GEONIS Tools Documentation

Release 0.9.8

Geocom Informatik AG / VertiGIS

Feb 03, 2020

Contents:

CHAPTER 1

gntools package

1.1 Subpackages

1.1.1 gntools.common package

1.1.1.1 Submodules

1.1.1.1.1 gntools.common.geometry module

GEONIS-specific module to handle or parse Esri geometries.

exception gntools.common.geometry.**GeometrySerializationError**
Bases: exceptions.ValueError

gntools.common.geometry.**get_angle**(*center_point*, *from_arcpoint*, *to_arcpoint*)
Calculates and returns the angle of a center-point arc in degrees.

Parameters

- **center_point** (*tuple, list*) – Arc center point ((x, y)).
- **from_arcpoint** (*tuple, list*) – The start point of the arc ((x, y, ...)).
- **to_arcpoint** (*tuple, list*) – The last point of the arc ((x, y, ...)).

Returns An angle in degrees.

Return type float

gntools.common.geometry.**get_arc_center**(*start_point*, *mid_point*, *end_point*)
Calculates and returns the center point as a tuple of (x, y) for the defined 3-point arc points.

Parameters

- **start_point** (*tuple, list*) – First point ((x, y, ...)).
- **mid_point** (*tuple, list*) – Middle or other point on the arc ((x, y, ...)).

- **end_point** (*tuple, list*) – Last point ((x, y, ...)).

Returns A tuple of X, Y float values.

Return type tuple

gntools.common.geometry.**is_clockwise**(*ring*)

Returns True when the given list or tuple of polygon ring coordinates turns clockwise.

This is achieved by calculating an area approximation for the ring using the Shoelace formula. When the area is positive, the ring turns clockwise. When negative, the ring turns counterclockwise.

Parameters **ring** (*tuple, list*) – An EsriJSON ring.

Return type bool

gntools.common.geometry.**is_minor**(*start_point*, *mid_point*, *end_point*)

Returns True when the defined 3-point arc is minor (less than 180 degrees).

Parameters

- **start_point** (*tuple, list*) – First point ((x, y, ...)).
- **mid_point** (*tuple, list*) – Middle or other point on the arc ((x, y, ...)).
- **end_point** (*tuple, list*) – Last point ((x, y, ...)).

Return type bool

gntools.common.geometry.**serialize**(*geometry*)

Serializes Esri Geometry, an Esri Point, EsriJSON or a coordinate iterable into GEONIS Protocol XML geometry. Regardless of the dimensions of the input geometry, the output geometry will always be 2D.

Parameters **geometry** (*Geometry, str, unicode, tuple, list*) – An Esri Geometry or Point instance, an Esri JSON string or a coordinate iterable.

Returns An XML 'Geometry' element.

Return type Element

See also:

gntools.protocol.Logger, *gntools.protocol.Feature*

1.1.1.1.2 gntools.common.const module

Module with specific GEONIS-related constants. All GPF constants are imported as well, which means that this module basically extends the gpf.common.const module.

This module also contains GEONIS definition mappings (*GNTABLES* and *GNFIELDS*, for example). These mappings are stored as dictionary structures. Mapping structures determine which Python Definition property/attribute is mapped to what GEONIS definition (e.g. table or field name). When a language-based or custom override is available, the definition override is looked up in the definition table for a certain solution, which is stored in the database. If this table does not exist, or no override was found, the default (German) value is returned instead.

A mapping should always have the following dictionary structure:

```
mapping_name = {
  solution_name1: {
      code_property_name1: (definition_table_property_name, default_definition_value),
      code_property_name2: (..., ...),
      ...
  },
```

(continues on next page)

(continued from previous page)

```
  solution_name2: {
      ...
  }
}
```

If no definition table property name (i.e. lookup key) is available, it should be set to `None`. Default definition values should always have a value, so that any failed lookup immediately returns this value.

1.1.1.2 Module contents

The *common* subpackage contains modules that can be used across various other subpackages.

1.2 Submodules

1.2.1 gntools.datasources module

The *datasource* module simplifies working with GEONIS Datasource XML files.

class gntools.datasources.**Datasource**(*datasource*)

Bases: gpf.paths.Workspace

Helper class to read a GEONIS Datasource XML, so that the user is able to generate fully qualified paths for elements (tables, feature datasets etc.) in an Esri workspace. To GEONIS, an Esri Workspace means an SDE connection file or a local File/Personal Geodatabase.

