COMP 8042 Final Project

See learn.bcit.ca for the due date

All work should be done individually.

Geographic Information System

Geographic information systems organize information pertaining to geographic features and provide various kinds of access to the information. A geographic feature may possess many attributes (see below). In particular, a geographic feature has a specific location. There are a number of ways to specify location. For this project, we will use latitude and longitude, which will allow us to deal with geographic features at any location on Earth. A reasonably detailed tutorial on latitude and longitude can be found in the Wikipedia at http://en.wikipedia.org/wiki/Latitude and http://en.wikipedia.org/wiki/Longitude.

The GIS record files were obtained from the website for the USGS Board on Geographic Names (http://geonames.usgs.gov). The file begins with a descriptive header line, followed by a sequence of GIS records, one per line, which contain the fields provided in Table 1 in Appendix in the indicated order.

Notes:

- See https://geonames.usgs.gov/docs/pubs/Nat_State_Topic_File_formats.pdf for the field descriptions.
- The type specifications used here have been modified from the source (URL above) to better reflect the realities of your programming environment.
- Latitude and longitude may be expressed in DMS (degrees/minutes/seconds, 0820830W) format, or DEC (real number, -82.1417975) format. In DMS format, latitude will always be expressed using 6 digits followed by a single character specifying the hemisphere, and longitude will always be expressed using 7 digits[1] followed by a hemisphere designator (for more information look at https://gisgeography.com/decimal-degrees-dd-minutes-seconds-dms/).
- Although some fields are mandatory, some may be omitted altogether. Best practice is to treat every field as if it may be left unspecified. Certain fields are necessary in order to index a record: the feature name and the primary latitude and primary longitude. If a record omits any of those fields, you may discard the record, or index it as far as possible.

In the GIS record file, each record will occur on a single line, and the fields will be separated by pipe ('|') symbols. Empty fields will be indicated by a pair of pipe symbols with no characters between them. See the posted VA_Monterey.txt file for many examples.

[1] While latitude lines range between -90 and +90 degrees, longitude coordinates are between -180 and +180 degrees.

COMP 8042 Final Project

See learn.bcit.ca for the due date

GIS record files are guaranteed to conform to this syntax, so there is no explicit requirement that you validate the files. On the other hand, some error-checking during parsing may help you detect errors in your parsing logic.

The file can be thought of as a sequence of bytes, each at a unique offset from the beginning of the file, just like the cells of an array. So, each GIS record begins at a unique offset from the beginning of the file.

Line Termination

Each line of a text file ends with a particular marker (known as the line terminator). In MS-DOS/Windows file systems, the line terminator is a sequence of two ASCII characters (CR + LF, `0X0D0A`). In Unix systems, the line terminator is a single ASCII character (LF). Other systems may use other line termination conventions.

Why should you care? Which line termination is used has an effect on the file offsets for all but the first record in the data file. As long as we're all testing with files that use the same line termination, we should all get the same file offsets. But if you change the file format (of the posted data files) to use different line termination, you will get different file offsets than are shown in the posted log file(s). Most good text editors will tell you what line termination is used in an opened file, and also let you change the line termination scheme.

In Figure 1, note that some record fields are optional, and that when there is no given value for a field, there are still delimiter symbols for it.

