JEquation

A typical equation looks like:

```
<key> = <value>
```

To be more specific, an equation consists of:

- left operand or 'key';
- assignment marker, usually '=';
- right operand or 'value'; and
- end-of-line marker, usually '\n'.

The JEquation class can be used to parse equations in a general way. It comprises the following data members:

type	member
JString	key
char	sep
JString	value

These data members can be accessed via member methods getKey(), getSeparator() and getValue(), respectively.

The various markers needed to parse an equation are defined in the auxiliary class JEquationParameters. The list of data members of this class includes:

type	member	default	comment
std::string	sep	"="	assignment
std::string	eol	"\n\r;"	end-of-line
std::string	div	"./"	division
std::string	skip	"#"	skip line
char	left	'('	left bracket
char	right	')'	right bracket
std::string	ws	" \t\n\v\f\r"	white space

The values of the various data members can be defined at construction or via calls to the corresponding get and set methods. The data members sep, eol, div, skip, and ws consists of a string of characters of which each individually applies to its designated function. The characters contained in sep separates a key from its value; those contained in eol determine the end of a line; and those contained in div separate different tokens within a composite key. If the first character of a read line is contained in skip, the whole line is ignored and the reading continues. Any character in ws is considered a white space; it will not contribute to the reading of a key or value. The characters left and right refer to a possible list of values which should together be treated. This makes it possible to parse equations that are contained within a value.

The parameters to parse an equation can be passed to the STL I/O operators using the class JEquationFacet, e.g:

```
in.imbue(locale(in.getloc(), new JEquationFacet(JEquationparameters(..))));
```

where `in` is an `std::istream` object which 'owns' the facet (the newly created facet will be deleted by the owner at the appropriate time). Following this statement, the given `std::istream` object knows how to parse an equation.

To read equations one-by-one, the following for-loop can be employed.

```
const JEquationFacet facet(JEquationParameters("=", ";\n", "./", "#"));

in.imbue(locale(in.getloc(), facet.clone()));

for (JEquation equation; in >> equation; ) {
}
```

As a result, each equation is defined by a single line ending with a '`\n`' or '`;`' but not starting with a '`#`'. To select a genuine equation, the following condition should be checked within the for-loop.

```
if (facet.isSeparator(equation.getSeparator())) {
  // some action
}
```

One can recursively subdivide a possible composite key as follows.

```
for ( ; facet.isDivision(equation.getSeparator()); ++i) {
  equation.setEquation(facet);
}
```

The actual value of an equation is of type `JString`, so that further parsing can be done using the various member methods of this class or those of its base class `std::string`. Of course, the value can also be parsed using the `std::istringstream` class. It is interesting to note that the classes `JProperties` and `JRootStreamer` make use of the `JEquation` class to parse equations. The first can be used to set up a dictionary for ASCII I/O yourself and the latter uses the ROOT dictionary generated by `rootcint` for ASCII I/O of a data structure.