 =~ s/ /@/s;

  my $filetype = check_if_chan_is_an_nmsg_chan($chanflagstring);
  my $outputfilename = 'sie-ch' . $chanflagstring . '-{' . $string1 .
    '}-{' . $string2 . '}' . $filetype;

  return($outputfilename);
  }

# -----------------------------------------------------------------------------
sub print_usage_info
  {
  print <<'FOO';
Usage:

  sie_get_pl channel "now" minutesBack
  Example: sie_get_pl 212 now 15

    OR

  sie_get_pl channel "startdatetime" "enddatetime"
  Example: sie_get_pl 212 "2020-01-07 00:13:00" "2020-01-07 00:28:00"

Convenience functions:

  Check SIE-Batch API key:                      sie_get_pl checkkey
  Get a listing of channels:                    sie_get_pl channels
  Get datetime range and volume for a channel:  sie_get_pl 212

Notes:

  Datetimes are UTC and must be quoted. (Current UTC datetime: $ date -u )
  Zero pad any single digit months, days, hours, minutes or seconds.
  Seconds must be entered as part of the UTC datetimes (but are ignored)
  Ending datetime in the future? It will be clamped to current datetime.

  minutesBack must be >= 1 and a whole number
FOO
  exit(0);
  }

# ------------------------------------------------------------------------------
sub one_real_arg
  {
  # because users can ask for channel info by specifying a channel
  # number, we need to know the list of defined channel numbers

  my $first_arg = $_[2];
  my $chanflagstring = $ARGV[0];

  my %defined_channels = (24 =>  'true',
                          25 =>  'true',
                          27 =>  'true',
                          42 =>  'true',
                          80 =>  'true',
                          114 => 'true',
                          115 => 'true',
                          204 => 'true',
                          206 => 'true',
                          207 => 'true',
                          208 => 'true',
                          211 => 'true',
                          212 => 'true',
                          213 => 'true',
                          214 => 'true',
                          221 => 'true',
                         );

    my $first_arg_looks_numeric = ($first_arg =~ /^\d+$/s);
    my $status;

    if ($first_arg eq 'channels') {
        # list channels for the user
        list_channels ($endpoint, $useproxy);
        exit(0);
    } elsif ($first_arg eq 'checkkey') {
        # check the user's key for validity
        $status = validateapikeyonline($endpoint, $useproxy);
        print('API key status is ' . $status . "\n");
        exit(0);
    } elsif ($defined_channels{$first_arg}) {
        # list details about the specified channel
        (my $earliest, my $latest, my $datasize) =
          show_intervals($endpoint, $useproxy, $chanflagstring);
        format_and_printout_chan_time_limits($chanflagstring, $earliest,
          $latest, $datasize);
        exit(0)
    } elsif ((not($defined_channels{$chanflagstring})) &&
         ($first_arg_looks_numeric)) {
        ### the requested channel is not one we offer, so...
        print("Channel not offered via this script\n");
        exit(0);
    } else {
        print_usage_info();
        exit(0);
    }
}

# ------------------------------------------------------------------------------
sub three_real_args
```

```
    {
    my $first_arg = $ARGV[0];
    my $startdatetime = $ARGV[1];
    my $enddatetime = $ARGV[2];

    my $myapikey = getkeyfromlocalfile;

    ($startdatetime, $enddatetime) = fix_times();

    ($startdatetime, $enddatetime) = check_channel($endpoint, $useproxy,
        $first_arg, $startdatetime, $enddatetime);

    my $outputfilename = build_filename($first_arg, $startdatetime,
        $enddatetime);

    my %params = ('apikey' => getkeyfromlocalfile,
       'channel' => int($first_arg),
       'start_time' => $startdatetime,
       'end_time' => $enddatetime,);

    # the backslash in the following is critical; w/o it, $params2 becomes 4
    my $params2 = encode_json \%params;

    my $queryURL = 'https://' . $endpoint . '/siebatchd/v1/siebatch/chfetch';

    my $response = make_query($queryURL, $useproxy, $params2,
        $outputfilename);

    # write the SIE-Batch data out to our file
    open my $fh, '>:raw', $outputfilename or croak();
    print $fh $response;
    close($fh) or croak ("Could not successfully close output file $fh\n");
    exit(0);
}

# ============================================================================
# main

my $first_arg;
my $status;

if (@ARGV) {
  $first_arg = $ARGV[0];
}

my $command_line_arg_count = $#ARGV + 1;

if ($command_line_arg_count == 1)
{
  one_real_arg($endpoint, $useproxy, $first_arg);
  exit(0);
}
elsif ($command_line_arg_count == 3)
{
  three_real_args($endpoint, $useproxy, $first_arg);
  exit(0);
}
elsif (($command_line_arg_count <= 0) ||
       ($command_line_arg_count == 2) ||
       ($command_line_arg_count >= 4))
{
      print_usage_info();
      exit(0);
}
```

```
$ cat .perlcritic
severity = brutal

[TooMuchCode::ProhibitUnusedImport]
[Variables::ProhibitUnusedVariables]

[Subroutines::RequireArgUnpacking]
      allow_subscripts = 2

[Variables::ProhibitReusedNames]
        allow = $useproxy $endpoint

[RegularExpressions::RequireExtendedFormatting]
        minimum_regex_length_to_complain_about = 10

[ControlStructures::ProhibitCascadingIfElse]
        max_elsif = 6

[InputOutput::RequireBriefOpen]
        lines = 5

[InputOutput::RequireCheckedSyscalls]
functions = :builtins
exclude_functions = print

[-CodeLayout::ProhibitParensWithBuiltins]
[-CodeLayout::RequireTidyCode]
[-InputOutput::RequireBracedFileHandleWithPrint]
[-Miscellanea::ProhibitUnrestrictedNoCritic]
[-Modules::RequireVersionVar]
[-NamingConventions::Capitalization]
[-References::ProhibitDoubleSigils]
[-RegularExpressions::ProhibitEscapedMetacharacters]
[-RegularExpressions::RequireExtendedFormatting]
[-RegularExpressions::RequireLineBoundaryMatching]
[-Subroutines::ProhibitAmpersandSigils]
[-TooMuchCode::ProhibitDuplicateLiteral]
[-ValuesAndExpressions::ProhibitCommaSeparatedStatements]
[-ValuesAndExpressions::ProhibitMagicNumbers]
[-ValuesAndExpressions::ProhibitNoisyQuotes]
```

Appendix VII. sie_get_c client source code

```
$ cat Makefile
EXEC = sie_get_c
PREFIX = /usr/local

CC = clang
CFLAGS = -Wall -Werror -std=c11 -O3
CPPFLAGS = -I /usr/local/include
LDFLAGS = -L /usr/local/lib -Wall
LDLIBS = -lcurl

