

# Scripting

- [Discover Members of a Function/Class](#)
- [DocStrings](#)
- [Diffable JSON Strings](#)
- [Determine Ignition Edition Installed](#)

# Discover Members of a Function/Class

If you ever need to find out what members a function or class has, you can use the following trick (this example uses the "json" class):

```
# New unknown script package I have never used before
import json

# Find out all members in this package
for item in dir(json):
    print item
```

Which will output the following:

```
JSONDecoder
JSONEncoder
__all__
__author__
__builtins__
__doc__
__file__
__name__
__package__
__path__
__version__
_default_decoder
_default_encoder
decoder
dump
dumps
encoder
load
loads
scanner
```

You can even print the docstring from a class using the syntax like the following example for "json.dump":

```
print json.dump.__doc__
```

Which should output:

```
"""
Serialize ``obj`` as a JSON formatted stream to ``fp`` (a
    ``.write()``-supporting file-like object).

If ``skipkeys`` is true then ``dict`` keys that are not basic types
(``str``, ``unicode``, ``int``, ``long``, ``float``, ``bool``, ``None``)
will be skipped instead of raising a ``TypeError``.

If ``ensure_ascii`` is true (the default), all non-ASCII characters in the
output are escaped with ``\uXXXX`` sequences, and the result is a ``str``
instance consisting of ASCII characters only. If ``ensure_ascii`` is
``False``, some chunks written to ``fp`` may be ``unicode`` instances.
This usually happens because the input contains unicode strings or the
``encoding`` parameter is used. Unless ``fp.write()`` explicitly
understands ``unicode`` (as in ``codecs.getwriter``) this is likely to
cause an error.

If ``check_circular`` is false, then the circular reference check
for container types will be skipped and a circular reference will
result in an ``OverflowError`` (or worse).

If ``allow_nan`` is false, then it will be a ``ValueError`` to
serialize out of range ``float`` values (``nan``, ``inf``, ``-inf``)
in strict compliance of the JSON specification, instead of using the
JavaScript equivalents (``NaN``, ``Infinity``, ``-Infinity``).

If ``indent`` is a non-negative integer, then JSON array elements and
object members will be pretty-printed with that indent level. An indent
level of 0 will only insert newlines. ``None`` is the most compact
representation. Since the default item separator is ``', '```, the
output might include trailing whitespace when ``indent`` is specified.
You can use ``separators=(',', ': ')`` to avoid this.
```

If `separators` is an `(item_separator, dict_separator)` tuple then it will be used instead of the default `(' ', ': ')` separators. `(' ', ': ')` is the most compact JSON representation.

`encoding` is the character encoding for str instances, default is UTF-8.

`default(obj)` is a function that should return a serializable version of obj or raise `TypeError`. The default simply raises `TypeError`.

If `*sort_keys*` is `True` (default: `False`), then the output of dictionaries will be sorted by key.

To use a custom `JSONEncoder` subclass (e.g. one that overrides the `.default()` method to serialize additional types), specify it with the `cls` kwarg; otherwise `JSONEncoder` is used.

"""

Source: <https://forum.inductiveautomation.com/t/dos-and-donts-when-developing-first-project-with-ignition/93592/33>

# DocStrings

Use doc strings when writing scripts for better documentation and to assist auto-complete documentation tooltips. For instance, this example function script:

```
def func1(parms):  
    '''  
        Helpful descriptions to help others understand your code.  
    [Note: will show up in autofill details, so keep it concise.  
  
    [Args:  
    [parms (dict): Small note explaining the param if necessary.  
  
    [Returns:  
    [tuple: Small explanations of the return  
    [  
        ...  
        return parms
```

Will show up in the script editor when using auto-complete like this:

```
func1(  
    parms: dict  
): tuple  
  
Method on agvs  
  
Helpful descriptions to help others understand your code.  
Note: will show up in autofill details, so keep it concise.  
  
Parameters  
    • parms: dict  
        • Small note explaining the param if necessary.  
  
Returns: tuple  
  
Small explanations of the return
```



The following datatypes are supported for arguments and returns:

- bool
- int
- float
- str
- tuple
- list
- dict
- any

Source: <https://forum.inductiveautomation.com/t/dos-and-donts-when-developing-first-project-with-ignition/93592/35>

# Diffable JSON Strings

If you need a JSON object/string to be consistently ordered regardless of how it was built or ordered in memory, the following script can help force it to be organized/ordered alphabetically so that it can be used with source control tools or use diff tools to compare different outputs to one another.

```
# Functions designed to be called via objectScript() in UI expressions.

from java.util import TreeMap
from com.inductiveautomation.ignition.common.util import Comparators
from com.inductiveautomation.ignition.common.gson import GsonBuilder

ciAlnumCmp = Comparators.alphaNumeric(False)

def __ordering(subject, listKey='name'):
    """
    Deep copy with conversion of maps to key-ordered maps and
    conversion of lists-of-dicts that contain a 'name' key into
    ordered lists.
    Keys other than "name" may be supplied, or None to disable
    re-ordering lists of dictionaries.
    """
    if hasattr(subject, 'items'):
        subst = TreeMap(ciAlnumCmp)
        for k, v in subject.items():
            subst[k] = __ordering(v)
        return subst
    if hasattr(subject, '__iter__'):
        # Use a generator to exit quickly if any element of the
        # list-like object is *not* a dictionary-like object.
        if listKey and all(hasattr(inner, 'items') for inner in subject):
            reordered = TreeMap(ciAlnumCmp)
            reordered.update([(x.get(listKey, str(i)), __ordering(x)) for (i, x) in enumerate(subject)])
            return reordered.values()
        return [__ordering(x) for x in subject]
    return subject

def orderedJson(json, useGson = True):
```

```
'''
```

```
    Re-orders a JSON string alphabetically so it can be use with diff tools or other source control tools
```

```
Args:
```

```
    json (str): JSON Dictionary as a string to be re-ordered
```

```
    useGson (bool): Use Google's Gson library to pretty-print the JSON
```

```
Returns:
```

```
    str: Re-ordered JSON string
```

```
    ...
```

```
source = system.util.jsonDecode(json)
```

```
ordered = __ordering(source)
```

```
if useGson:
```

```
    gson = GsonBuilder().setPrettyPrinting().create()
```

```
    return gson.toJson(ordered)
```

```
else:
```

```
    return system.util.jsonEncode(dict(_=ordered), 2)[6:-1]
```

Source slightly tweaked from: <https://forum.inductiveautomation.com/t/json-diffing-discussion/96800/26>



# Determine Ignition Edition Installed

```
from com.inductiveautomation.ignition.common.model import PlatformEdition
print PlatformEdition.isEdge()
# or
# print PlatformEdition.isMaker()
# print PlatformEdition.isCloud()
# print PlatformEdition.isStandard()
```