Params:

- **xml_path** (str, unicode):

  The full GEONIS Datasource XML file path.

  Raises **ValueError** – If the GEONIS Datasource XML does not exist or failed to parse.

Note: All class methods are inherited from `gpf.paths.Workspace`. The initialization process of the *Datasource* class is different, because it will read the workspace path from the GEONIS Datasource XML. Another difference is that the *Datasource* class also stores the GEONIS *medium* it applies to. Furthermore, it features 2 helper functions, which try to guess the paths of the media directory (*get_media_dir()*) and the project directory (*get_project_dir()*) for that same medium.

---

basename(*keep_ext=True*)

Returns a file name (if initial path is a file) or a directory name.

Parameters **keep_ext** (*bool*) – For files, setting this to `False` will remove the file extension. Defaults to `True`. Note that for directories, this might not have any effect.

Return type str, unicode

exists

Returns `True` if the workspace path exists and/or is a valid Esri workspace.

Return type bool

extension (*keep_dot=True*)
Returns the extension part of the initial path. For directories or files without extension, an empty string is returned.

Parameters **keep_dot** (*bool*) – When `False`, the extension's trailing dot will be removed. Defaults to `True`.

Return type str, unicode

find_path (*table*, *refresh=False*)
Tries to resolve the full (qualified) path for the specified table or feature class and returns it, by looking up the matching feature dataset (or workspace root when not found). In contrast to the *make_path()* function, the path is verified before it is returned.

Parameters

- **table** (*str*) – The (unqualified) table or feature class name to find.
- **refresh** (*bool*) – Updates the internal table lookup first. See note below.

Return type str

Raises **ValueError** – If the table was not found or found multiple times (should not happen).

Example:

```
>>> wm = Workspace(r'C:/temp/db_user.sde')
>>> wm.qualifier
'user'
>>> wm.find_path('ele_cable')  # finds the feature class in feature dataset
↪"ELE"
'C:\temp\db_user.sde\user.ele\user.ele_cable'
```

Note: The feature dataset lookup is created once on the first call to this function. This means that the first call is relatively slow and consecutive ones are fast. When the user creates new feature class paths using the *make_path()* method, the lookup is updated automatically, so that this function can find the new feature class. However, when the workspace is updated *from the outside*, the lookup is not updated. If the user wishes to force-update the lookup, set the *refresh* argument to `True`.

from_basename (*basename*)
Returns the initial path with an alternative basename. This will work for both directories and files.

Parameters **basename** (*str, unicode*) – The new basename. If basename contains an extension and the initial path is a file path that also had an extension, both the name and extension will be changed. If the initial path is a directory path, the directory name will simply be replaced by the basename.

Return type str, unicode

Examples:

```
>>> with Path(r'C:/temp/myfile.txt') as pm:
>>>     pm.from_basename('newfile')
C:\temp\newfile.txt
>>>     pm.from_basename('newfile.log')
C:\temp\newfile.log
```

from_extension (*extension*, *force=False*)
Returns the initial path with an alternative file *extension*. If the initial path did not have an extension (e.g.

when it is a directory), this function will return the initial unmodified path instead (and have no effect), unless *force* is `True`.

Parameters

- **extension** (*str, unicode*) – New file extension (with or without trailing dot).
- **force** (*bool*) – When `True`, the extension will always be appended, even if the initial path is a directory. The default is `False`.

Return type str, unicode

Examples:

```
>>> with Path(r'C:/temp/myfile.txt') as pm:
>>>     pm.from_extension('log')
C:\temp\myfile.log
>>> with Path(r'C:/temp/mydir') as pm:
>>>     pm.from_extension('log')
C:\temp\mydir
>>>     pm.from_extension('gdb', force=True)
C:\temp\mydir.gdb
```

get_media_dir()
Gets the derived full path to the corresponding GEONIS media directory.

Raises **OSError** – When the derived directory path does not exist.

classmethod get_parent(*path*, *outside_gdb=False*)
Class method that extracts the parent workspace path for a given Esri table/feature class path. Depending on the path, this might return a feature dataset workspace or the root workspace path.