.SUFFIXES:
.SUFFIXES: .c .o

objects = \
      build_filename.o \
      check_channel.o \
      check_intervals.o \
      convert_relative_times_to_real_datetimes.o \
      getkeyfromlocalfile.o \
      fix_times.o \
      isNumeric.o \
        list_channels.o \
      make_query.o \
      one_real_arg.o \
      parson.o \
        printout_intervals.o \
      print_usage_info.o \
      replString.o \
      search_a_list.o \
      sie_get_c.o \
      string_fmt_time_to_seconds.o \
      subString.o \
      three_real_args.o \
        validateapikeyonline.o \
        validate_input_time_date_format.o

all: $(objects)
      $(CC) -o $(EXEC) $(objects) $(LDFLAGS) $(LDLIBS)

sie_get.o :
getAPIkey.o :

.PHONY: install
install:
      mkdir -p $(PREFIX)/bin
      cp $(EXEC) $(PREFIX)/bin/.
      chmod a+rx $(PREFIX)/bin/$(EXEC)
#     mkdir -p $(MANPREFIX)/man1
#     cp $(EXEC).1 $(MANPREFIX)/man1/.

.PHONY: clean
clean:
      @rm -f $(EXEC) $(objects)
```

```
$ cat build_filename.h
#define SMALL    20
#define LARGE    200

/* prototypes */

int search_a_list(const char *, const char **const);
void build_filename(void);
char *replString(const char *, const char *, const char *);

/* global variables */

extern char firstarg[];
extern char secarg[];
extern char thirdarg[];

/* created here, no extern */
char outputfilename[LARGE];


$ cat build_filename.c
#include <stdio.h>
#include <string.h>
#include "build_filename.h"

void build_filename(void)
{
   /* need these for the file name we're building */
   char string1[SMALL];
   char string2[SMALL];

   /* these are nmsgchannel format channels */
   /*@null@*/
   const char *nmsgchannels[] = { "204", "206", "207", "208",
                                  "221", NULL };

   /* NOTE: last channel in the above list MUST be NULL as shown! */

   /* replace spaces with @'s in the filenames we make */
   snprintf(string1, sizeof string1, "%s", replString(secarg, " ", "@"));
   snprintf(string2, sizeof string2, "%s", replString(thirdarg, " ", "@"));

   /* assemble the filename */
   strlcpy(outputfilename, "sie-ch", sizeof(outputfilename));
   strlcat(outputfilename, firstarg, sizeof(outputfilename));
   strlcat(outputfilename, "-{", sizeof(outputfilename));
   strlcat(outputfilename, string1, sizeof(outputfilename));
   strlcat(outputfilename, "}-{", sizeof(outputfilename));
   strlcat(outputfilename, string2, sizeof(outputfilename));
   strlcat(outputfilename, "}", sizeof(outputfilename));

   if (search_a_list(firstarg, nmsgchannels) == -1)
   {
      strlcat(outputfilename, ".jsonl", sizeof(outputfilename));
   }
   else
   {
      strlcat(outputfilename, ".nmsg", sizeof(outputfilename));
   }
}
```

```
$ cat check_channel.h
#define SMALL    20
#define LARGE    255

/* prototypes */
void check_channel(void);
void check_intervals(void);
long string_fmt_time_to_seconds(const char *);

/* global variables */
extern char firstarg[];
extern char secarg[];
extern char thirdarg[];

extern char earliestdate_avail[SMALL];
extern char latestdate_avail[SMALL];
extern char volume[SMALL];


$ cat check_channel.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "check_channel.h"

void check_channel(void)
{
   long requested_start_seconds;
   long requested_stop_seconds;

   long earliest_date_seconds;
   long latest_date_seconds;

   /* make sure that the channel is available and the dates are in-range */

   /* check the available datetime range for this channel */
   (void)check_intervals();

   /* convert the requested and available datetimes into Unix seconds */
   /* (easier to compare dates as Unix epoch seconds */
   requested_start_seconds = string_fmt_time_to_seconds(secarg);
   earliest_date_seconds   = string_fmt_time_to_seconds(
      earliestdate_avail);

   requested_stop_seconds = string_fmt_time_to_seconds(thirdarg);

   latest_date_seconds = string_fmt_time_to_seconds(
      latestdate_avail);

   /* REQUIRD: requested ending time can't be in the future */
   if (requested_stop_seconds > latest_date_seconds)
   {
      printf("Stop time can't be in the future!\n");
      exit(EXIT_FAILURE);
   }

   /* REQUIRED: start datetime must be earlier than stop datetime */
   if ((requested_stop_seconds - requested_start_seconds) < 0)
   {
      printf("Start datetime must be earlier than stop datetime\n");
      exit(EXIT_FAILURE);
   }

   /* REQUIRED: start datetime must be after the earliest data available */
   if ((requested_start_seconds - earliest_date_seconds) < 0)
   {
      printf("Start datetime out of range. Must be no earlier than\n");
      printf("%s\n", earliestdate_avail);
      exit(EXIT_FAILURE);
   }
} /* check_channel */
```

```
$ cat check_intervals.h
#define SMALL    20
#define LARGE    200

/* prototypes */
char *getkeyfromlocalfile(void);
char *make_query(char[LARGE], char[LARGE]);
char *replString(const char *, const char *, const char *);
void check_intervals(void);

/* global variables */
extern char firstarg[];
extern char secarg[];
extern char thirdarg[];
extern char endpoint[];
extern char outputfilename[];

/* defined here, so no extern */
char earliestdate_avail[SMALL];
char latestdate_avail[SMALL];
char volume[SMALL];


$ cat check_intervals.c
#include <string.h>
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include "parson.h"   /* https://github.com/kgabis/parson/ */
#include "check_intervals.h"

void check_intervals()
{
   char  chan_to_check[8];
   char  earliest[SMALL];
   char  latest[SMALL];
   char *myapikeyval;
   char  other_format[8];
   char  params[LARGE];
   char *response;
   char  url[LARGE];

   JSON_Object *my_obj_1;
   JSON_Object *my_obj_2;

   JSON_Value *my_jv;

   myapikeyval = getkeyfromlocalfile();

   /* build our parameters for the query */
   /* {"channels":[212],"apikey":"blah"}  (note: no quotes around [chan]) */
   strlcpy(chan_to_check, "[", sizeof(chan_to_check));
   strlcat(chan_to_check, firstarg, sizeof(chan_to_check));
   strlcat(chan_to_check, "]", sizeof(chan_to_check));

   strlcpy(params, "{\"apikey\":\"", sizeof(params));
   strlcat(params, myapikeyval, sizeof(params));
   strlcat(params, "\",\"channels\":", sizeof(params));
   strlcat(params, chan_to_check, sizeof(params));
   strlcat(params, "}", sizeof(params));
   /* build the endpoint we're going to visit */
   strlcpy(url, "https://", sizeof(params));
   strlcat(url, endpoint, sizeof(params));
   strlcat(url, "/siebatchd/v1/siebatch/chdetails", sizeof(params));

