dataio Reference Manual

Generated by Doxygen 1.2.8.1

Thu Sep 6 13:45:40 2001

Contents

Chapter 1

dataio Module Index

1.1 dataio Modules

Here is a list of all modules:

Chapter 2

dataio Hierarchical Index

2.1 dataio Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

dataio Compound Index

3.1 dataio Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

dataio File Index

4.1 dataio File List

Here is a list of all £les with brief descriptions:

Chapter 5

dataio Page Index

5.1 dataio Related Pages

Here is a list of all related documentation pages:

Chapter 6

dataio Module Documentation

6.1 dataio

Compounds

- class _dataiorec

  io record template class used by dataio.

- class _dataiorecbase

  io record base class used by ndataio class. Internal use only.

- class dataio

  dataio main class to perform data input/output features The primitive data that will read/write are: bool, char, wchar_t (wchar_t work? perphaps, no), string, int, long, unsigned, unsigned long, ¤oat, double, long double.

De£nes

- #de£ne _DATAIODEFAULTCOLUMNSEPARATOR '\t'

  default value of column separator (tab separated).

- #de£ne _DATAIODEFAULTAUTOLINESEPARATOR '\0'

  default value of line separator that assume standard text £le that use one of folowwing new line conventions: 'n' (unix), rn' (ansi - DOS, windows), 'r' (VAX,VMS), 'rn' (unknow (there exist?).

- #de£ne _DATAIODEFAULTLINESEPARATOR _DATAIODEFAULTAUTOLINESEPARATOR

  default value of line separator (standard text £le).

- #de£ne _DATAIODEFAULTATTRIBSEPARATOR '\0'

  default value for con£gattribseparator (disabled) if need to read con£g £le, active this as '=' (this is the con£guration for most applications) Observe that the activation of attrib separator do not disable the use of column separator to separate variable and data.

- #de£ne _DATAIODEFAULTDECIMAL '.'

  default value of decimal charecter (as C/C++ locale).

- #de£ne _DATAIODEFAULTIGNORECASE false

  default value of ignorere case frag (case sensive).

- #de£ne _DATAIODEFAULTCOMMENTLINE "//"

  default value of comment line marker (C++ line comment).

- #de£ne _DATAIODEFAULTCOMMENTOPEN "/∗"

  default value of comment block open delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTCOMMENTCLOSE "∗/"

  default value of comment block close delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTSTDCOMMENTOPEN "/∗"

  default value of standard comment block open delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTSTDCOMMENTCLOSE "∗/"

  default value of comment block close delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTLINEWRAP ""

  default value of line wrap (append next line) marker (disabled).

- #de£ne _DATAIODEFAULTMAXCOLUMNONLINE (∼0U)

  default value of maximun column on the line. It column exceds this and line wrap is actived (!linewrap.empty()), operator>> perform line wrapping.

- #de£ne _DATAIODEFAULTCOLUMNORIENTED false

  default value of column oriented data ¤ag (is not column oriented).

- #de£ne _DATAIODEFAULTISTABLE false

  default value of istable data ¤ag (is not the table data).

- #de£ne _DATAIODEFAULTEMPTYISVALID true

  default value of emptyisvalid data ¤ag (empty is valid value, if suitable).

- #de£ne _DATAIODEFAULTEXTENDEDMODE true

  default value of extendedmode data ¤ag (work in extended mode).

- #de£ne _DATAIODEFAULTVALIDATEALL true

  default value of validateall() (the unrefered variables is validated).

- #de£ne _DATAIODEFAULTCLEAREMPTYTAIL true

  default value for clearemptytail.

- #de£ne _DATAIODEFAULTSTRINGDELIMITER '\"'

  default value of stringdelimiter (C/C++ java mode delimiter).

- #de£ne _DATAIODEFAULTSCAPECHAR '\0'

  default value of scapechar (disabled: for C/C++ java mode char scape sequence, use '').

- #de£ne _DATAIODEFAULTTHROWEXCEPTION false

  default value of throwexception (disabled).

- #de£ne _DATAIODEFAULTPRINTERROR true

  default value of printerror (enabled).

- #de£ne _DATAIODEFAULTCOLLECTNAMES true

  default value for refersubnames() (collect refered names).

- #de£ne _DATAIODEFAULTPARSEALLINPUTSTRING false

  default values for parseallinputstring.

- #de£ne _DATAIODEFAULTSECTIONNAMEOPEN '\0'

  Default value for sectionameopen (disabled: for windows ini £le like, use '[').