Parameters

- **path** (*str, unicode*) – Full path to an Esri table, feature class or feature dataset.
- **outside_gdb** (*bool*) – If `True`, this will allow the function to return the parent directory of a Geodatabase, if the workspace is a GDB path. This effectively means that the function is allowed to go 'outside' of the Geodatabase. By default, this value is set to `False`. For non-Geodatabase paths, this will have no effect: the parent directory is returned.

Return type str, unicode

Examples:

```
>>> Workspace.get_parent(r'C:/temp/test.gdb')
'C:\temp\test.gdb'
>>> Workspace.get_parent(r'C:/temp/test.gdb/feature_dataset')
'C:\temp\test.gdb'
>>> Workspace.get_parent(r'C:/temp/test.gdb', outside_gdb=True)
'C:\temp'
>>> Workspace.get_parent(r'C:/temp/test.shp')
'C:\temp'
```

get_project_dir()
Gets the derived full path to the corresponding GEONIS project directory.

Raises **OSError** – When the derived directory path does not exist.

classmethod get_root(*path*)
Class method that extracts the root workspace for a given Esri table/feature class path.

A root workspace is the Esri workspace of the "highest order". For an SDE feature class, this is the SDE connection file. For a File Geodatabase table, this is the File Geodatabase directory (.gdb) itself. For a Shapefile path, this will return the parent directory. For an in memory workspace, this will return 'in_memory'.

Parameters **path** (*str, unicode*) – Full path to an Esri table, feature class or feature dataset.

Return type str, unicode

Examples:

```
>>> Workspace.get_root(r'C:/temp/test.gdb/ele/ele_kabel')
'C:\temp\test.gdb'
>>> Workspace.get_root(r'C:/temp/mydir/test.shp')
'C:\temp\mydir
>>> Workspace.get_root(r'C:/temp/test.gdb/ele')
'C:\temp\test.gdb'
```

is_dir

Returns `True` if the initial path is an existing directory.

Return type bool

is_file

Returns `True` if the initial path is an existing file.

Return type bool

is_gdb

Returns `True` if the workspace path seems to be an Esri Geodatabase (remote or local).

Return type bool

is_remote

Returns `True` if the workspace path seems to be a remote SDE geodatabase connection.

Return type bool

make_path(**parts*, *{qualifier}*, *{separator}*)

Constructs a (qualified) path for the given named parts (data elements) in the order they appear.

Parameters

- **parts** – Feature dataset, feature class and/or table name(s) to concatenate. Note that if the workspace is a FileSystem directory, the last part of *parts* should include a file extension (e.g. '.shp').
- **qualifier** – Optional qualifier if the one derived from the DB connection should be overridden.
- **separator** – Optional separator if the initial one should be overridden (defaults to '.').

Return type str, unicode

Raises **IndexError** – When more than 2 *parts* have been specified, this function will fail.

In the following example, the qualifier ("user") is derived from the connection:

```
>>> wm = Workspace(r'C:/temp/db_user.sde')
>>> wm.qualifier
'user'
>>> wm.make_path('ele', 'ele_kabel')
'C:\temp\db_user.sde\user.ele\user.ele_kabel'
```

Using the `Workspace` above, we can override the qualifier with a custom one:

```
>>> wm.make_path('ele', 'ele_kabel', qualifier='editor')
'C:\temp\db_user.sde\editor.ele\editor.ele_kabel'
```

medium
Returns the name of the GEONIS medium to which the data source applies.

parent
Returns the parent workspace as a new `Workspace` instance.

Return type Workspace

qualifier
Returns the qualifier for the current Esri workspace. For local workspaces, this is an empty string. The trailing separator will not be included in the output string.

Return type str, unicode

qualify(*name*, *qualifier=None*, *separator=None*)
Qualifies (prefixes) a data element name for SDE workspaces. If the workspace is not an SDE workspace or the name is qualified already, the input name is returned as-is.

Parameters

- **name** (*str, unicode*) – Feature dataset, feature class or table name.
- **qualifier** (*str, unicode*) – Optional qualifier if the one derived from the DB connection should be overridden.
- **separator** (*str, unicode*) – Optional separator if the initial one should be overridden (defaults to `'.'`).

Return type str, unicode

Raises **ValueError** – If no table name was specified.

root
Returns the root workspace as a new `Workspace` instance. If the initial path already was the root path, this will return the current instance (`self`).

Return type Workspace

separator
Returns the separator for the current Esri workspace. For local workspaces, this is an empty string.