   /* actually do the query */
   /* -999 is a magic value meaning no file output */
   strlcpy(outputfilename, "-999", 5);
   response = make_query(url, params);

   /* Now process the JSON results */

   /* get the json string, parse it, and extract the channels and descrp */
   my_jv = json_parse_string(response);

   /* it's all under the channels leg of the JSON tree */
   my_obj_1 = json_object_get_object(json_object(my_jv), "channels");

   /* to pick a chan,  we need the number w/o brackets but with ch */
   strlcpy(other_format, "ch", sizeof(other_format));
   strlcat(other_format, firstarg, sizeof(other_format));

   /* we're now down a specific channel number in the JSON tree */
   my_obj_2 = json_object_get_object(my_obj_1, other_format);

   /* extract the two values we actually care about */
   /* NOT printing anything here , just load it into the struct */
   strlcpy(earliestdate_avail,
           json_object_get_string(my_obj_2, "earliest"),
           sizeof(earliest));
   strlcpy(latestdate_avail,
           json_object_get_string(my_obj_2, "latest"),
           sizeof(latest));
} /* check_intervals */
```

```
$ cat convert_relative_times_to_real_datetimes.h
#define SMALL    20

/* prototypes */
char *subString(const char *, int, int, char *);

/* global variables */
extern char secarg[];
extern char thirdarg[];
char        saved_minutes_back[SMALL];

$ cat convert_relative_times_to_real_datetimes.c
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>
#include "convert_relative_times_to_real_datetimes.h"

void convert_relative_times_to_real_datetimes()
{
   /* local variables */

   long      startseconds;
   long      endingtime_seconds;
   long      extraseconds;
   struct tm gmendingtime;
   long      mysecondsback;

   char *myformat;
   char  lasttwo[3];

   myformat = "%Y-%m-%d %H:%M:%S";

   /* time_t is raw epoch seconds */
   time_t epochseconds;

   /* save a copy of the number of minutes worth of data we want to go */
   /* back in relative time */
   strlcpy(saved_minutes_back, thirdarg, sizeof(saved_minutes_back));

   /* when retrieving data, there are two options: */
   /* */
   /* one option is relative times; if we get one, make it a real time */
   /* */
   /* in relative format, the initial "ending time" is actually the minutes */
   /* worth of data we want to retrieve */
   /* */
   /* the "real" ending datetime will be created from the current GMT time */
   /* we will be doing math on the epoch seconds */

   epochseconds = time(NULL);
   gmendingtime = *gmtime(&epochseconds);

   /* now compute the formatted ending date time in standard  */
   /* YYYY-MM-DD HH:MM:SS */
   strftime(thirdarg, SMALL, myformat, &gmendingtime);

   /* find just the seconds from that string (we need the starting point) */
   lasttwo[0] = 0;  /* make sure the string is NULL terminated */
   subString(thirdarg, (strlen(thirdarg) - 2), 2, lasttwo);
   extraseconds = atol(lasttwo);
   /* subtract the seconds from the full datetime to end up with 00 seconds */
   endingtime_seconds = timegm(&gmendingtime) - extraseconds;

   /* let's now work on the starting time */
   /* we compute the "real" starting datetime by offsetting backwards */
   /* our to-be-modified datetime is in epoch seconds, so convert min */
   /* to seconds */

   mysecondsback = atol(saved_minutes_back) * 60;
   startseconds  = endingtime_seconds - mysecondsback;
   struct tm ts = *gmtime(&startseconds);
   strftime(secarg, SMALL, myformat, &ts);
   strftime(thirdarg, SMALL, myformat, gmtime(&endingtime_seconds));
} /* convert_relative_times_to_real_datetimes */
```

```
$ cat fix_times.h
#define SMALL    20
#define BIG      200

/* prototypes */
void convert_relative_times_to_real_datetimes(void);
void fix_times(void);
void validate_input_time_date_format(char[SMALL]);

/* global variables */
extern char secarg[];
extern char thirdarg[];


$ cat fix_times.c
#include <stdio.h>
#include <strings.h>
#include "fix_times.h"

void fix_times(void)
{
   /* handles calling the rest of the routines to fix up times */
   /* arguments come in from the command line as global variables */
   /* so we don't pass them in */

   /* if relative times, replace the ending time with the current GMT time */
   /* set the starting time back by the specified number of minutes */

   if (strncmp(secarg, "now", 4) == 0)
   {
      (void)convert_relative_times_to_real_datetimes();
   }
   else
   {
      /* we have real timedate stamps for starting and ending datetimes */
      /* process the starting datetime value... */
      /* correctly written datetime value? */

      (void)validate_input_time_date_format(secarg);

      /* repeat for the ending datetime value... */
      (void)validate_input_time_date_format(thirdarg);
   }
}
```

```
$ cat getkeyfromlocalfile.h
/* global variables */
char  filepath[200];
char *myapikey;
char *homedir;

FILE *f;

size_t  len = 0;
ssize_t linelength;


$ cat getkeyfromlocalfile.c
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "getkeyfromlocalfile.h"

char *getkeyfromlocalfile()
{
   /* Retrieves the SIE-Batch API key */

   if ((homedir = getenv("HOME")) == NULL)
   {
      homedir = getpwuid(getuid())->pw_dir;
   }

   strlcpy(filepath, homedir, sizeof(filepath));
   strlcat(filepath, "/.sie-get-key.txt", sizeof(filepath));

   if (access(filepath, F_OK) == -1)
   {
      printf("\nERROR:\n\n  No SIE-Batch API keyfile at ~/.sie-get-key.txt\n");
      exit(1);
   }

   f          = fopen(filepath, "r");
   linelength = getline(&myapikey, &len, f);

   if (myapikey[linelength - 1] == '\n')
   {
      myapikey[linelength - 1] = 0;
   }

   fclose(f);

   return(myapikey);
}
```

```
$ cat isNumeric.c
#include <ctype.h>
#include <stdlib.h>

/* see https://rosettacode.org/wiki/Determine_if_a_string_is_numeric#C */

int isNumeric(const char *s)
{
   if (s == NULL || *s == '\0' || isspace(*s))
   {
      return(0);
   }

   char *p;
   strtod(s, &p);
   return(*p == '\0');
}
```

```
$ cat list_channels.h
#define LARGE    200

/* prototypes */
char *make_query(char[LARGE], char[LARGE]);
void list_channels();
char *getkeyfromlocalfile();
char *replString(const char *, const char *, const char *);

/* global variables */
extern char endpoint[];
extern char outputfilename[];