- #de£ne _DATAIODEFAULTSECTIONNAMECLOSE '\0'

  Default value for sectionameclose (disabled: for windows ini £le like, use ']').

- #de£ne _DATAIODEFAULTSTDSECTIONNAMEOPEN '['

  Default value for sectionameopen for default section mode (windows ini line).

- #de£ne _DATAIODEFAULTSTDSECTIONNAMECLOSE ']'

  Default value for sectionameclose for default section mode (windows ini £le like).

- #de£ne _DATAIODEFAULTREVERSEBYTEORDER false

  for binary input/output usage default values for reversebyteorder.

- #de£ne _DATAIODEFAULTYESLIST {"true", "on", "yes", 0}

  used by bool type input: the item on list is assumed true Note: the empty value is assumed false.

- #de£ne _DATAIODEFAULTNOLIST {"false", "off", "no", 0}

  used by bool type input: the item on list is assumed false Note: if velue is not on yes list and on false list, is assumed false.

- #de£ne _DATAIODEFAULTYESVALUE "yes"

  defalt word for true value on bool type.

- #de£ne _DATAIODEFAULTNOVALUE "no"

  defalt word for false value on bool type.

6.1.1 De£ne Documentation

6.1.1.1 #de£ne _DATAIODEFAULTATTRIBSEPARATOR '\0'

default value for con£gattribseparator (disabled) if need to read con£g £le, active this as '=' (this is the con£guration for most applications) Observe that the activation of attrib separator do not disable the use of column separator to separate variable and data.

De£nition at line 78 of £le dataio.h.

6.1.1.2 #de£ne _DATAIODEFAULTAUTOLINESEPARATOR '\0'

default value of line separator that assume standard text £le that use one of folowwing new line conventions: 'n' (unix), rn' (ansi - DOS, windows), 'r' (VAX,VMS), 'rn' (unknow (there exist?).

\\\\\\

De£nition at line 70 of £le dataio.h.

6.1.1.3 #de£ne _DATAIODEFAULTCLEAREMPTYTAIL true

default value for clearemptytail.

De£nition at line 118 of £le dataio.h.

6.1.1.4 #de£ne _DATAIODEFAULTCOLLECTNAMES true

default value for refersubnames() (collect refered names).

De£nition at line 132 of £le dataio.h.

6.1.1.5 #de£ne _DATAIODEFAULTCOLUMNORIENTED false

default value of column oriented data ¤ag (is not column oriented).

De£nition at line 102 of £le dataio.h.

6.1.1.6 #de£ne _DATAIODEFAULTCOLUMNSEPARATOR '\t'

default value of column separator (tab separated).

De£nition at line 65 of £le dataio.h.

6.1.1.7 #de£ne _DATAIODEFAULTCOMMENTCLOSE "*/"

default value of comment block close delimiter (C/C++/java comment block marker).

De£nition at line 88 of £le dataio.h.

6.1.1.8 #de£ne _DATAIODEFAULTCOMMENTLINE "//"

default value of comment line marker (C++ line comment).

De£nition at line 84 of £le dataio.h.

6.1.1.9 #de£ne _DATAIODEFAULTCOMMENTOPEN "/∗"

default value of comment block open delimiter (C/C++/java comment block marker).

De£nition at line 86 of £le dataio.h.

6.1.1.10 #de£ne _DATAIODEFAULTDECIMAL '.'

default value of decimal charecter (as C/C++ locale).

De£nition at line 80 of £le dataio.h.

6.1.1.11 #de£ne _DATAIODEFAULTEMPTYISVALID true

default value of emptyisvalid data ¤ag (empty is valid value, if suitable).

De£nition at line 106 of £le dataio.h.

6.1.1.12 #de£ne _DATAIODEFAULTEXTENDEDMODE true

default value of extendedmode data ¤ag (work in extended mode).

De£nition at line 108 of £le dataio.h.

6.1.1.13 #de£ne _DATAIODEFAULTIGNORECASE false

default value of ignorere case frag (case sensive).

De£nition at line 82 of £le dataio.h.

6.1.1.14 #de£ne _DATAIODEFAULTISTABLE false

default value of istable data ¤ag (is not the table data).

De£nition at line 104 of £le dataio.h.

6.1.1.15 #de£ne _DATAIODEFAULTLINESEPARATOR _-DATAIODEFAULTAUTOLINESEPARATOR

default value of line separator (standard text £le).

De£nition at line 72 of £le dataio.h.