Also, some of the lines are "wrapped" to fit into the text box; lines are never "wrapped" in the actual data files.

```
FEATURE_ID|FEATURE_NAME|FEATURE_CLASS|STATE_ALPHA|STATE_NUMERIC|COUNTY_NAME|COUNTY_NUMERIC|PRIMARY_LAT_DMS|PRIM_LONG_DMS|PRIM_LAT_DEC|PRIM_LON
G_DEC|SOURCE_LAT_DMS|SOURCE_LONG_DMS|SOURCE_LAT_DEC|SOURCE_LONG_DEC|ELEV_IN_M|ELEV_IN_FT|MAP_NAME|DATE_CREATED|DATE_EDITED
1479116|Monterey Elementary School|School|VA|51|Roanoke (city)|770|371906N|0795608W|37.3183753|-
79.9355857|||||323|1060|Roanoke|09/28/1979|09/15/2010
1481345|Asbury Church|Church|VA|51|Highland|091|382607N|0793312W|38.4353981|-79.5533807|||||818|2684|Monterey|09/28/1979|
1481852|Blue Grass|Populated Place|VA|51|Highland|091|383000N|0793259W|38.5001188|-79.5497702|||||777|2549|Monterey|09/28/1979|
1481878|Bluegrass Valley|Valley|VA|51|Highland|091|382953N|0793222W|38.4981745|-79.539492|382601N|0793800W|38.4337309|-
79.6333833|759|2490|Monterey|09/28/1979|
1482110|Buck Hill|Summit|VA|51|Highland|091|381902N|0793358W|38.3173452|-79.5661577|||||1003|3291|Monterey SE|09/28/1979|
1482176|Burners Run|Stream|VA|51|Highland|091|382509N|0793409W|38.4192873|-79.5692144|382531N|0793538W|38.4252778|-
79.5938889|848|2782|Monterey|09/28/1979|
1482324|Mount Carlyle|Summit|VA|51|Highland|091|381556N|0793353W|38.2656799|-79.5647682|||||698|2290|Monterey SE|09/28/1979|
1482434|Central Church|Church|VA|51|Highland|091|382953N|0793323W|38.4981744|-79.5564371|||||773|2536|Monterey|09/28/1979|
1482557|Claylick Hollow|Valley|VA|51|Highland|091|381613N|0793238W|38.2704021|-79.5439343|381733N|0793324W|38.2925|-
79.5566667|573|1880|Monterey SE|09/28/1979|
1482785|Crab Run|Stream|VA|51|Highland|091|381707N|0793144W|38.2854018|-79.528934|381903N|0793415W|38.3175|-79.5708333|579|1900|Monterey
SE|09/28/1979|
1482950|Davis Run|Stream|VA|51|Highland|091|381824N|0793053W|38.3067903|-79.5147671|382057N|0793505W|38.3491667|-79.5847222|601|1972|Monterey
SE|09/28/1979|
1483281|Elk Run|Stream|VA|51|Highland|091|382936N|0793153W|38.4934524|-79.5314362|383121N|0793056W|38.5226185|-
79.5156027|757|2484|Monterey|09/28/1979|
1483492|Forks of Waters|Locale|VA|51|Highland|091|382856N|0793031W|38.4823417|-79.5086575|||||705|2313|Monterey|09/28/1979|
1483527|Frank Run|Stream|VA|51|Highland|091|382953N|0793310W|38.4981744|-79.5528258|383304N|0793341W|38.5512285|-
79.5614381|780|2559|Monterey|09/28/1979|
1483647|Ginseng Mountain|Summit|VA|51|Highland|091|382850N|0793139W|38.480675|-79.527547|||||978|3209|Monterey|09/28/1979|
1483860|Gulf Mountain|Summit|VA|51|Highland|091|382940N|0793103W|38.4945636|-79.5175468|||||1006|3300|Monterey|09/28/1979|
1483916|Hamilton Chapel|Church|VA|51|Highland|091|381740N|0793707W|38.2945677|-79.6186591|||||823|2700|Monterey SE|09/28/1979|
1484097|Highland High School|School|VA|51|Highland|091|382426N|0793444W|38.4071387|-79.5789333|||||879|2884|Monterey|09/28/1979|09/15/2010
1484099|Highland Wildlife Management Area|Park|VA|51|Highland|091|381905N|0793439W|38.3181785|-79.577547|||||954|3130|Monterey SE|09/28/1979|
. . .
```

Figure 1: Sample Geographic Data Records

COMP 8042 Final Project

See learn.bcit.ca for the due date

Project Description

You will implement a system that indexes and provides search features for a file of GIS records, as described above.

Your system will build and maintain several in-memory index data structures to support these operations:

- Importing new GIS records into the database file
- Retrieving data for all GIS records matching given geographic coordinates
- Retrieving data for all GIS records matching a given feature name and state
- Retrieving data for all GIS records that fall within a given (rectangular) geographic region
- Displaying the in-memory indices in a human-readable manner

You will implement a single software system in C++ to perform all system functions.

Program Invocation

The program will take the names of three files from the command line, like this:

```
./GIS <database file name> <command script file name> <log file name>
```

Note that this implies your main class must be named GIS, and be able to be compiled simply using a `g++` compile command. Preferably, you are encouraged to create make files for the project and provide the required dependency files in your submission.