$ cat list_channels.c
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "parson.h"
#include "list_channels.h"

void list_channels()
{
   /* no real args come in because we always just want to list all channels */

   char queryURL[LARGE];
   char params[LARGE];

   char *returned_content;
   char *myapikeyval;

   myapikeyval = getkeyfromlocalfile();

   strlcpy(params, "{\"apikey\":\"", sizeof(params));
   strlcat(params, myapikeyval, sizeof(params));
   strlcat(params, "\"}", sizeof(params));

   strlcpy(queryURL, "https://", sizeof(queryURL));
   strlcat(queryURL, endpoint, sizeof(queryURL));
   strlcat(queryURL, "/siebatchd/v1/validate", sizeof(queryURL));

   strlcpy(outputfilename, "-999", 5);
   returned_content = make_query(queryURL, params);

   /* json processing */

   struct keys { char key[7]; };
   struct vals { char val[70]; };

   struct keys mykeys[24];
   struct vals myjvals[24];

   JSON_Value *my_jv;

   JSON_Object *my_obj_1;
   JSON_Object *my_obj_2;

   int j_i;
   int j_j;

   char tempkey[7];
   char tempval[70];

   /* parse the json string, extract and print the channels and descrips */
   my_jv = json_parse_string(returned_content);

   /* the data we need is under profile.siebatch */
   /* NOTE: do NOT get distracted by the profile.channels object! */
   my_obj_1 = json_object_dotget_object(json_object(my_jv),
                                        "profile.siebatch");

   /* pull a list of channel numbers */
   for (j_i = 0; j_i < json_object_get_count(my_obj_1); j_i++)
   {
      /* we know how many objects now, but what are the object names? */
      sprintf(mykeys[j_i].key, "%s", json_object_get_name(my_obj_1, j_i));

      /* now that we've got the names, step through those objects */
      my_obj_2 = json_object_get_object(my_obj_1, mykeys[j_i].key);

      /* the channel description's the last thing we need */
      sprintf(myjvals[j_i].val, "%s",
              json_object_get_string(my_obj_2, "description"));
   }

   /* the channel names are prefixed with ch, remove that for sorting */
   for (j_i = 0; (j_i < json_object_get_count(my_obj_1)); j_i++)
   {
      strlcpy(tempkey, replString(mykeys[j_i].key, "ch", ""),
              sizeof(tempkey));
      /* now restore the ch-less keys to their original location */
      strlcpy(mykeys[j_i].key, tempkey, sizeof(mykeys[j_i]));
   }

   /* crumby little bubblesort (works fine for a tiny hash like this one) */
   for (j_j = 0; (j_j < json_object_get_count(my_obj_1) - 1); j_j++)
   {
      for (j_i = 0; (j_i < (json_object_get_count(my_obj_1) - j_j - 1)); j_i++)
      {
         if (atoi(mykeys[j_i].key) > atoi(mykeys[j_i + 1].key))
         {
            strlcpy(tempkey, mykeys[j_i].key, sizeof(tempkey));
            strlcpy(tempval, myjvals[j_i].val, sizeof(tempval));

            strlcpy(mykeys[j_i].key, mykeys[j_i + 1].key, sizeof(mykeys[j_i].key));
            strlcpy(myjvals[j_i].val, myjvals[j_i + 1].val, sizeof(myjvals[j_i].val));

            strlcpy(mykeys[j_i + 1].key, tempkey, sizeof(mykeys[j_i].key));
            strlcpy(myjvals[j_i + 1].val, tempval, sizeof(myjvals[j_i].val));
         }
      }
   }

   /* print the actual channel listing */
   for (j_i = 0; (j_i < json_object_get_count(my_obj_1)); j_i++)
   {
      printf("ch%s  %s\n", mykeys[j_i].key, myjvals[j_i].val);
   }
   exit(EXIT_SUCCESS);
} /* list_channels */
```

```
$ cat make_query.h
#define LARGE    200

/* memory handling */
/* see https://curl.haxx.se/libcurl/c/getinmemory.html */

struct MemoryStruct
{
   char * memory;
   size_t size;
};

/* function declarations */
char *make_query(char[LARGE], char[LARGE]);

/* global variables */
extern char useproxy[];
extern char outputfilename[LARGE];


$ cat make_query.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>
#include <curl/curl.h>
#include <fcntl.h>
#include <unistd.h>
#include "make_query.h"

/* memory handling */
/* see https://curl.haxx.se/libcurl/c/getinmemory.html */

static size_t WriteMemoryCallback(void *contents, size_t size,
                                  size_t nmemb, void *userp)
{
   size_t realsize          = size * nmemb;
   struct MemoryStruct *mem = (struct MemoryStruct *)userp;

   char *ptr = realloc(mem->memory, mem->size + realsize + 1);

   if (ptr == NULL)
   {
      /* out of memory! */
      printf("not enough memory (realloc returned NULL)\n");
      return(0);
   }

   mem->memory = ptr;
   memcpy(&(mem->memory[mem->size]), contents, realsize);
   mem->size += realsize;
   mem->memory[mem->size] = 0;

   return(realsize);
}

char *make_query(char url[LARGE], char params[LARGE])
{
   /* variables */
```

```
CURL *   mycurl;
CURLcode res;
FILE *   f = NULL;

long http_code = 0;

char errbuf[CURL_ERROR_SIZE];
struct curl_slist * list = NULL;
struct MemoryStruct chunk;

/* the libcurl example code requires use of this malloc/realloc */
chunk.memory = malloc(1);  /* grown as needed by the realloc above */
chunk.size   = 0;          /* no data at this point */

double tries   = 0;
double elapsed = 0;
bool   done    = false;

if (strncmp(outputfilename, "-999", 5) != 0)
{
   f = fopen(outputfilename, "wb");
}

mycurl = curl_easy_init();

/* if needed for debugging */
/*  curl_easy_setopt(mycurl, CURLOPT_VERBOSE, 1L); */

/* the end point we're gong to visit */
curl_easy_setopt(mycurl, CURLOPT_URL, url);

/* force IPv4 only for now */
curl_easy_setopt(mycurl, CURLOPT_IPRESOLVE, CURL_IPRESOLVE_V4);

/* we're doing a POST */
curl_easy_setopt(mycurl, CURLOPT_POST, true);

/* pass our arguments to the query */
curl_easy_setopt(mycurl, CURLOPT_POSTFIELDS, params);

/* redirect if need be */
curl_easy_setopt(mycurl, CURLOPT_FOLLOWLOCATION, 1L);

/* we pass an HTTP header to ensure it's a JSON world */
list = curl_slist_append(list, "Content-Type: application/json");
curl_easy_setopt(mycurl, CURLOPT_HTTPHEADER, list);

/* handle timeouts */
curl_easy_setopt(mycurl, CURLOPT_CONNECTTIMEOUT, 30L);
curl_easy_setopt(mycurl, CURLOPT_TIMEOUT, 3000L);
curl_easy_setopt(mycurl, CURLOPT_USERAGENT, "sie_get_c/1.0");