6.1.1.16 #de£ne _DATAIODEFAULTLINEWRAP ""

default value of line wrap (append next line) marker (disabled).

De£nition at line 96 of £le dataio.h.

6.1.1.17 #de£ne _DATAIODEFAULTMAXCOLUMNONLINE (~0U)

default value of maximun column on the line. It column exceds this and line wrap is actived (!linewrap.empty()), operator>>() perform line wrapping.

De£nition at line 100 of £le dataio.h.

6.1.1.18 #de£ne _DATAIODEFAULTNOLIST {"false", "off", "no", 0}

used by bool type input: the item on list is assumed false Note: if velue is not on yes list and on false list, is assumed false.

De£nition at line 156 of £le dataio.h.

6.1.1.19 #de£ne _DATAIODEFAULTNOVALUE "no"

defalt word for false value on bool type.

De£nition at line 160 of £le dataio.h.

6.1.1.20 #de£ne _DATAIODEFAULTPARSEALLINPUTSTRING false

default values for parseallinputstring.

De£nition at line 134 of £le dataio.h.

6.1.1.21 #de£ne _DATAIODEFAULTPRINTERROR true

default value of printerror (enabled).

De£nition at line 127 of £le dataio.h.

6.1.1.22 #de£ne _DATAIODEFAULTREVERSEBYTEORDER false

for binary input/output usage default values for reversebyteorder.

De£nition at line 148 of £le dataio.h.

6.1.1.23 #de£ne _DATAIODEFAULTSCAPECHAR '\0'

default value of scapechar (disabled: for C/C++ java mode char scape sequence, use ").

\

De£nition at line 122 of £le dataio.h.

6.1.1.24 #de£ne _DATAIODEFAULTSECTIONNAMECLOSE '\0'

Default value for sectionameclose (disabled: for windows ini £le like, use ']').

De£nition at line 139 of £le dataio.h.

6.1.1.25 #de£ne _DATAIODEFAULTSECTIONNAMEOPEN '\0'

Default value for sectionameopen (disabled: for windows ini £le like, use '[').

De£nition at line 137 of £le dataio.h.

6.1.1.26 #de£ne _DATAIODEFAULTSTDCOMMENTCLOSE "*/"

default value of comment block close delimiter (C/C++/java comment block marker).

De£nition at line 93 of £le dataio.h.

6.1.1.27 #de£ne _DATAIODEFAULTSTDCOMMENTOPEN "/*"

default value of standard comment block open delimiter (C/C++/java comment block marker).

De£nition at line 91 of £le dataio.h.

6.1.1.28 #de£ne _DATAIODEFAULTSTDSECTIONNAMECLOSE ']'

Default value for sectionameclose for default section mode (windows ini £le like).

De£nition at line 144 of £le dataio.h.

6.1.1.29 #de£ne _DATAIODEFAULTSTDSECTIONNAMEOPEN '['

Default value for sectionameopen for default section mode (windows ini line).

De£nition at line 142 of £le dataio.h.

6.1.1.30 #de£ne _DATAIODEFAULTSTRINGDELIMITER '\"'

default value of stringdelimiter() (C/C++ java mode delimiter).

De£nition at line 120 of £le dataio.h.

6.1.1.31 #de£ne _DATAIODEFAULTTHROWEXCEPTION false

default value of throwexception (disabled).

De£nition at line 125 of £le dataio.h.

6.1.1.32 #de£ne _DATAIODEFAULTVALIDATEALL true

default value of validateall() (the unrefered variables is validated).

De£nition at line 115 of £le dataio.h.

6.1.1.33 #de£ne _DATAIODEFAULTYESLIST {"true", "on", "yes", 0}

used by bool type input: the item on list is assumed true Note: the empty value is assumed false.

De£nition at line 153 of £le dataio.h.

6.1.1.34 #de£ne _DATAIODEFAULTYESVALUE ”yes”

defalt word for true value on bool type.

De£nition at line 158 of £le dataio.h.

6.2 stringutil

De£nes

- #de£ne _STRINGUTILDEFAULTPRINTERROR true

  default values for stringutilprinterror
  See also:
  stringutilprinterror.

- #de£ne _STRINGUTILDEFAULTTHROWEXCEPTION false

  defaultvalues for stringutilthrowexception
  See also:
  stringutilthrowexception.

- #de£ne _STRINGUTILSTRDELIMITER '\"'

  default string delimiter char speci£cation.

- #de£ne _STRINGUTILSCAPECHAR '\\'

  default scape char speci£cation.