The database file should be created as an empty file; note that the specified database file may already exist, in which case the existing file should be truncated or deleted and recreated. If the command script file is not found the program should write an error message to the console and exit. The log file should be rewritten every time the program is run, so if the file already exists it should be truncated or deleted and recreated.

System Overview

The system will create and maintain a GIS database file that contains all the records that are imported as the program runs. The GIS database file will be empty initially. All the indexing of records will be done relative to this file.

There is no guarantee that the GIS record file will not contain two or more distinct records that have the same geographic coordinates. In fact, this is natural since the coordinates are expressed in the usual DMS system. So, we cannot treat geographic coordinates as a primary (unique) key.

The GIS records will be indexed by the `Feature Name` and `State` (abbreviation) fields. This *name index* will support finding offsets of GIS records that match a given feature name and state abbreviation.

The GIS records will also be indexed by geographic coordinate. This *coordinate index* will support finding offsets of GIS records that match a given primary latitude and primary longitude.

The system will include a *buffer pool*, as a front end for the GIS database file, to improve search speed. See the discussion of the buffer pool below for detailed requirements. When performing searches, retrieving a GIS record from the database file must be managed through the buffer pool. During an import operation, when records are written to the database file, the buffer pool will be bypassed, since the buffer pool would not improve performance during imports.

When searches are performed, complete GIS records will be retrieved from the GIS database file that your program maintains. The only complete GIS records that are stored in memory at any time are those that have just been retrieved to satisfy the current search, or individual GIS records created while importing data or GIS records stored in the buffer pool.

Aside from where specific data structures are required, you may use any suitable STL library implementation you like (but not other libraries).

Each index should have the ability to write a nicely-formatted display of itself to an output stream.

Name Index Internals

The *name index* will use a hash table for its physical organization[2]. Each hash table entry will store a feature name and state abbreviation (separately or concatenated, as you like) and the file offset(s) of the matching record(s). Since each GIS record occupies one line in the file, it is a trivial matter to locate and read a record given nothing but the file offset at which the record begins.

[2] You are not allowed to use `std::map` here. You may use and modify any of the lab provided starter codes as your own implementation of a hash table.

Your table will use quadratic probing to resolve collisions, with the quadratic function $\frac{(n^2+n)}{2}$ to compute the step size. The hash table must use a contiguous physical structure (array). The initial size of the table will be `1024`, and the table will resize itself automatically, by doubling its size whenever the table becomes 70% full. Since the specified table sizes given above are powers of 2, an empty slot will always be found unless the table is full.

You can use your desired hash function (e.g. *elfhash*), and apply it to the concatenation of the feature name and state abbreviation field of the data records.

You must be able to display the contents of the hash table in a readable manner.

Coordinate Index Internals

The coordinate index will use a *bucket* PR quadtree[3] for the physical organization. In a bucket PR quadtree, each leaf stores up to K data objects (for some fixed value of K). Upon insertion, if the added value would fall into a leaf that is already full, then the region corresponding to the leaf will be partitioned into quadrants and the K+1 data objects will be inserted into those quadrants as appropriate. As is the case with the regular PR quadtree, this may lead to a sequence of partitioning steps, extending the relevant branch of the quadtree by multiple levels. In this project, K will probably equal 4, but I reserve the right to specify a different bucket size with little notice, so this should be easy to modify.

The index entries held in the quadtree will store a geographic coordinate and a collection of the file offsets of the matching GIS records in the database file.

Note: do not confuse the bucket size with any limit on the number of GIS records that may be associated with a single geographic coordinate. A quadtree node can contain index objects for up to K different geographic coordinates. Each such index object can contain references to an unlimited number of different GIS records.

The PR quadtree implementation should follow good design practices, and its interface should be somewhat similar to that of the BST.

You must be able to display the PR quadtree in a readable manner. The display must clearly indicate the structure of the tree, the relationships between its nodes, and the data objects in the leaf nodes (think of this problem as an in-order traversal of tree with four-children).

Buffer Pool Details

The buffer pool for the database file should be capable of buffering up to 15 records, and will use LRU replacement. You may use any structure you like to organize the pool slots; however, since the pool will have to deal with record replacements, some structures will be more efficient (and simpler) to use. You may use any classes from STL library you think are appropriate.

