We define a new system variable for our settings:

```latex
\definesystemvariable{bnf}
```

We need some constants for the multi-lingual interface,

```latex
\startconstants
  \english english \dutch dutch
  terminalstart: terminalstart terminalstart
  terminalstop: terminalstop terminalstop
  nonterminalstart: nonterminalstart nonterminalstart
  nonterminalstop: nonterminalstop nonterminalstop
  is: is worden
\stopconstants
```

and while we’re at it, let’s define some variables.

```latex
\startvariables
  \english english \dutch dutch
  bnfgrammar: bnfgrammar bnfspraakleer
  bnfgrammars: bnfgrammars bnfspraakleer
\stopvariables
```

Finally, we want the commands to be multi-lingually accessible, so we set that up as well:

```latex
\startcommands
  \english english \dutch dutch
  setupbnfgrammar: setupbnfgrammar stelbnfspraakleer
  startbnfgrammar: startbnfgrammar startbnfspraakleer
  stopbnfgrammar: stopbnfgrammar startbnfspraakleer
\stopcommands
```

Now to the interesting parts, those that are actually useful to the outside world. First we have the `\startbnfgrammar` and `\stopbnfgrammar` pairs, which are of course used to delimit BNF grammars.

We would like to define `\startbnfgrammar` as `\def\startbnfgrammar[#1]`, but a bug in CONTEXT prevents us from doing this, as the first character in the grammar may be active, for example `<`, but while checking for the presence of `[`, it gets ruined. A way around it is of course to require that the user pass an empty `[` pair, and we will use this method at the moment.

```latex
\def\complexstartbnfgrammar\[#1\]%
  {\endgraf\nobreak\medskip
    \begingroup
    \setupbnfgrammar[#1]%
    \chardef\bnfsinglequote=’
    \defineactivecharacter : {\@@bnfis}
    \defineactivecharacter | {\@@bnfoption}
    \defineactivecharacter " %
    {\thinspace\bgroup\@@bnfterminalstart\setupinlineverbatim%}
    \defineactivecharacter " {\@@bnfterminalstop\egroup\thinspace
    %}
    {\thinspace\bgroup\@@bnfterminalstart\setupinlineverbatim%}
    \defineactivecharacter ’ {\@@bnfterminalstop\egroup\thinspace%}
    %}
    \catcode’=13
    \let\par=\bnfgrammarline
    \obeylines}
```
We need a couple more macros to deal with the interior of a BNF grammar. \(<\) is used for non--terminals, and \(\text{bnfgrammarrule}\) is used later on in \(\text{bnfgrammarswitch}\) for continuing a line.

These macros deal with the ending of a line in a grammar. \(\text{bnfgrammarline}\) is called whenever a new line begins, and invokes \(\text{bnfgrammarswitch}\) to determine what to do next. If the next token is \(<\), we will call upon \(\text{bnfgrammarrule}\) to deal with the new rule. If it is \(\text{stopbnfgrammar}\), we end the top--level group, and let it process \(\text{stopbnfgrammar}\) afterwards. Otherwise we invoke \(\text{bnfgrammarcont}\), which will end the line and add some indentation to the continuing line.

We also define a useful abbreviation to be used for header texts and labels.
And we use it here:

Finally we define a float to be use with BNF grammars, so that we can finish off with something like this:

\placebnfgrammar
  {} {}  
  {An example of a placed grammar.}
  {\startbnfgrammar[]}
  \placebnfgrammar

\definefloat
  [\v!bnfgrammars]
  [\v!bnfgrammar]

\protect \endinput
Grammars

\(<>\) 2 \quad \text{\bnfgrammarswitch} 2
\quad \text{\setupbnfgrammar} 2
\quad \text{\startbnfgrammar} 1
\quad \text{\stopbnfgrammar} 1
\quad \text{\setupbnfgrammar} 2
\quad \text{\startbnfgrammar} 2
\quad \text{\stopbnfgrammar} 1
\quad \text{\setupbnfgrammar} 2
\quad \text{\startbnfgrammar} 1
\quad \text{\stopbnfgrammar} 1