Return type str, unicode

1.2.2 gntools.definitions module

1.2.3 gntools.parsers module

This module contains classes that help parse arguments for GEONIS menu or form-based Python scripts.

class gntools.parsers.**FormArgParser**(**param_names*)
Bases: gntools.parsers._BaseArgParser

Data class that reads and stores the Python arguments for GEONIS form scripts. Simply instantiate this class in a form-based Python script to get easy access to all arguments passed to it.

Script argument values are cleaned before they are returned, which means that excessive single or double quotes will be removed. Furthermore, any "#" values (representing NoData placeholders) are replaced by `None` and trailing NoData values are removed.

Using the GEONIS form definition (XML), a user can pass additional parameters to the script. These parameters do not have a name, but the `FormArgParser` can name them for you if it is initialized with one or more parameter names, so you can access them as a namedtuple.

Example:

```
>>> params = FormArgParser('arg1', 'arg2')
>>> params.arguments.arg1
'This value has been set in the GEONIS form XML'
>>> params.arguments.arg2
'This is another argument'
```

```
>>> # Note that you can still get the arguments by index or unpack as a tuple
>>> params.arguments[1]
'This is another argument'
>>> params.arguments == 'This value has been set in the GEONIS form XML', 'This␣
↪is another argument'
True
```

If you don't specify any parameter names, the `arguments` property returns a regular `tuple`.

Params:

- **param_names**:

  Optional parameter names to set on the parsed form arguments.

arguments
Returns the arguments passed to the script (if defined in the GEONIS XML script configuration).

Returns A namedtuple or tuple with additional script arguments (if any).

field_value
The value of the table key field name for the feature/row to which the form script applies.

Return type str

key_field
Field name that holds the key value (ID) for the feature/row to which the form script applies.

Return type str

project_vars
Any optional GEONIS project variables passed to the script (often not used).

Return type dict

script
The full path to the script that was called by the Python interpreter.

Return type str

table
Table or feature class name to which the form script applies.

Return type str

workspace
Returns a `gpf.paths.Workspace` instance for the Esri workspace (and optionally a qualifier) specified in the script arguments.

Return type gpf.paths.Workspace

class gntools.parsers.**MenuArgParser**(**param_names*)

Bases: gntools.parsers._BaseArgParser

Data class that reads and stores the Python arguments for GEONIS menu scripts. Simply instantiate this class in a menu-based Python script to get easy access to all arguments passed to it.

Script argument values are cleaned before they are returned, which means that excessive single or double quotes will be removed. Furthermore, any "#" values (representing NoData placeholders) are replaced by `None` and trailing NoData values are removed.

Using the GEONIS menu definition (XML), a user can pass additional parameters to the script. These parameters do not have a name, but the `MenuArgParser` can name them for you if it is initialized with one or more parameter names, so you can access them as a namedtuple.

Example:

```
>>> params = MenuArgParser('arg1', 'arg2')
>>> params.arguments.arg1
'This value has been set in the GEONIS menu XML'
>>> params.arguments.arg2
'This is another argument'
```

```
>>> # Note that you can still get the arguments by index or unpack as a tuple
>>> params.arguments[1]
'This is another argument'
>>> params.arguments == 'This value has been set in the GEONIS menu XML', 'This␣
↪is another argument'
True
```

If you don't specify any parameter names, the `arguments` property returns a regular `tuple`.

Params:

- **param_names**:

  Optional parameter names to set on the parsed menu arguments.

arguments

Returns the arguments passed to the script (if defined in the GEONIS XML script configuration).

Returns A namedtuple or tuple with additional script arguments (if any).

project_vars

Any optional GEONIS project variables passed to the script (often not used).

Return type dict

script

The full path to the script that was called by the Python interpreter.

Return type str

workspace

Returns a `gpf.paths.Workspace` instance for the Esri workspace (and optionally a qualifier) specified in the script arguments.

Return type gpf.paths.Workspace

exception gntools.parsers.**ParameterWarning**

Bases: exceptions.UserWarning

Warning that is displayed when a (minor) issue occurred while parsing the script parameters.

args

message

gntools.parsers.**clean_arg**(*value*, *default*)
Strips all trailing quotes and NoData (#) values from text.

Parameters

- **value** – The argument value that should be evaluated. If this is not a string, it will be passed as-is.
- **default** – The default value that should be returned if the value is a NoData string (#).

gntools.parsers.**eval_arg**(*value*, *default*)
Evaluates a literal string as a Python object in a safe manner.