[3] A quadtree is a tree each node of which can have up to four children. A PR Quadtree limits quadtree nodes to either 4 children or none! (https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/PRquadtree.html)

It is up to you to decide whether your buffer pool stores interpreted or raw data; i.e., whether the buffer pool stores GIS record objects or just strings.

You must be able to display the contents of the buffer pool, listed from MRU to LRU entry, in a readable manner. The order in which you retrieve records when servicing a multi-match search is not specified, so such searches may result in different orderings of the records within the buffer pool. That is OK.

A Note on Coordinates and Spatial Regions

It is important to remember that there are fundamental differences between the notion that a geographic feature has specific coordinates (which may be thought of as a point) and the notion that each node of the PR quadtree corresponds to a particular sub-region of the coordinate space (which may contain many geographic features).

In this project, coordinates of geographic features are specified as latitude/longitude pairs, and the minimum resolution is one second of arc. Thus, you may think of the geographic coordinates as being specified by a pair of integer values.

On the other hand, the boundaries of the sub-regions are determined by performing arithmetic operations, including division, starting with the values that define the boundaries of the "world". Unless the dimensions of the world happen to be powers of 2, this can quickly lead to regions whose boundaries cannot be expressed exactly as integer values. You may use floating-point values or integer values to represent region boundaries when computing region boundaries during splitting and quadtree traversals. If you use integers, be careful not to unintentionally create "gaps" between regions.

Your implementation should view the boundary between regions as belonging to one of those regions. The choice of a particular rule for handling this situation is left to you.

When carrying out a region search, you must determine whether the search region overlaps with the region corresponding to a subtree node before descending into that subtree. Think about a proper divide and conquer approach to solve the region search problem.

Other System Elements

There should be an overall controller that validates the command line arguments and manages the initialization of the various system components. The controller should hand off execution to a command processor that manages retrieving commands from the script file, and making the necessary calls to other components in order to carry out those commands.

Naturally, there should be a data type that models a GIS record.

There may well be additional system elements, whether data types or data structures, or system components that are not mentioned here. The fact no additional elements are explicitly identified here does not imply that you will not be expected to analyze the design issues carefully, and to perhaps include such elements.

Aside from the command-line interface, there are no specific requirements for interfaces of any of the classes that will make up your software; it is up to you to analyze the specification and come up with an appropriate set of classes, and to design their interfaces to facilitate the necessary interactions. It is probably worth pointing out that an index (e.g., a geographic coordinate index) should not simply be a naked container object (e.g, quadtree); if that's not clear to you, think more carefully about what sort of interface would be appropriate for an index, as opposed to a container.

Command File

The execution of the program will be driven by a script file. Lines beginning with a semicolon character (';') are comments and should be ignored. Blank lines are possible. Each line in the command file consists of a sequence of tokens, which will be separated by single tab characters. A line terminator will immediately follow the final token on each line. The command file is guaranteed to conform to this specification, so you do not need to worry about error-checking when reading it.

The first non-comment line will specify the world boundaries to be used:

```
world<tab><westLong><tab><eastLong><tab><southLat><tab><northLat>
```

> This will be the first command in the file, and will occur once. It specifies the boundaries of the coordinate space to be modeled. The four parameters will be longitude and latitudes expressed in DMS format, representing the vertical and horizontal boundaries of the coordinate space.
> It is certainly possible that the GIS record file will contain records for features that lie outside the specified coordinate space. Such records should be ignored; i.e., they will not be indexed.

Each subsequent non-comment line of the command file will specify one of the commands described below.
One command is used to load records into your database from external files:

```
import<tab><GIS record file>
```

> Add all the valid GIS records in the specified file to the database file. This means that the records will be appended to the existing database file, and that those records will be indexed in the manner described earlier. When the import is completed, log the number of entries added to each index, and the longest probe sequence that was needed when inserting to the hash table. (A valid record is one that lies within the specified world boundaries.)
> Note: `<GIS record file>` will be a relative address (e.g. `./VA_Monterey.txt` will point to the file named `VA_Monterey.txt` which is placed besides the GIS executable.