- #de£ne _STRINGUTILCOMMENTCHAR '#'

  default comment char speci£cation.

- #de£ne _STRINGUTILCOLUMNSEPARATOR '\t'

  default column separator.

- #de£ne _STRINGUTILCNTRLMARK '^'

  speci£cator for cntrl char.

- #de£ne _STRINGUTILHEXAMARK 'X'

  speci£cator for hexadecimal.

- #de£ne _STRINGUTILCHARTABLE

  the spceial characters speci£cation table.

Functions

- bool stringutilprinterror ()

  print error message default value is _STRINGUTILDEFAULTPRINTERROR;
  See also:
  stringutilthrowexception, _STRINGUTILDEFAULTPRINTERROR.

- bool stringutilprinterror (bool status)
- bool stringutilthrowexception ()

  throw exception default value is _STRINGUTILDEFAULTTHROWEXCEPTION

See also:
stringutilprinterror, _STRINGUTILDEFAULTPRINTERROR.

- bool stringutilthrowexception (bool status)
- bool isextended (char c)
- bool iswhite (char c)

  white char detection, used by string util and dataio.

- char tocntrl (char c)
- unsigned stringcase£nd (string const &s1, string const &s2, unsigned pos=0)

  case insensive version of string::£nd().

- unsigned stringcase£nd_last_of (string const &s1, string const &s2, unsigned pos=0)

  case insensive version of string::£nd_last_of() does not used by dataio.

- string& parsestring (string &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  clear extra spaces and parse delimited string of s the sequence of iswhite() is replaced by single space, except inside of delimited string (inside delimiters, the sequence of two delimiters is assumed one delimiters).

- vector<string>& parsestring (vector< string > &strlist, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  for each element of list, clear extra spaces and parse delimited string.

- string& reverseparsestring (string &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  translate s to delimited string (is delimiter is found, substitute by double of one).

- vector<string>& reverseparsestring (vector< string > &strlist, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  for each element of list, convert to delimited string.

- unsigned £ndstringdelimiterclose (string const &s, unsigned pos=0, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  return the position of string delimiter to close, started in the position pos. case delimiter='\0', force that delimiter is s[pos] otherwise: if s[pos] is not delimiter, assume that £rst delimiter is to open, and start closer £ndding. return ~0U if not found. return s.size() if delimiter do not occur in the s (starting the pos) CAUTION: This routine will change in future in way to support C/C++ like cher speci£cation.

- bool stringdelimiterbalanced (string const &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, unsigned pos1=0, unsigned pos2=(~0U))

  return true, if string delimiter is closed correctly between [pos1, pos2).

- bool needdelimiter (string const &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  return true, if string delimiters is required if contain two adjacent iswhite() or string delimiters.

- istream& gettextline (istream &f, string &line, char comment=_STRINGUTILCOMMENTCHAR, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, char newlinedelimiter='\0')

getline() that can read text £le of ansi, DOS, windows (newline == '\r
'), UNIX (newline=='n') and VAX/VMS (newline=='r') withouth speci£cation. if (newline == '\0'), auto detect new line format. otherwise, work as getline() Unless (newline == EOF), assume end of £le if EOF character is found or £sical eof occur It is necessary, because some text editor write EOF character at end of £le If (newline == EOF), read EOF char into memory, if found.

- istream& getline (istream &f, vector< string > &dataline, char separator=_STRINGUTILCOLUMNSEPARATOR, char comment=_STRINGUTILCOMMENTCHAR, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, char newline='\0')

  vectorized getline get vector <string> (one line separated as colmns) from istream does not used by dataio newline=='\0' is autodetect mode.

- istream& getlines (istream &f, vector< vector< string > > &datalist, char separator=_STRINGUTILCOLUMNSEPARATOR, char comment=_STRINGUTILCOMMENTCHAR, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, char newline='\0')

  vectorized getlines newline=='\0' is auto detect mode.

- ostream& putline (ostream &f, vector< string > &dataline, char separator=_STRINGUTILCOLUMNSEPARATOR, char newline='\0')

  put the vector<string> to ostream dataio use it, only if _DATAIODEBUG_ is de£ned newline=='\0' is system newline.

- ostream& putlines (ostream &f, vector< vector< string > > &datalist, char separator=_STRINGUTILCOLUMNSEPARATOR, char newline='\0')

  vectorized putlines newline=='\0' is system new lines.

- vector<vector <string> >& transpose (vector< vector< string > > &data)

  transpose the vector<vector <string> > data array (replace row and column).