Parameters

- **value** (*str, unicode*) – The string that should be evaluated. If it is not a string, the *default* value is returned.
- **default** – The default value to return if evaluation failed.

Raises **TypeError** – When the evaluated object does not have the same type as the *default* value. This error can only be raised when *default* is not `None`.

1.2.4 gntools.plans module

Module that facilitates working with GEONIS Generalized Plans (*GP* or *"Planwelt"*).

class gntools.plans.**PlanHelper**(*plan*, *{workspace}*, *{user_prefix}*)
Helper class that returns proper names for *Generalized Plan* (*GP* or *Planwelt*) feature datasets, feature classes and field names. The generated names conform to the GEONIS conventions, i.e. if a plan name contains a number >= 1000, it is assumed that the plan is a custom user plan, which means that it will be prefixed with a *U_*.

Params:

- **plan** (str, unicode):

  GEONIS Generalized Plan (Planwelt) name. For user plans (where numeric part >= 1000), the *U_* prefix can be omitted if the field names should **not** have this prefix.

- **workspace** (str, unicode, gpf.paths.Workspace):

  An optional Esri workspace path (`str`) or a `gpf.paths.Workspace` instance. When specified, the *`get_feature_class()`* and *`get_feature_dataset()`* functions will return full paths instead of just the names.

Keyword params:

- **user_prefix** (str, unicode):

  The prefix for custom user plans (where numeric part >= 1000). Defaults to *U*.

If you initialize the `PlanHelper` with a workspace, full paths will be returned:

```
>>> gph = PlanHelper('pw3', r'C:/temp/test.gdb')
>>> gph.get_feature_dataset('was')
'C:\temp\test.gdb\WAS_PW3'
>>> gph.get_feature_class('sew', 'haltung')
'C:\temp\test.gdb\SEW_PW3\SEW_PW3_HALTUNG'
```

(continues on next page)

(continued from previous page)

```
>>> gph.get_feature_class('sew', 'awk_haltung')  # note: you can include a prefix␣
↪override!
'C:\temp\test.gdb\SEW_PW3\AWK_PW3_HALTUNG'
```

The following example shows how to use a `PlanHelper` for custom user plans (names only):

```
>>> gph = PlanHelper('gp1000')
>>> gph.get_feature_dataset('ELE')
'U_ELE_GP1000'
>>> gph.get_feature_class('ele', 'cable')
'U_ELE_GP1000_CABLE'
```

abbreviation
Returns the Generalized Plan abbreviation (e.g. "GP" or "PW").

Return type str

get_feature_class (*fds_base*, *fc_base*)
Returns a feature class name for the current *Generalized Plan/Planwelt*.

Parameters

- **fds_base** – Case-insensitive base name (GEONIS Solution) of the parent feature dataset, e.g. *ELE*.
- **fc_base** – Case-insensitive base name of the feature class, e.g. *(ELE_)KABEL*. If *fc_base* already contains *fds_base* as a prefix, it will be replaced.

Return type str, unicode

get_feature_dataset (*fds_base*)
Returns a feature dataset name for the current *Generalized Plan/Planwelt*.

Parameters **fds_base** – Case-insensitive base name (GEONIS Solution) of the feature dataset, e.g. *ELE*, *WAS* etc.

Return type str, unicode

number
Returns the Generalized Plan number as a string (e.g. "1", "2", "1001" etc.).

Return type str

prefix
Returns the Generalized Plan user prefix including the separator (underscore), e.g. `U_`.

Return type str

workspace
Returns the Workspace instance for this PlanHelper (specified on initialization).

Rtyoe gpf.paths.Workspace

1.2.5 gntools.protocol module

This module can be used to write text and features to the GEONIS Protocol (XML). The Protocol is typically used to report issues (e.g. validation) with certain features.

class gntools.protocol.**Feature** (*table*, *global_id*, *{geometry}*, *{globalid_field}*)
Bases: object

Creates a Feature for GEONIS Protocol logging purposes.

Params:

- **table** (str, unicode):

  The full path to the table or feature class that contains the logged feature.

- **global_id** (str, unicode, gpf.tools.guids.Guid):

  GlobalID value of the logged feature.

- **geometry** (str, arcpy.Geometry):

  Esri *Geometry* instance or EsriJSON string.