Another command requires producing a human-friendly display of the contents of an index structure:

`debug<tab>[ quad | hash | pool | world]`

Log the contents of the specified index structure in a fashion that makes the internal structure and contents of the index clear. It is not necessary to be overly verbose here, but you should include information like key values and file offsets where appropriate.
Hint: for `debug<tab>world` think of a recursive method that can divide the 2-D world space in 4 parts each time and continue the recursion for a number of times.

Another simply terminates execution, which is handy if you want to process only part of a command file:

`quit<tab>`

Terminate program execution.

The other commands involve searches of the indexed records. For the following commands, if a `<geographic coordinate>` is specified for the command, it will be expressed as a pair of latitude/longitude values, expressed in the same DMS format that is used in the GIS record files. In the script files, `<geographic coordinate>` will show up as `<geographic coordinate latitude><tab><geographic coordinate longitude>`.

`what_is_at<tab><geographic coordinate>`

For every GIS record in the database file that matches the given `<geographic coordinate>`, log the offset at which the record was found, and the feature name, county name, and state abbreviation. Do not log any other data from the matching records.

`what_is<tab><feature name><tab><state abbreviation>`

For every GIS record in the database file that matches the given `<feature name>` and `<state abbreviation>`, log the offset at which the record was found, and the county name, the primary latitude, and the primary longitude. Do not log any other data from the matching records.

`what_is_in<tab><geographic coordinate><tab><half-height><tab><half-width>`

For every GIS record in the database file whose coordinates fall within the closed rectangle with the specified height and width, centered at the `<geographic coordinate>`, log the offset at which the record was found, and the feature name, the state name, and the primary latitude and primary longitude. Do not log any other data from the matching records. The half-height and half-width are specified as total seconds.

The `what_is_in` command takes an optional modifier, `-long`, which causes the display of a long listing of the relevant records. The switch will be the first token following the name of the command. If this switch is present, then for every GIS record in the database file whose coordinates fall within the closed rectangle with the specified height and width, centered at the `<geographic coordinate>`, log every important non-empty field, nicely formatted and labeled. See the posted log file(s) for an example. Do not log any empty fields. The half-height and half-width are specified as seconds.

The `what_is_in` command also takes an optional modifier, causing the search results to be filtered: `-filter [ pop | water | structure ]`
The switch and its modifier will be the first and second tokens following the name of the command (for simplicity we assume that either `-filter` will be provided or `-long` optional modifier, but not both). If present, this causes the set of matching records to be filtered to only show those whose feature type field corresponds to the given filter specifier. See Table 2 in the Appendix for instructions on how to interpret the feature types shown above.

Sample command script(s), and corresponding log file(s), are provided alongside this description file. As a general rule, every command should result in some output. In particular, a descriptive message should be logged if a search yields no matching records. Also, to make sure you are logging everything properly, please avoid using `std::cout` in your code unless you are debugging.

Log File Description

Your output should be clear, concise, well labeled, and correct. You should begin the log with a few lines identifying yourself, and listing the names of the input files that are being used.

The remainder of the log file output should come directly from your processing of the command file. You are required to echo each comment line, and each command that you process to the log file so that it's easy to determine which command each section of your output corresponds to. Each command (except for "world") should be numbered, starting with 1, and the output from each command should be well formatted, and delimited from the output resulting from processing other commands.

COMP 8042 Final Project

See learn.bcit.ca for the due date

Quick Start

Reading through this whole document might be quite overwhelming and could make it hard to get started. This section is intended to reduce that complexity and give you a road-map on the tasks that need to be done in order. Following this road-map will help you save time while enabling you to finish up each part as soon as you learn about it in the class. Lets get started:

1. This project has a certain structure described throughout the document. Get started by creating a `GIS.cpp` file and populating the project structure in it. You will need a `main` function, as well as the classes: `GISRecord`, `NameIndex`, `CoordinateIndex`, `BufferPool`, `Logger`, `SystemManager`, and `CommandProcessor`. You will also need a `enum Command` structure for the different commands described in the project description and a `struct DMS` to keep the parsed coordinates.

2. Once done with the project structure, get started on filling up the `GISRecord` and `CommandProcessor` classes. By the end of this step your project should be able to read the script file and parse its content reporting different commands and the arguments passed to each. Your project should also be able to create the database file and log file and fill out the initial and final lines of the execution logs as well as copying the comments and commands using `Logger` class. Note that the results of running commands are not expected in this step.

