This text defines the format of a qedeq module file for the rule version 1.00.00. Such a file contains ASCII characters and has a lisp like structure. There are two kinds of axioms: integers and strings. Strings are quoted (") that means strings are surrounded by quotes (") and quotes inside the strings are escaped by doubling them. All other types are lists preceded by an function operator. These lists are surrounded by brackets and contain a sequence of elements, separated by commas. Each element is an atom or further operator lists. E.g.:

```
AND(PROP(1), OR(PROP(2), PROP(3)))
```

The following text shows the structure of a qedeq module file in BNF notation.

⟨**MODULE**⟩ start terminal for qedeq module format

```
  ::= 'MODULE('
      ⟨HEADER⟩,          header
      [⟨IMPORTS⟩],       needed other qedeq modules
      [⟨USEDBY⟩],        this qedeq module is used by
      ⟨PARAGRAPHS⟩       paragraphs with axioms, propositions etc.
      ')';
```

⟨**HEADER**⟩ header of a qedeq module

```
  ::= 'HEADER('
      ⟨SPEC⟩,            specification of this module
      ⟨HEADLINE⟩,        title of this module
      ⟨DESCRIPTION⟩,     abstract
      ⟨EMAIL⟩,           email address of module administrator
      ⟨AUTHORS⟩          authors of this module
      ')';
```

⟨**SPEC**⟩ specification of a qedeq module

```
  ::= 'SPEC('
      ⟨NAME⟩,            name of qedeq module
      ⟨VERSION⟩,         version of qedeq module
      ⟨VERSION⟩,         needed rule version
      ⟨LOCATIONS⟩        list of absolute or relative URLs, the specified module could be found, only the path to the module should stand here, the file name is: ⟨NAME⟩_⟨VERSION⟩_⟨VERSION⟩.qedeq
      ')';
```

⟨**NAME**⟩ name

```
  ::= 'NAME('
      ⟨text⟩
      ')';
```

⟨**text**⟩ text atom, an arbitrary string

⟨**VERSION**⟩ version nummer in format nn.nn.nn

```
::= 'VERSION('
    ⟨text⟩
    ')';
```

⟨**LOCATIONS**⟩ list of qedeq module URLs

```
::= 'LOCATIONS('
    ⟨LOCATION⟩,
    {⟨LOCATION⟩}
    ')';
```

at least one URL must be there

⟨**LOCATION**⟩ relative or absolute URL, which points to the directory of an qedeq module file

```
::= 'LOCATION('
    ⟨text⟩
    ')';
```

⟨**HEADLINE**⟩ title of a qedeq module

```
::= 'HEADLINE('
    ⟨text⟩
    ')';
```

⟨**DESCRIPTION**⟩ description of qedeq module ("abstract")

```
::= 'DESCRIPTION('
    ⟨text⟩
    ')';
```

⟨**EMAIL**⟩ email address

```
::= 'EMAIL('
    ⟨text⟩
    ')';
```

⟨**AUTHORS**⟩ author list

```
::= 'AUTHORS('
    ⟨AUTHOR⟩,
    {⟨AUTHOR⟩}
    ')';
```

⟨**AUTHOR**⟩

```
::= 'AUTHOR('
    ⟨text⟩,
    ⟨EMAIL⟩
    ')';
```

author name
email address of author

⟨**IMPORTS**⟩ list of imported modules

```
  ::= 'IMPORTS('
      ⟨IMPORT⟩,
      {⟨IMPORT⟩}
      ')';
```

⟨**IMPORT**⟩ informations about an import module

```
  ::= 'IMPORT('
      ⟨SPEC⟩,          module specification
      ⟨LABEL⟩          alias name for an imported module
      ')';
```

⟨**LABEL**⟩ label for referencing

```
  ::= 'LABEL('
      ⟨text⟩
      ')';
```

⟨**USEDBY**⟩ list of modules, which use this one

```
  ::= 'USEDBY('
      ⟨SPEC⟩,
      {⟨SPEC⟩}
      ')';
```

⟨**PARAGRAPHS**⟩ paragraphs

```
  ::= 'PARAGRAPHS('
      ⟨PARAGRAPH⟩,
      {⟨PARAGRAPH⟩}
      ')';
```

⟨**PARAGRAPH**⟩ paragraph

```
  ::= 'PARAGRAPH('
      ⟨LABEL⟩,              reference anchor
      [⟨text⟩],             LaTeX text
      (
       ⟨ABBREVIATION⟩ |
       ⟨AXIOM⟩ |
       ⟨DECLARERULE⟩ |
       ⟨PROPOSITION⟩
      ),                    or list
      [⟨text⟩]              LaTeX text
      ')';
```