/* connect via a proxy or connect directly with SSL/TLS? */
if (strcmp(useproxy, "yes") == 0)
{
   /* printf("trying to use the proxy...\n"); */
   curl_easy_setopt(mycurl, CURLOPT_PROXY, "127.0.0.1");
   curl_easy_setopt(mycurl, CURLOPT_PROXYPORT, 1080);
   curl_easy_setopt(mycurl, CURLOPT_PROXYTYPE, CURLPROXY_SOCKS5);
}
else
```

```
{
   /* printf("trying to connect directly...\n"); */
   curl_easy_setopt(mycurl, CURLOPT_SSL_VERIFYPEER, 1);
   curl_easy_setopt(mycurl, CURLOPT_SSL_VERIFYHOST, 1);
}

/* write the actual data files directly to the outputfile */
/* or for (apikey check, channel listing, etc.) we're just */
/* going to write to a buffer (and then return that)) */
if (strncmp(outputfilename, "-999", 5) == 0)
{
   /* printf("writing to the function, not to a file\n"); */
   /* send all data to this function  */
   curl_easy_setopt(mycurl, CURLOPT_WRITEFUNCTION,
                    WriteMemoryCallback);
   /* we pass our 'chunk' struct to the callback function */
   curl_easy_setopt(mycurl, CURLOPT_WRITEDATA,
                    (void *)&chunk);
}
else
{
   /* printf("writing directly to the file\n"); */
   curl_easy_setopt(mycurl, CURLOPT_WRITEDATA, f);
}

/* try the query three times, just in case there's an */
/* intermittent issue */
while (tries != 3 && !done)
{
   /* do the actual curl command */
   res = curl_easy_perform(mycurl);
   /* printf("after curl_easy_perform, res=%d\n",res); */

   /* was the request successful? */
   curl_easy_getinfo(mycurl, CURLINFO_RESPONSE_CODE, &http_code);
   /* printf("Response code=%ld\n", http_code); */

   if (res != CURLE_OK)
   {
      size_t len = strlen(errbuf);
      fprintf(stderr, "\nlibcurl: (%d) ", res);
      if (len)
      {
         fprintf(stderr, "%s%s", errbuf,
                 ((errbuf[len - 1] != '\n') ? "\n" : ""));
      }
      else
      {
         fprintf(stderr, "%s\n", curl_easy_strerror(res));
      }
   }

   /* how long did it take? */
   curl_easy_getinfo(mycurl, CURLINFO_TOTAL_TIME, &elapsed);
   /* printf("Elapsed time=%f\n", elapsed); */

   /* how big's the download? we primarily care about short downloads */
   curl_off_t cl;
   res = curl_easy_getinfo(mycurl,
                           CURLINFO_CONTENT_LENGTH_DOWNLOAD_T, &cl);
   if (!res)
```

```
      {
         /* printf("Download size: %" CURL_FORMAT_CURL_OFF_T "\n", cl); */
      }

      if (res != CURLE_OK || http_code != 200)
      {
         printf("tries = %f\n", tries);
      }
      else
      {
         done = true;
      }
   }

   curl_easy_cleanup(mycurl);
   curl_global_cleanup();

   if (strncmp(outputfilename, "-999", 5) == 0)
   {
      /* return results from the query */
      return(chunk.memory);
   }
   else
   {
      /* this means that we've written a file with our output */
      exit(0);
   }
} /* make_query */
```

```
$ cat one_real_arg.h
extern int  myargcount;
extern char firstarg[];
extern char secarg[];
extern char thirdarg[];

/* prototypes */
void list_channels(void);
void one_real_arg(void);
void print_usage_info(void);
void format_and_printout_chan_time_limits(void);
void printout_intervals(void);
char *validateapikeyonline();
int isNumeric(char *);
int search_a_list(const char *, const char **const);


$ cat one_real_arg.c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <curl/curl.h>
#include "one_real_arg.h"

void one_real_arg(void)
{
   /* constants */

   const char *defined_channels[] =
   {
      "24",  "25",  "27",  "42",
      "80",  "114", "115", "204",
      "206", "207", "208", "211",
      "212", "213", "214", "221",
      NULL
   };

   /* NOTE: last entry in above list MUST be NULL as shown! */

   /* what did the user want to do? */
   if (strncmp(firstarg, "checkkey", 8) == 0)
   {
      /* check the user's key for validity and exit */
      printf("API key status is %s\n", validateapikeyonline());
      exit(EXIT_SUCCESS);
   }
   else if (strncmp(firstarg, "channels", 8) == 0)
   {
      /* list channels for the user and exit */
      (void)list_channels();
      exit(EXIT_SUCCESS);
   }
   else
   {
      /* The only other options should be a specific numeric channel */

      /* non-numeric single argument (unknown command) */
      if (!(isNumeric(firstarg)))
      {
         printf("not a channel number nor a known command\n");
         exit(EXIT_FAILURE);
      }

      /* channel number's too big or too small */
      if ((atoi(firstarg) < 10) || (atoi(firstarg) > 255))
      {
         printf("out of range SIE-Batch API channel\n");
         exit(EXIT_FAILURE);
      }

      /* did the user ask for a channel this script doesn't know about? */
      if (search_a_list(firstarg, defined_channels) == -1)
      {
         printf("Channel not currently available from this script\n");
         exit(EXIT_FAILURE);
      }

      /* get earliest available datetime, latest available datetime */
      /* and the volume that range represents for the specified channel */

      /* retrieve details about a specific channel the user asked about */
      (void)printout_intervals();
      exit(EXIT_SUCCESS);
   }
} /* one_real_arg */
```

parson.c and parson.h:

See http://kgabis.github.com/parson/

```
$ cat print_usage_info.h
/* prototypes */
void print_usage_info(void);


$ cat print_usage_info.c
#include <stdlib.h>
#include <stdio.h>
#include "print_usage_info.h"

void print_usage_info(void)
{
   printf("Usage:\n\n");
   printf("  sie_get_c channel \"now\" minutesBack\n");
   printf("  Example: sie_get_c 212 now 15\n\n");
   printf("OR\n\n");
   printf("  sie_get_c channel \"startdatetime\" \"enddatetime\"\n");
   printf("  Example: sie_get_c 212 \"2020-01-07 00:13:00\" \"2020-01-07
00:28:00\"\n\n");
   printf("Convenience functions:\n\n");
   printf("  Check SIE-Batch API key:                      sie_get_c checkkey\n");
   printf("  Get a listing of channels:                    sie_get_c channels\n");
   printf("  Get datetime range and volume for a channel:  sie_get_c 212\n\n");
   printf("Notes:\n");
   printf("  Datetimes are UTC and must be quoted. (Current UTC datetime: $ date -u
)\n");
   printf("  Zero pad any single digit months, days, hours or minutes.\n");
   printf("  Seconds must be entered as part of the UTC datetimes (but are ignored)\n");
   printf("  Ending datetime in the future? It will be clamped to current
datetime.\n\n");
   printf("  minutesBack must be >= 1 and a whole number\n");
   exit(EXIT_SUCCESS);
}
```