3. Next tackle implementing the first command, `world`. In this step, you don't need to create indices with it but later you will modify the code to initiate the indices along with world creation.

4. Fill up the `BufferPool` to write to a file, read from a file, and fill up/use its cache. Make sure it has a `str()` command implemented to be used for `debug pool`.

5. Work on `import` command. In this step, you will need to create your `HashTable` and `PRQuadtree` classes to support *NameIndex* and *CoordinateIndex* classes. Fill out your support classes with the help of what you have learnt about Trees and Hashing. You can use the starter code provided in the labs as a starting point for this step as well.

6. To complete `what_is` command, finish up your *NameIndex*. Make sure it has a `str()` command implemented to be used for `debug hash`.

7. To complete `what_is_at` and `what_is_in` commands, finish up your *CoordinateIndex*. Make sure it has `str()` and (optionally) `visualize()` functions implemented to be used for `debug quad` and `debug world`.

8. Wrap up the project by re-reading through the project description and taking care of the details that have been left out.

9. You are done!

Submission

For this project, you must submit an archive (zip or tar) file containing all the source code files for your implementation (i.e., `.cpp` files). Submit only the source files. Do not submit the compiled files or any of object files. If you use packages in your implementation (and that's good practice), your archive file must include the correct directory structure for those packages, and your GIS.cpp file must be in the top directory when the archive file is unpacked. **Your code must be ready to compile using `g++ -std=c++11` or a simple Makefile if you have a more complicated structure**. Make sure no visual studio related dependencies or solution files are there when submitting the result, since I certainly will not use visual studio to test and grade your project.

Alongside your source files, I need a pdf file describing your solution, general architecture of your code, and the list of data structures you have implemented or used from STL. Run `script01.txt` and put a screen shot of its created log file, as well. **Don't forget to comment out all the debug commands to produce this log file.**

The correctness of your solution will be evaluated by executing your solution on a collection of test data files. Be sure to test your solution with all of the data sets that are posted, since I will use a variety of data sets, including at least one very large data one (perhaps hundreds of thousands of records) in my evaluation.

As it is stated in the beginning of the file description this project **must** be done individually. You are **not allowed to copy a single function from another person** nor from the internet without citing them. **You may use any of the previously provided lab source codes completed by yourself to reduce the implementation time of this project** (don't forget to mention which lab codes you have used in your final report).

Appendix

Table 1: Geographic Data Record Format

Name	Type	Length/ Decimals	Short Description
Feature ID	Integer	10	Permanent, unique feature record identifier
Feature Name	String	120	and official feature name
Feature Class	String	50	See Table 2 later in this specification
State Alpha	String	2	The unique two letter alphabetic code and the unique two number code for a US State
State Numeric	String	2	
County Name	String	100	The name and unique three number code for a county or county equivalent
County Numeric	String	3	
Primary Latitude DMS	String	7	The official feature location *DMS-degrees/minutes/seconds* *DEC-decimal degrees* Note: Records showing "Unknown" and zeros for the latitude and longitude DMS and decimal fields, respectively, indicate that the coordinates of the feature are unknown. They are recorded in the database as zeros to satisfy the format requirements of a numerical data type. They are not errors and do not reference the actual geographic coordinates at 0 latitude, 0 longitude.
Primary Longitude DMS	String	8	
Primary Latitude DEC	Real Number	11/7	
Primary Longitude DEC	Real Number	12/7	
Source Latitude DMS	String	7	Source coordinates of linear feature only (Class = Stream, Valley, Arroyo) *DMS-degrees/minutes/seconds* *DEC-decimal degrees* Note: Records showing "Unknown" and zeros for the latitude and longitude DMS and decimal fields, respectively, indicate that the coordinates of the feature are unknown. They are recorded in the database as zeros to satisfy the format requirements of a numerical data type. They are not errors and do not reference the actual geographic coordinates at 0 latitude, 0 longitude.
Source Longitude DMS	String	8	

Source Latitude DEC	Real Number	11/7	
Source Longitude DEC	Real Number	12/7	
Elevation (meters)	Integer	5	Elevation in meters above (-below) sea level of the surface at the primary coordinates
Elevation (feet)	Integer	6	Elevation in feet above (-below) sea level of the surface at the primary coordinates
Map Name	String	100	Name of USGS base series topographic map containing the primary coordinates.
Date Created	String	Date	The date the feature was initially committed to the database.
Date Edited	String	Date	The date any attribute of an existing feature was last edited.

