QDL Initialization (ini) files

QDL support a plain text file format. This allows an application to read a configuration information in a structured way. In essence, this is just an ASCII/text way to write a single stem. Why have them? So that there is a simple, consistent, text-based way to send along configuration and initialization information to an application. No special knowledge of QDL or programming is needed to write an ini file. Also, ini files solve the “bootstrapping problem” of how to start a complex system. The easiest way is to have external information which has to exist out side of the system and should be in an easily editable format.

File syntax

```
[section(.x)*]
line: name =|:= entry (,entry)*

entry: boolean | integer | decimal | scientific number | 'string';

comment: // .* | /* .* */
```

the section and name are identifiers – standard syntax for variable name in QDL applies, as well as simple URNs like **`foo:/bar/baz`**. As well, slashes in names are supported so this is pefectly fine:

```
[books]
urn:isbn:0143039431 := 'The Grapes of Wrath'
```

If the section identifiers have embedded periods, they will be turned into stem entries as well. A line may have multiple entries separated by commas which will be turned into a list. You may have blank lines.

Unlike QDL, each entry is restricted to a single line and does not end with a ; (semi-colon). If you end a line with a semi-colon, it is ignored.

Extra commas are ignored.

Multiple entries on a line are turned in to a list.

A comment is either from a // (double slash) to the end of the line or anything between /* . . . */, which may span multiple lines. You may have these anywhere and in any number. They will be discarded in parsing.

Reading an ini file

You access these in the workspace by issuing a **`file_read`** with the type of 2. The result is a stem of the form

```
stem.section.name0 := line entry
stem.section.name1 := line entry
```

Example

Let's say we have an ini file **`/tmp/sample.ini`**:

```
// last modified on 1 April 2021 by Robespierre Braithwate.
[owner]
name='Fiona Smythe'
organization := 'Big State University/Physics','Big State University/Astronomy'

[database]
# use IP address in case network name resolution is not working
server= '192.168.1.42'
port=1029
```

The resulting stem then

```
  ini. := file_read('/tmp/sample.ini',2);
  ini.

{
 owner: {
  organization:  ['Big State University/Physics','Big State University/Astronomy'],
  name:'Fiona Smythe'
 },
 database: {
  server:'192.168.1.42',
  port:1029
 }
}
```

So if you want to access the information for the owner name, you issue

```
  ini.owner.name
Fiona Smythe
```

Writing ini files from stems

You can simply save a stem as an ini file by using

```
file_write(file_name, contents., 2)
```

where the final argument, 2, tells the system to convert `contents.` Do note that ini files are substantially simpler than a general stem. Values may be scalars or simple lists of scalars and attempts to write something more complex will throw an exception (because there is no good way for a user to enter such a thing without effectively having it all be just QDL).

Example

If you have the following stem (line breaks added)

```
    my_ini. :=
   {'owner':
     {'organization':
        ['Big State University/Physics','Big State University/Astronomy'],
       'name':'Fiona Smythe'
     },
    'database':{'server':'192.168.1.42', 'port':1029}
   };
```

Then you can save it to an ini file, /tmp/my.ini by issuing

```
    file_write('/tmp/my.ini', my_ini., 2);
true
```

When saving stems as ini files, there is no way to put in comments.

Conversions

You may also use the convert module to turn your stems into ini file format with the ini_out (and ini_in for importing). This allows you to, for instance, read in an your favorite format and turn it into YAML, ini file, HOCON or other formats and convert between them.

Example with section stem names

In this example, a more complex topology of the stem is made. The section names have embedded periods, which means the section is turned into its own stem. Note that names within the section with embedded periods are *not* parsed as stem entries, just the section headers.

The ini file

```
// test file for QDL ini format.
 [qwe ]
 q:='test'

a := .456, -47,,, true
// foo
b = 'p',,,'q', 345.66, -3.13E17

[blarf]

z := -234,65.34,'bar'

[woof.foo]
 q := 42
```

```
becomes the stem
{blarf : {z : [-234,65.34,bar]},
   qwe : {a : [0.456,-47,true],
          b : [p,q,345.66,-3.13E17],
          q : test},
  woof : {foo : {q:42}}}
```

This has its uses, but should not be overused since it reduces readability. It certainly does help to indent sub sections, however be advised that the parser is *not* aware of indenting, so

```
[a]
x:=4
  [a.b]
```

```
   y:=5
z:=6
```

would yield

```
{a:{x:4,
    b:{y:5,
       z:6}
   }
}
```

i.e. the final z is still in a.b To get it into the a section, put it first:

```
[a]
x:=4
z:=6
  [a.b]
   y:=5
```

List support

You can simply write out lists, but there is a little syntactic sugar since ini files know what integers and decimals are. Should 234.567 as a section name be a stem or a decimal, for instance. And how to set the value? We don’t want to write something really bad like **`0:=42`** where the left side is an index and the right side is the value. The solution is that names of the form _integer, e.g. _0 are converted to integer entries, so

```
  x.a:=[;5]
  j_use('convert')
true
  ini_out(x.)
[a]
_0 := 0
_1 := 1
_2 := 2
_3 := 3
_4 := 4
```

This works for import as well as export

```
  ini_in(ini_out(x.))
{a:[0,1,2,3,4]}

  ini_out(n(2,3,[-6;0]))
[_0]
_0 := -6
_1 := -5
_2 := -4

[_1]
_0 := -3
_1 := -2
_2 := -1
```

```
  ini_in(ini_out(n(2,3,[-6;0])))
[[-6,-5,-4],[-3,-2,-1]]
```

This is not a format for dragging around your huge matrix (convert the stem to input form and write that to a file), but will allow for reasonable small use cases.

When using file_read and file_write, the default is to support lists (type 2) but if you want to turn that off, you may do so setting the file type to 3. This means that any lists in the wrong place will cause the operation to fail. The convert utility also allows you to turn it on an off as a parameter, but the default is to support lists.

Smuggling in complex objects

Let us say you really wanted to import a complex object in an ini file. E.g. some complex JSON that has been turned into a stem and littering the ini file with its entries would be a mess. The parsing is fairly simple on purpose, since it is a way to mostly splay out a stem with an alternate notation, possibly for people with little to no experience with QDL. This however, makes sending complex object hard, so if you a really need to stash something complex, one way is to use an encoding, then decode and interpret.

E.g.

I want to pass in a stem,

```
   z. := {'a':'b', 'c':'d'}
```

This is a simple example, but it illustrates the point. I can then base 64 encode it as

```
  encode(input_form(z.))
eydhJzonYicsICdjJzonZCd9
```

Which I can put in my ini file:

```
[my]
stem:='eydhJzonYicsICdjJzonZCd9';
```

And in QDL, load it as

```
   ini.:=file_read('/path/to/my/file.ini',2);
   ini.my.stem:=interpret(decode(ini.my.stem))
   ini.my.stem
{
 a:b,
 c:d
}
```