```
$ cat printout_intervals.h
#define SMALL    20
#define LARGE    200

/* prototypes */
char *getkeyfromlocalfile(void);
char *make_query(char[LARGE], char[LARGE]);
char *replString(const char *, const char *, const char *);
void printout_intervals(void);

/* global variables */
extern char firstarg[];
extern char secarg[];
extern char thirdarg[];
extern char endpoint[];
extern char outputfilename[];

/* defined here, so no extern */
char volume[SMALL];


$ cat printout_intervals.c
#include <string.h>
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include "parson.h"      /* https://github.com/kgabis/parson/ */
#include "printout_intervals.h"

void printout_intervals()
{
   char  chan_to_check[8];
   char  earliest[SMALL];
   char  fmtfirstarg[8];
   char  latest[SMALL];
   char *myapikeyval;
   char  other_format[8];
   char  params[LARGE];
   char *response;
   char  url[LARGE];
   char  volume_string[SMALL];

   double bigness;

   JSON_Object *my_obj_1;
   JSON_Object *my_obj_2;

   JSON_Value *my_jv;

   myapikeyval = getkeyfromlocalfile();

   /* build our parameters for the query */
   /* {"channels":[212],"apikey":"blah"}  (note: no quotes around [chan]) */
   strlcpy(chan_to_check, "[", sizeof(chan_to_check));
   strlcat(chan_to_check, firstarg, sizeof(chan_to_check));
   strlcat(chan_to_check, "]", sizeof(chan_to_check));

   strlcpy(params, "{\"apikey\":\"", sizeof(params));
   strlcat(params, myapikeyval, sizeof(params));
   strlcat(params, "\",\"channels\":", sizeof(params));
   strlcat(params, chan_to_check, sizeof(params));
   strlcat(params, "}", sizeof(params));

   /* build the endpoint we're going to visit */
   strlcpy(url, "https://", sizeof(params));
   strlcat(url, endpoint, sizeof(params));
   strlcat(url, "/siebatchd/v1/siebatch/chdetails", sizeof(params));

   /* actually do the query */
   /* -999 is a magic value meaning no file output */
   strlcpy(outputfilename, "-999", 5);
   response = make_query(url, params);

   /* Now process the JSON results */

   /* get the json string, parse it, and extract the channels and descrp */
   my_jv = json_parse_string(response);

   /* it's all under the channels leg of the JSON tree */
   my_obj_1 = json_object_get_object(json_object(my_jv), "channels");

   /* to pick a chan,  we need the number w/o brackets but with ch */
   strlcpy(other_format, "ch", sizeof(other_format));
   strlcat(other_format, firstarg, sizeof(other_format));

   /* we're now down a specific channel number in the JSON tree */
   my_obj_2 = json_object_get_object(my_obj_1, other_format);

   /* extract the three values we actually care about */
   strlcpy(earliest, json_object_get_string(my_obj_2, "earliest"),
           sizeof(earliest));
   strlcpy(latest, json_object_get_string(my_obj_2, "latest"),
           sizeof(latest));
   /* this one's numeric, not a string */
   bigness = json_object_get_number(my_obj_2, "size");

   /* format the volume with thousands commas */
   setlocale(LC_NUMERIC, "");
   if (bigness >= 999.0)
   {
      snprintf(volume_string, SMALL, "%'d", (int)bigness);
   }

   /* could add a header, but it's pretty self-obvious, right? */
   /* printf('chan  earliest datetime    latest datetime       octets\n') */

   /* strip the square brackets from the channel number for output*/
   strlcpy(fmtfirstarg, replString(firstarg, "[", ""),
           sizeof(fmtfirstarg));
   strlcpy(fmtfirstarg, replString(fmtfirstarg, "]", ""),
           sizeof(fmtfirstarg));

   printf("%s  \"%s\"  \"%s\"  %s\n",
          fmtfirstarg, earliest, latest, volume_string);
} /* printout_intervals */
```

```
$ cat replString.c
/* https://creativeandcritical.net/str-replace-c */

#include <string.h>
#include <stdlib.h>
#include <stddef.h>

/* replace a substring with a new substring */
/* modified to use strlcpy */

#if (__STDC_VERSION__ >= 199901L)
# include <stdint.h>
#endif

char *replString(const char *, const char *, const char *);

char *replString(const char *str, const char *from, const char *to)
{
   /* Adjust each of the below values to suit your needs. */

   /* Increment positions cache size initially by this number. */
   size_t cache_sz_inc = 16;

   /* Thereafter, each time capacity needs to be increased,
    * multiply the increment by this factor. */
   const size_t cache_sz_inc_factor = 3;
   /* But never increment capacity by more than this number. */
   const size_t cache_sz_inc_max = 1048576;

   char *      pret, *ret = NULL;
   const char *pstr2, *pstr = str;
   size_t      i, count = 0;

    #if (__STDC_VERSION__ >= 199901L)
   uintptr_t *pos_cache_tmp, *pos_cache = NULL;
    #else
   ptrdiff_t *pos_cache_tmp, *pos_cache = NULL;
    #endif
   size_t cache_sz = 0;
   size_t cpylen, orglen, retlen, tolen, fromlen = strlen(from);

   /* Find all matches and cache their positions. */
   while ((pstr2 = strstr(pstr, from)) != NULL)
   {
      count++;

      /* Increase the cache size when necessary. */
      if (cache_sz < count)
      {
         cache_sz     += cache_sz_inc;
         pos_cache_tmp = realloc(pos_cache, sizeof(*pos_cache) * cache_sz);
         if (pos_cache_tmp == NULL)
         {
            goto end_repl_str;
         }
         else
         {
            pos_cache = pos_cache_tmp;
         }
         cache_sz_inc *= cache_sz_inc_factor;
         if (cache_sz_inc > cache_sz_inc_max)
         {
            cache_sz_inc = cache_sz_inc_max;
         }
      }

      pos_cache[count - 1] = pstr2 - str;
      pstr = pstr2 + fromlen;
   }

   orglen = pstr - str + strlen(pstr);

   /* Allocate memory for the post-replacement string. */
   if (count > 0)
   {
      tolen  = strlen(to);
      retlen = orglen + (tolen - fromlen) * count;
   }
   else
   {
      retlen = orglen;
   }
   ret = malloc(retlen + 1);
   if (ret == NULL)
   {
      goto end_repl_str;
   }

   if (count == 0)
   {
      /* If no matches, then just duplicate the string. */
      strlcpy(ret, str, sizeof(ret));
   }
   else
   {
      /* Otherwise, duplicate the string whilst performing
       * the replacements using the position cache. */
      pret = ret;
      memcpy(pret, str, pos_cache[0]);
      pret += pos_cache[0];
      for (i = 0; i < count; i++)
      {
         memcpy(pret, to, tolen);
         pret  += tolen;
         pstr   = str + pos_cache[i] + fromlen;
         cpylen = (i == count - 1 ? orglen : pos_cache[i + 1]) - pos_cache[i] - fromlen;
         memcpy(pret, pstr, cpylen);
         pret += cpylen;
      }
      ret[retlen] = '\0';
   }

end_repl_str:

   /* Free the cache and return the post-replacement string,
    * which will be NULL in the event of an error. */
   free(pos_cache);
   return(ret);
} /* replString */
```

```
$ cat search_a_list.c
/* https://rosettacode.org/wiki/Search_a_list#C */

#include <stdio.h>
#include <string.h>

int search_a_list(const char *needle, const char **hs)
{
   int i = 0;

   while (hs[i] != NULL)
   {
      if (strcmp(hs[i], needle) == 0)
      {
         return(i);
      }

      i++;
   }
   return(-1);
}
```

```
$ cat sie_get_c.h
#define SMALL    20

/* global variables */
char endpoint[] = "batch.sie-remote.net";
char useproxy[] = "no"; /* "yes" or "no" */

int  myargcount;
char firstarg[SMALL]; /* NULL, an actual channum, checkkey, channels */
char secarg[SMALL];   /* NULL, "2020-05-25 12:14:42" */
char thirdarg[SMALL]; /* NULL, "2020-05-25 12:14:42" */
```

```
$ cat sie_get_c.c
#include <stdlib.h>
#include <string.h>
#include "sie_get_c.h"

int main(int argc, char *argv[])
{
   /* prototypes */
   void three_real_args(void);
   void one_real_arg(void);
   void print_usage_info(void);

   /* argc=4 --> three arguments */
   /* argc=2 --> single argument */
   /* argc=1 --> no arguments */

   if (argc == 4)
   {
      /* we'll pulling actual data */

      /* load the globals */
      strlcpy(firstarg, argv[1], sizeof(firstarg));
      strlcpy(secarg, argv[2], sizeof(firstarg));
      strlcpy(thirdarg, argv[3], sizeof(firstarg));

      (void)three_real_args();
   }
   else if (argc == 2)
   {
      /* checkkey, channels or an actual channel number provided */

      /* save that info as a global */
      strlcpy(firstarg, argv[1], sizeof(firstarg));

      (void)one_real_arg();
   }
   else
   {
      /* wrong number of arguments, provide precis and bail */

      (void)print_usage_info();
      exit(EXIT_FAILURE);
   }
} /* main */
```

```
$ cat string_fmt_time_to_seconds.h
/* prototypes */
long string_fmt_time_to_seconds(char *);

/* global variables */
struct tm dt;
long      epoch_seconds;
char *    string_format_time;
```

```
$ cat string_fmt_time_to_seconds.h
/* prototypes */
long string_fmt_time_to_seconds(char *);

/* global variables */
struct tm dt;
long      epoch_seconds;
char *    string_format_time;
```

```
Joe:sie_get_c joe$ cat string_fmt_time_to_seconds.c
#include <string.h>
#include <stdio.h>
#include <time.h>
#include "string_fmt_time_to_seconds.h"

long string_fmt_time_to_seconds(char *string_format_time)
{
   /* utility function to convert a string format time to epoch seconds */

   strptime(string_format_time, "%Y-%m-%d %H:%M:%S", &dt);
   epoch_seconds = mktime(&dt);
   return(epoch_seconds);
}
```

```
$ cat subString.c
#include <string.h>
#include <stdlib.h>

/* get substring */
/* stackoverflow.com/questions/2114377/strings-in-c-how-to-get-substring */

/* prototype */
char *subString(char *, int, int, char *);

char *subString(char *input, int offset, int len, char *dest)
{
   int input_len = strlen(input);

   if (offset + len > input_len)
   {
      return(NULL);
   }

   strncpy(dest, input + offset, len);
   return(dest);
}
```

```
$ cat three_real_args.h
#define SMALL    20
#define LARGE    200

/* prototypes */
void build_filename(void);
void check_channel(void);
void fix_times(void);
char *getkeyfromlocalfile(void);
char *make_query(char[LARGE], char[LARGE]);
void three_real_args(void);

/* global variables */

extern char endpoint[];
extern char firstarg[];
extern char secarg[];
extern char thirdarg[];
extern char outputfilename[];


$ cat three_real_args.c
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <curl/curl.h>
#include "parson.h"    /* https://github.com/kgabis/parson/ */
#include "three_real_args.h"

void three_real_args(void)
{
   char  params[LARGE];
   char  queryURL[LARGE];
   char *myapikey;

   /* make sure the times are sane (right format and not in the future) */
   (void)fix_times();

   /* make sure the times are in-range for this particular channel */
   (void)check_channel();

   /* construct the output filename from the channel and dates */
   (void)build_filename();

   /* retrieve the API key we need */
   myapikey = getkeyfromlocalfile();

   /* construct the parameters we're going to pass-in */
   strlcpy(params, "{\"apikey\":\"", sizeof(params));
   strlcat(params, myapikey, sizeof(params));
   strlcat(params, "\",\"channel\":", sizeof(params));
   strlcat(params, firstarg, sizeof(params));
   strlcat(params, ",\"start_time\":\"", sizeof(params));
   strlcat(params, secarg, sizeof(params));
   strlcat(params, "\", \"end_time\":\"", sizeof(params));
   strlcat(params, thirdarg, sizeof(params));
   strlcat(params, "\"}", sizeof(params));

   strlcpy(queryURL, "https://", sizeof(queryURL));
   strlcat(queryURL, endpoint, sizeof(queryURL));
   strlcat(queryURL, "/siebatchd/v1/siebatch/chfetch", sizeof(queryURL));
   (void)make_query(queryURL, params);

   exit(EXIT_SUCCESS);
} /* three_real_args */
```

```
$ cat validate_input_time_date_format.h
/* prototypes */
void validate_input_time_date_format(char *);


$ cat validate_input_time_date_format.c
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <regex.h>
#include "validate_input_time_date_format.h"

void validate_input_time_date_format(char *mydatetime)
{
   /* make sure the user has followed the required datetime format */
   /* parameter is datetime to format check. if invalid, abort run. */
   /* if valid, return the validated (but unchanged) datetime (could skip */
   /* doing this for now, but at some point we might decide to fix up bad */
   /* string formatting as a convenience to the user, so...) */

   /* local variables */
   regex_t     myregex;
   int         myreti;
   const char *mypattern;

   mypattern =
      "^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}$";

   myreti = regcomp(&myregex, mypattern, REG_EXTENDED);
   if (myreti)
   {
      printf("Problem compiling myregex\n");
      exit(1);
   }
   myreti = regexec(&myregex, mydatetime, 0, NULL, 0);

   if (!myreti)
   {
      // printf("Match\n");
   }
   else if (myreti == REG_NOMATCH)
   {
      printf("bad time format -- must be \"YYYY-MM-DD HH:MM:SS\"\n");
      exit(1);
   }
}
```