The Feature Class field of the GIS records may contain any of the following designators. We care about these because of the `-filter` switch that may be used with the `what_is_in` commands. The table below indicates which of the standard correspond to one of the filter specifiers.

Table 2: Feature Class Designators

Class	Type	Description
Airport	structure	Manmade facility maintained for the use of aircraft (airfield, airstrip, landing field, landing strip).
Arch		Natural arch-like opening in a rock mass (bridge, natural bridge, sea arch).
Area		Any one of several areally extensive natural features not included in other categories (badlands, barren, delta, fan, garden).
Arroyo	water	Watercourse or channel through which water may occasionally flow (coulee, draw, gully, wash).
Bar		Natural accumulation of sand, gravel, or alluvium forming an underwater or exposed embankment (ledge, reef, sandbar, shoal, spit).
Basin		Natural depression or relatively low area enclosed by higher land (amphitheater, cirque, pit, sink).
Bay	water	Indentation of a coastline or shoreline enclosing a part of a body of water; a body of water partly surrounded by land (arm, bight, cove, estuary, gulf, inlet, sound).

Beach		The sloping shore along a body of water that is washed by waves or tides and is usually covered by sand or gravel (coast, shore, strand).
Bench		Area of relatively level land on the flank of an elevation such as a hill, ridge, or mountain where the slope of the land rises on one side and descends on the opposite side (level).
Bend	water	Curve in the course of a stream and (or) the land within the curve; a curve in a linear body of water (bottom, loop, meander).
Bridge	structure	Manmade structure carrying a trail, road, or other transportation system across a body of water or depression (causeway, overpass, trestle).
Building	structure	A manmade structure with walls and a roof for protection of people and (or) materials, but not including church, hospital, or school.
Canal	water	Manmade waterway used by watercraft or for drainage, irrigation, mining, or water power (ditch, lateal).
Cape		Projection of land extending into a body of water (lea, neck, peninsula, point).
Cave		Natural underground passageway or chamber, or a hollowed out cavity in the side of a cliff (cavern, grotto).
Cemetery		A place or area for burying the dead (burial, burying ground, grave, memorial garden).
Census		A statistical area delineated locally specifically for the tabulation of Census Bureau data (census designated place, census county division, unorganized territory, various types of American Indian/Alaska Native statistical areas). Distinct from Civil and Populated Place.
Channel	water	Linear deep part of a body of water through which the main volume of water flows and is frequently used as aroute for watercraft (passage, reach, strait, thoroughfare, throughfare).
Church	structure	Building used for religious worship (chapel, mosque, synagogue, tabernacle, temple).
Civil		A political division formed for administrative purposes (borough, county, incorporated place, municipio, parish, town, township). Distinct from Census and Populated Place.

Cliff		Very steep or vertical slope (bluff, crag, head, headland, nose, palisades, precipice, promontory, rim, rimrock).
Crater		Circular-shaped depression at the summit of a volcanic cone or one on the surface of the land caused by the impact of a meteorite; a manmade depression caused by an explosion (caldera, lua).
Crossing		A place where two or more routes of transportation form a junction or intersection (overpass, underpass).
Dam	structure	Water barrier or embankment built across the course of a stream or into a body of water to control and (or) impound the flow of water (breakwater, dike, jetty).
Falls	water	Perpendicular or very steep fall of water in the course of a stream (cascade, cataract, waterfall).
Flat		Relative level area within a region of greater relief (clearing, glade, playa).
Forest		Bounded area of woods, forest, or grassland under the administration of a political agency (see "woods") (national forest, national grasslands, State forest).
Gap		Low point or opening between hills or mountains or in a ridge or mountain range (col, notch, pass, saddle, water gap, wind gap).
Glacier	water	Body or stream of ice moving outward and downslope from an area of accumulation; an area of relatively permanent snow or ice on the top or side of a mountain or mountainous area (icefield, ice patch, snow patch).
Gut	water	Relatively small coastal waterway connecting larger bodies of water or other waterways (creek, inlet, slough).
Harbor	water	Sheltered area of water where ships or other watercraft can anchor or dock (hono, port, roads, roadstead).
Hospital	structure	Building where the sick or injured may receive medical or surgical attention (infirmary).
Island		Area of dry or relatively dry land surrounded by water or low wetland (archipelago, atoll, cay, hammock, hummock, isla, isle, key, moku, rock).
Isthmus		Narrow section of land in a body of water connecting two larger land areas.
Lake	water	Natural body of inland water (backwater, lac, lagoon, laguna, pond, pool, resaca, waterhole).