⟨**ABBREVIATION**⟩ definition of an abbreviation

```
  ::= 'ABBREVIATION('
      ⟨FORMULA⟩,            operator to be defined with pattern variables
```

⟨FORMULA⟩ definition for the operator, the same pattern variables as before must occur

')';

⟨**FORMULA**⟩ formula of predicate calculus

(
⟨PROP⟩ | proposition variable
⟨NOT⟩ | negation
⟨AND⟩ | conjunction (logical "and")
⟨OR⟩ | disjunction (logical "or")
⟨IMPL⟩ | implication
⟨EQUI⟩ | logical equivalence
⟨PREDVAR⟩ | predicate variable
⟨FORALL⟩ | universal quantifier
⟨EXISTS⟩ | existential quantifier
⟨FPATTERN⟩ | pattern variable which can stand for any formula (used for abbreviations)
⟨SPATTERN⟩ pattern variable which can stand for any predicate variable (used for abbreviations)
) or list
')';

⟨**PROP**⟩ proposition variable

::= 'PROP('
⟨number⟩ identification number
')';

⟨**number**⟩ numeric atom, an integer

⟨**NOT**⟩ negation

::= 'NOT('
⟨FORMULA⟩ formula to negate
')';

⟨**AND**⟩ conjunction (logical "and")

::= 'AND('
⟨FORMULA⟩, first argument
⟨FORMULA⟩ second argument
')';

⟨**OR**⟩ disjunction (logical "or")

::= 'OR('
⟨FORMULA⟩, first argument
⟨FORMULA⟩ second argument
')';

⟨**IMPL**⟩ logical implication

```
  ::= 'IMPL('
      ⟨FORMULA⟩,        first formula
      ⟨FORMULA⟩         second formula
      ')';
```

⟨**EQUI**⟩ logical equivalence

```
  ::= 'EQUI('
      ⟨FORMULA⟩,        formula
      ⟨FORMULA⟩         formula
      ')';
```

⟨**PREDVAR**⟩ predicate variable

```
  ::= 'PREDVAR('
      ⟨number⟩,         identification number
      ⟨L⟩               list of subject variables
      ')';
```

⟨**L**⟩ list of subject variables

```
  ::= 'L('
      {⟨VAR⟩}
      ')';
```

⟨**VAR**⟩ subject variable

```
  ::= 'VAR('
      ⟨number⟩          identification number
      ')';
```

⟨**FORALL**⟩ universal quantifier

```
  ::= 'FORALL('
      ⟨VAR⟩,            quantify over this subject variable
      ⟨FORMULA⟩         formula which has the above subject variable as a free
                        variable
      ')';
```

⟨**EXISTS**⟩ existential quantifier

```
  ::= 'EXISTS('
      ⟨VAR⟩,            quantify over this subject variable
      ⟨FORMULA⟩         formula which has the above subject variable as a free
                        variable
      ')';
```

⟨**FPATTERN**⟩ formula pattern variable

```
  ::= 'FPATTERN('
```

⟨number⟩	identification number
')';	

⟨**SPATTERN**⟩ subject variable pattern variable

::= 'SPATTERN('	
⟨number⟩	identification number
')';	

⟨**AXIOM**⟩ declaration of an axiom, that is a stressed formula

::= 'AXIOM('	
⟨FORMULA⟩	axiom formula
')';	

⟨**DECLARERULE**⟩ declaration of a new rule

::= 'DECLARERULE('	
⟨text⟩,	rule name
⟨text⟩,	rule description
{⟨LINK⟩}	references to necessary axioms, propositions
')';	

⟨**LINK**⟩ rererences to a label of an axiom, proposition

::= 'LINK('	
⟨text⟩	label name
')';	

⟨**PROPOSITION**⟩ consists of a mathematical theorem and its proof

::= 'PROPOSITION('	
⟨SENTENCE⟩,	theorem
⟨PROOF⟩	proof, last proof formula must be identical to theorem
')';	

⟨**SENTENCE**⟩ mathematical theorem

::= 'SENTENCE('	
⟨FORMULA⟩	true mathematical formula
')';	

⟨**PROOF**⟩ proof of a mathematical theorem, consists sequent proof lines, each one is constructed by using logical rules by foregoing proof lines, axioms, abbreviations or propositions

::= 'PROOF('	
⟨LINE⟩,	proof lines
{⟨LINE⟩}	
')';	