```
$ cat validateapikeyonline.h
#define SMALL    20
#define LARGE    200

/* prototypes */
char *validateapikeyonline(int, char *, char *);
char *getkeyfromlocalfile(void);
char *make_query(char[LARGE], char[LARGE]);

char *result[SMALL];

/* global variables */
extern char endpoint[];
extern char outputfilename[LARGE];


$ cat validateapikeyonline.c
#include <string.h>
#include <stdio.h>
#include "parson.h"
#include "validateapikeyonline.h"

char *validateapikeyonline()
{
   /* check the API key for validity on the live SIE-Batch API server */

   /* local variables */
   char *      myapikeyval;
   char        params[LARGE];
   char        queryURL[LARGE];
   char *      returned_content;
   const char *status_from_json;

   JSON_Value * my_json_value;
   JSON_Object *my_json_object;

   /* get the SIE-Batch API key from local file */
   myapikeyval = getkeyfromlocalfile();

   /* assemble the parameter for checking online */
   strlcpy(params, "{\"apikey\":\"", sizeof(params));
   strlcat(params, myapikeyval, sizeof(params));
   strlcat(params, "\"}", sizeof(params));

   /* assemble the endpoint we'll visit to check */
   strlcpy(queryURL, "https://", sizeof(params));
   strlcat(queryURL, endpoint, sizeof(params));
   strlcat(queryURL, "/siebatchd/v1/validate", sizeof(params));

   /* make the query (-999 == don't write output to a file) */
   strlcpy(outputfilename, "-999", 5);
   returned_content = make_query(queryURL, params);

   /* json processing */

   /* load the returned JSON string */
   my_json_value = json_parse_string(returned_content);

   /* it initially comes in as a value, we need an object */
   my_json_object = json_object(my_json_value);
   /* extract the one value we need to confirm that the key's okay */
   status_from_json =
      json_object_get_string(my_json_object, "_status");

   return((char *)status_from_json);
} /* validateapikeyonline */
```

```
$ cat ~/.uncrustify.cfg
# from
https://raw.githubusercontent.com/uncrustify/uncrustify/master/documentation/htdocs/ben.c
fg.txt

newlines                        = LF            # AUTO (default), CRLF, CR, or LF

indent_with_tabs        = 0         # 1=indent to level only, 2=indent with tabs
input_tab_size                = 8         # original tab size
output_tab_size               = 3         # new tab size
indent_columns                = output_tab_size
# indent_label                = 0         # pos: absolute col, neg: relative column
indent_align_string           = False           # align broken strings
indent_brace                  = 0
indent_class                  = true

nl_start_of_file        = remove
# nl_start_of_file_min        = 0
nl_end_of_file                = force
nl_end_of_file_min            = 1
nl_max                        = 4
nl_before_block_comment       = 2
nl_after_func_body            = 2
nl_after_func_proto_group     = 2

nl_assign_brace               = add       # "= {" vs "= \n {"
nl_enum_brace                 = add       # "enum {" vs "enum \n {"
nl_union_brace                = add       # "union {" vs "union \n {"
nl_struct_brace               = add       # "struct {" vs "struct \n {"
nl_do_brace             = add       # "do {" vs "do \n {"
nl_if_brace             = add       # "if () {" vs "if () \n {"
nl_for_brace                  = add       # "for () {" vs "for () \n {"
nl_else_brace                 = add       # "else {" vs "else \n {"
nl_while_brace                = add       # "while () {" vs "while () \n {"
nl_switch_brace               = add       # "switch () {" vs "switch () \n {"
nl_func_var_def_blk           = 1
nl_before_case                = 1
nl_fcall_brace                = add       # "foo() {" vs "foo()\n{"
nl_fdef_brace                 = add       # "int foo() {" vs "int foo()\n{"
nl_after_return               = TRUE
nl_brace_while                = remove
nl_brace_else                 = add
nl_squeeze_ifdef        = TRUE

pos_bool                = trail           # BOOL ops on trailing end

eat_blanks_before_close_brace = TRUE
eat_blanks_after_open_brace   = TRUE


mod_paren_on_return           = add       # "return 1;" vs "return (1);"
mod_full_brace_if       = add       # "if (a) a--;" vs "if (a) { a--; }"
mod_full_brace_for            = add       # "for () a--;" vs "for () { a--; }"
mod_full_brace_do       = add       # "do a--; while ();" vs "do { a--; } while ();"
mod_full_brace_while          = add       # "while (a) a--;" vs "while (a) { a--; }"

sp_before_byref               = remove
sp_before_semi                = remove
sp_paren_paren                = remove    # space between (( and ))
sp_return_paren               = remove    # "return (1);" vs "return(1);"
sp_sizeof_paren               = remove    # "sizeof (int)" vs "sizeof(int)"
```

```
sp_before_sparen        = force           # "if (" vs "if("
sp_after_sparen               = force           # "if () {" vs "if (){"
sp_after_cast                 = remove    # "(int) a" vs "(int)a"
sp_inside_braces        = force           # "{ 1 }" vs "{1}"
sp_inside_braces_struct       = force           # "{ 1 }" vs "{1}"
sp_inside_braces_enum         = force           # "{ 1 }" vs "{1}"
sp_inside_paren               = remove
sp_inside_fparen        = remove
sp_inside_sparen        = remove
sp_inside_square        = remove
#sp_type_func                 = ignore
sp_assign               = force
sp_arith                = force
sp_bool                       = force
sp_compare              = force
sp_assign               = force
sp_after_comma                = force
sp_func_def_paren       = remove    # "int foo (){" vs "int foo(){"
sp_func_call_paren            = remove    # "foo (" vs "foo("
sp_func_proto_paren           = remove    # "int foo ();" vs "int foo();"
sp_func_class_paren           = remove
sp_before_angle               = force
sp_after_angle                = force
sp_inside_angle               = remove
sp_sparen_brace               = add
sp_fparen_brace               = add
sp_after_ptr_star       = remove
sp_before_ptr_star            = force
sp_between_ptr_star           = remove

align_with_tabs               = FALSE           # use tabs to align
align_on_tabstop        = FALSE           # align on tabstops
align_enum_equ_span           = 4
align_nl_cont                 = TRUE
align_var_def_span            = 1
align_var_def_thresh          = 12
align_var_def_inline          = TRUE
align_var_def_colon           = TRUE
align_assign_span       = 1
align_assign_thresh           = 12
align_struct_init_span        = 3
align_var_struct_span         = 99
align_right_cmt_span          = 3
align_pp_define_span          = 3
align_pp_define_gap           = 4
align_number_right            = TRUE
align_typedef_span            = 5
align_typedef_gap       = 3

cmt_star_cont                 = TRUE
```