Keyword params:

- **globalid_field** (str, unicode):

  If the *GlobalID* field has a different name, specify it using this option.

fid
Returns the (validated) GlobalID of the current Feature feature.

Return type str

write_elements(*parent_element*)
Adds a <feature> XML element to the *parent_element* based on the current Feature values.

Parameters **parent_element** – Element

class gntools.protocol.**Logger**
Bases: object

Logger class to write GEONIS XML protocols (e.g. for validations, reporting etc.).

Note: Because Logger is a singleton, instantiating it multiple times will always refer to the same object. The Logger stores an XML root element, to which all Logger instances write. Once an instance has called the *flush()* method, the XML root element will be reset. Calling *flush()* will set the project name and path and write out the XML file. After this call, remaining Logger instances or new ones will simply write into a new XML root element.

---

blank()
Logs a blank Entry to the GEONIS XML protocol (appears as a blank line in the protocol window).

error(*message*, *gn_feature=None*)
Logs an error to the GEONIS XML protocol, optionally accompanied by a feature.

Parameters

- **message** (*str, unicode*) – Text message to log.
- **gn_feature** (gntools.protocol.Feature) – Feature object to add to the log entry.

flush(*output_path*, *project_path*, *encoding=None*)
Flushes the root element buffer and writes the GEONIS Protocol to an XML file.

Once this function is called, the root element has been reset and you can reuse the Logger for another project or the same one, or you can exit your application.

Parameters

- **output_path** (*str, unicode*) – The full path to the output protocol XML that should be written.
- **project_path** (*str, unicode*) – The full path to the GEONIS project to which the protocol applies.
- **encoding** (*str, unicode*) – Optional encoding to use for the protocol file (default = ISO-8859-1).

Warning: The user must have write access in the specified output directory.

header(*message*, *gn_feature=None*)
Logs a header to the GEONIS XML protocol, optionally accompanied by a feature.

Parameters

- **message** (*str, unicode*) – Text message to log.
- **gn_feature** (gntools.protocol.Feature) – Feature object to add to the log entry.

info(*message*, *gn_feature=None*)
Logs an info notice to the GEONIS XML protocol, optionally accompanied by a feature.

Parameters

- **message** (*str, unicode*) – Text message to log.
- **gn_feature** (gntools.protocol.Feature) – Feature object to add to the log entry.

message(*message*, *gn_feature=None*)
Logs a basic message to the GEONIS XML protocol, optionally accompanied by a feature.

Parameters

- **message** (*str, unicode*) – Text message to log.
- **gn_feature** (gntools.protocol.Feature) – Feature object to add to the log entry.

subheader(*message*, *gn_feature=None*)
Logs a subheader to the GEONIS XML protocol, optionally accompanied by a feature.

Parameters

- **message** (*str, unicode*) – Text message to log.
- **gn_feature** (gntools.protocol.Feature) – Feature object to add to the log entry.

warn(*message*, *gn_feature=None*)
Logs a warning to the GEONIS XML protocol, optionally accompanied by a feature.

Parameters

- **message** (*str, unicode*) – Text message to log.
- **gn_feature** (gntools.protocol.Feature) – Feature object to add to the log entry.

1.3 Module contents

This is the documentation for the **gntools** package for **Python 2.7** (only). The package provides a toolset for the creation of GEONIS menu or form scripts.

GEONIS is a powerful GIS solution by Geocom Informatik AG (Switzerland), built on top of ArcGIS technology. Although this package does not require a license to run and works fully independent of the GEONIS software, please note that GEONIS itself *does* require a license.

This package is released under the Apache License 2.0 as an open-source product, allowing the community to freely use it, improve it and possibly add new features.

Several functions in this package require Esri's `arcpy` Python library, which does not make this a *free* package. However, users who have already installed and authorized ArcGIS Desktop (ArcMap, ArcCatalog etc.) should be able to work with this package without any problems.

One of the most notable classes is the *gntools.protocol.Logger* class, which can be used to write messages and features to a GEONIS XML log file (or *Protocol*). Users can read this file back in with GEONIS and click on the listed features, which is useful for inspection and/or validation purposes.

Also of interest might be the `gntools.params` module. This module contains classes that help parse passed-in arguments from GEONIS menu or form scripts in a standardized and user-friendly manner.

CHAPTER 2

Indices and tables

- genindex
- modindex

Python Module Index

g

Index