⟨**LINE**⟩ single proof line, consists of a derived formula and an information about the used rule and necessary references that enable the derivation

```
::= 'LINE('
     ⟨FORMULA⟩,
     (
      ⟨ADDAXIOM⟩ |                simple addition of an axiom to the prooflines
      ⟨ADDSENTENCE⟩ |             simple addition of an already proved proposition
      ⟨MODUSPONENS⟩ |             execution of Modus Ponens
      ⟨REPLACEPROP⟩ |             replacement of an propositional variable
      ⟨USEABBREVIATION⟩ |         usage of an abbreviation
      ⟨REVERSEABBREVIATION⟩       reverse an abbreviation
    |
      ⟨RENAMEFREEVARIABLE⟩ |      rename a free subject variable
      ⟨RENAMEBOUNDVARIABLE⟩       rename a bound subject variable
    |
      ⟨REPLACEPREDICATE⟩ |        replace a predicate variable by a formula
      ⟨GENERALIZATION⟩ |          execution of generalization
      ⟨PARTICULARIZATION⟩         execution of particularization
     )                            or list
     ')';
```

⟨**ADDAXIOM**⟩ add an axiom

```
::= 'ADDAXIOM('
     ⟨LINK⟩                      axiom reference
     ')';
```

⟨**ADDSENTENCE**⟩ add an already proven proposition

```
::= 'ADDSENTENCE('
     ⟨LINK⟩
     ')';
```

⟨**MODUSPONENS**⟩ Modus Ponens

```
::= 'MODUSPONENS('
     ⟨number⟩,                   proof line number which references formula A
     ⟨number⟩                    proof line number which references formula A -¿ B
     ')';
```

⟨**REPLACEPROP**⟩ propositional variable replacement

```
::= 'REPLACEPROP('
     ⟨number⟩,                   proof line number at which the rule should be executed
     ⟨PROP⟩,                     this propositional variable should be replaced
     ⟨FORMULA⟩                   replacement
     ')';
```

⟨**USEABBREVIATION**⟩ use definition of an abbreviation

::= 'USEABBREVIATION('
⟨number⟩, proof line number at which the rule should be executed
⟨LINK⟩, abbreviation reference
⟨number⟩ this occurrence number of the abbreviation shall be transformed
')';

⟨**REVERSEABBREVIATION**⟩ reverse definition of an abbreviation
::= 'REVERSEABBREVIATION('
⟨number⟩, proof line number at which the rule should be executed
⟨LINK⟩, abbreviation reference
⟨number⟩ this occurrence number of the abbreviation pattern shall be replaced by the abbreviation
')';

⟨**RENAMEFREEVARIABLE**⟩ rename of a free subject variable
::= 'RENAMEFREEVARIABLE('
⟨number⟩, proof line number at which the rule should be executed
⟨VAR⟩, rename this free subject variable
⟨VAR⟩ with this (non bound) subject variable
')';

⟨**RENAMEBOUNDVARIABLE**⟩ rename of a bound subject variable (at a specific location)
::= 'RENAMEBOUNDVARIABLE('
⟨number⟩, proof line number at which the rule should be executed
⟨VAR⟩, rename this bound subject variable
⟨VAR⟩, with this (non free) subject variable, which thereby must not get bound a second time
⟨number⟩ this occurrence of an quantor with the designated subject variable shall be target of the operation
')';

⟨**REPLACEPREDICATE**⟩ replace predicate by formula
::= 'REPLACEPREDICATE('
⟨number⟩, proof line number at which the rule should be executed
⟨PREDVAR⟩, with pairwise different pattern variables as arguments
⟨FORMULA⟩ formular which contains the same pattern variables, the set of free subject variables must be disjunct to the set of bounded subject variables of the referenced proof line and conversely (the set of bounded subject variables must be disjunct to the set of free subject variables of the referenced proof line) and the predicate variable must not occur in the sphere of action of an quantor with an associated subject variable that is also contained in this formula
')';

⟨**GENERALIZATION**⟩ generalization rule

::= 'GENERALIZATION('

⟨number⟩, proof line number at which the rule should be executed, must have the form A -¿ B(x), with x not contained in A

⟨VAR⟩ this subject variable is generalized

')';

⟨**PARTICULARIZATION**⟩ particularization rule

::= 'PARTICULARIZATION('

⟨number⟩, proof line number at which the rule should be executed, must have the form A(x) -¿ B, with x not contained in B

⟨VAR⟩ this subject variable is particularized

')';