6.2.1 De£ne Documentation

6.2.1.1 #de£ne _STRINGUTILCHARTABLE

Value:

```
    { \
    \
    \
    \
{'a', '\a'},    \
{'b', '\b'},    \
{'e', '\e'},     \
{'f', '\f'},     \
{'n', '\n'},    \
{'r', '\r'},     \
{'t', '\t'},    \
{'v', '\v'},    \
{'\0', '\0'},   \
}
```

the spceial characters speci£cation table.

De£nition at line 107 of £le stringutil.h.

6.2.1.2 #de£ne _STRINGUTILCNTRLMARK '^'

speci£cator for cntrl char.

De£nition at line 102 of £le stringutil.h.

6.2.1.3 #de£ne _STRINGUTILCOLUMNSEPARATOR '\t'

default column separator.

De£nition at line 100 of £le stringutil.h.

6.2.1.4 #de£ne _STRINGUTILCOMMENTCHAR '#'

default comment char speci£cation.

De£nition at line 98 of £le stringutil.h.

6.2.1.5 #de£ne _STRINGUTILDEFAULTPRINTERROR true

default values for stringutilprinterror

See also:
stringutilprinterror.

De£nition at line 88 of £le stringutil.h.

6.2.1.6 #de£ne _STRINGUTILDEFAULTTHROWEXCEPTION false

defaultvalues for stringutilthrowexception

See also:
stringutilthrowexception.

De£nition at line 91 of £le stringutil.h.

6.2.1.7 #de£ne _STRINGUTILHEXAMARK 'X'

speci£cator for hexadecimal.

De£nition at line 104 of £le stringutil.h.

6.2.1.8 #de£ne _STRINGUTILSCAPECHAR '\\'

default scape char speci£cation.

De£nition at line 96 of £le stringutil.h.

6.2.1.9 #de£ne _STRINGUTILSTRDELIMITER '\"'

default string delimiter char speci£cation.

De£nition at line 94 of £le stringutil.h.

6.2.2 Function Documentation

6.2.2.1 unsigned £ndstringdelimiterclose (string const & *s*, unsigned *pos* = 0, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR)

return the position of string delimiter to close, started in the position pos. case delimiter='\0', force that delimiter is s[pos] otherwise: if s[pos] is not delimiter, assume that £rst delimiter is to open, and start closer £ndding. return ~0U if not found. return s.size() if delimiter do not occur in the s (starting the pos) CAUTION: This routine will change in future in way to support C/C++ like cher speci£cation.

6.2.2.2 istream & getline (istream & *f*, vector< string > & *dataline*, char *separator* = _STRINGUTILCOLUMNSEPARATOR, char *comment* = _STRINGUTILCOMMENTCHAR, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR, char *newline* = '\0')

vectorized getline get vector <string> (one line separated as colmns) from istream does not used by dataio newline=='\0' is autodetect mode.

6.2.2.3 istream & getlines (istream & *f*, vector< vector< string > > & *datalist*, char *separator* = _STRINGUTILCOLUMNSEPARATOR, char *comment* = _STRINGUTILCOMMENTCHAR, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR, char *newline* = '\0')

vectorized getlines newline=='\0' is auto detect mode.

Examples:
samptablewithoutdataio.cpp.

6.2.2.4 istream & gettextline (istream & *f*, string & *line*, char *comment* = _STRINGUTILCOMMENTCHAR, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR, char *newlinedelimiter* = '\0')

getline() that can read text £le of ansi, DOS, windows (newline == '\r

'), UNIX (newline=='n') and VAX/VMS (newline=='r') withouth speci£cation. if (newline == '\0'), auto detect new line format. otherwise, work as getline() Unless (newline == EOF), assume end of £le if EOF character is found or £sical eof occur It is necessary, because some text editor write EOF character at end of £le If (newline == EOF), read EOF char into memory, if found.

\\

6.2.2.5 bool isextended (char *c*) `[inline]`

De£nition at line 141 of £le stringutil.h.

Referenced by iswhite().

6.2.2.6 bool iswhite (char *c*) `[inline]`

white char detection, used by string util and dataio.

De£nition at line 153 of £le stringutil.h.

6.2.2.7 bool needdelimiter (string const & *s*, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR)

return true, if string delimiters is required if contain two adjacent iswhite() or string delimiters.

6.2.2.8 vector< string > & parsestring<string> (vector< string > & *strlist*, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR)

for each element of list, clear extra spaces and parse delimited string.