Lava		Formations resulting from the consolidation of molten rock on the surface of the Earth (kepula, lava flow).
Levee	structure	Natural or manmade embankment flanking a stream (bank, berm).
Locale		Place at which there is or was human activity; it does not include populated places, mines, and dams (battlefield, crossroad, camp, farm, ghost town, landing, railroad siding, ranch, ruins, site, station, windmill).
Military		Place or facility used for various aspects of or relating to military activity.
Mine		Place or area from which commercial minerals are or were removed from the Earth; not including oilfield (pit, quarry, shaft).
Oilfield		Area where petroleum is or was removed from the Earth.
Park	structure	Place or area set aside for recreation or preservation of a cultural or natural resource and under some form of government administration; not including National or State forests or Reserves (national historical landmark, national park, State park, wilderness area).
Pillar		Vertical, standing, often spire-shaped, natural rock formation (chimney, monument, pinnacle, pohaku, rock tower).
Plain		A region of general uniform slope, comparatively level and of considerable extent (grassland, highland, kula, plateau, upland).
Populated Place	pop	Place or area with clustered or scattered buildings and a permanent human population (city, settlement, town, village). A populated place is usually not incorporated and by definition has no legal boundaries. However, a populated place may have a corresponding "civil" record, the legal boundaries of which may or may not coincide with the perceived populated place. Distinct from Census and Civil classes.
Post Office	structure	An official facility of the U.S. Postal Service used for processing and distributing mail and other postal material.
Range		Chain of hills or mountains; a somewhat linear, complex mountainous or hilly area (cordillera, sierra).
Rapids	water	Fast-flowing section of a stream, often shallow and with exposed rock or boulders (riffle, ripple).
Reserve		A tract of land set aside for a specific use (does not include forests, civil divisions, parks).
Reservoir	water	Artificially impounded body of water (lake, tank).
Ridge		Elevation with a narrow, elongated crest which can be part of a hill or mountain (crest, cuesta, escarpment, hogback, lae, rim, spur).

School	structure	Building or group of buildings used as an institution for study, teaching, and learning (academy, college, high school, university).
Sea	water	Large body of salt water (gulf, ocean).
Slope		A gently inclined part of the Earth's surface (grade, pitch).
Spring	water	Place where underground water flows naturally to the surface of the Earth (seep).
Stream	water	Linear body of water flowing on the Earth's surface (anabranch, awawa, bayou, branch, brook, creek, distributary, fork, kill, pup, rio, river, run, slough).
Summit		Prominent elevation rising above the surrounding level of the Earth's surface; does not include pillars, ridges, or ranges (ahu, berg, bald, butte, cerro, colina, cone, cumbre, dome, head, hill, horn, knob, knoll, mauna, mesa, mesita, mound, mount, mountain, peak, puu, rock, sugarloaf, table, volcano).
Swamp	water	Poorly drained wetland, fresh or saltwater, wooded or grassy, possibly covered with open water (bog, cienega, marais, marsh, pocosin).
Tower	structure	A manmade structure, higher than its diameter, generally used for observation, storage, or electronic transmission.
Trail		Route for passage from one point to another; does not include roads or highways (jeep trail, path, ski trail).
Tunnel	structure	Linear underground passageway open at both ends.
Unknown		This class is assigned to legacy data only. It will not be assigned to new or edited records.
Valley		Linear depression in the Earth's surface that generally slopes from one end to the other (barranca, canyon, chasm, cove, draw, glen, gorge, gulch, gulf, hollow, ravine).
Well	water	Manmade shaft or hole in the Earth's surface used to obtain fluid or gaseous materials.
Woods		Small area covered with a dense growth of trees; does not include an area of trees under the administration of a political agency (see "forest").