Referenced by dataio::add(), dataio::addnamed(), dataio::commentclose(), dataio::commentline(), and dataio::linewrap().

6.2.2.9 string & parsestring (string & *s*, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR)

clear extra spaces and parse delimited string of s the sequence of iswhite() is replaced by single space, except inside of delimited string (inside delimiters, the sequence of two delimiters is assumed one delimiters).

6.2.2.10 ostream & putline (ostream & *f*, vector< string > & *dataline*, char *separator* = _STRINGUTILCOLUMNSEPARATOR, char *newline* = '\0')

put the vector<string> to ostream dataio use it, only if _DATAIODEBUG_ is de£ned newline=='\0' is system newline.

Examples:
sampbinrec.cpp, samprec.cpp, sampreclist.cpp, and samptablewithoutdataio.cpp.

6.2.2.11 ostream & putlines (ostream & *f*, vector< vector< string > > & *datalist*, char *separator* = _STRINGUTILCOLUMNSEPARATOR, char *newline* = '\0')

vectorized putlines newline=='\0' is system new lines.

6.2.2.12 vector< string > & reverseparsestring<string> (vector< string > & *strlist*, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR)

for each element of list, convert to delimited string.

6.2.2.13 string & reverseparsestring (string & *s*, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR)

translate s to delimited string (is delimiter is found, substitute by double of one).

6.2.2.14 unsigned stringcase£nd (string const & *s1*, string const & *s2*, unsigned *pos* = 0)

case insensive version of string::£nd().

6.2.2.15 unsigned stringcase£nd_last_of (string const & *s1*, string const & *s2*, unsigned *pos* = 0)

case insensive version of string::£nd_last_of() does not used by dataio.

6.2.2.16 bool stringdelimiterbalanced (string const & *s*, char *strdelimiter* = _STRINGUTILSTRDELIMITER, char *scapechar* = _STRINGUTILSCAPECHAR, unsigned *pos1* = 0, unsigned *pos2* = (∼0U))

return true, if string delimiter is closed correctly between [pos1, pos2).

6.2.2.17 bool stringutilprinterror (bool *status*)

6.2.2.18 bool stringutilprinterror ()

print error message default value is _STRINGUTILDEFAULTPRINTERROR;

See also:
stringutilthrowexception, _STRINGUTILDEFAULTPRINTERROR.

6.2.2.19 bool stringutilthrowexception (bool *status*)

6.2.2.20 bool stringutilthrowexception ()

throw exception default value is _STRINGUTILDEFAULTTHROWEXCEPTION

See also:
stringutilprinterror, _STRINGUTILDEFAULTPRINTERROR.

6.2.2.21 char tocntrl (char *c*) `[inline]`

De£nition at line 158 of £le stringutil.h.

6.2.2.22 vector< vector< string > > & transpose<vector <string> > (vector< vector< string > > & *data*)

transpose the vector<vector <string> > data array (replace row and column).

Examples:
samptablewithoutdataio.cpp.

6.3 data format draft 0.1 and note

The project page is `http://dataio.sourceforge.net/.`

See also:
dataio

Note:
supported primitive data type

The supported primitive data type are:

bool, char, wchar_t (wchat_t does not work?), string, int, long, unsigned, unsigned long, ¤oat, double, long double.

To add new primitive type T, specialize the following two template methods:

_dataiorec<T>::stringtoitem(string &s, T &iten); // will change s, if desired

_dataiorec<T>::itemtostring(T const &item, string &s);

If T is supported primitive type (that above two methods is epecialized) vector <T> and vector<vector<T> > are automatically supported.

CSV (comma separated values) style
This is the basic data format used by dataio.

The data is separated by columnseparator

some conventions:

- The dataio library skip extra spaces between words, except inside block delimited by stringdelimiter.
- If space is used as column separator, the extra spaces is skiped.
- in the simple text mode (charspeci£er == '\0'??), the delimited string will contain any text, except the linedelimiter and no special representation is required. The string delimiter is represented by two subsequent string delimiter (the charspeci£er?? is not implemented yet)
- the standard delimited string is assumed as single delimited block (is suitable to check the standard if extendedmode is off).
- The dataio (change to do it only if extendedmode is on??), it will recognize sequence of several blocks and words. In this case:
  - extra space between words is deleted, and space between word and delimited block is desconsidered.
  - The adjacent delimited block (possible separated by white spaces) is joined.
- in the simple text mode (charspeci£er == '\0'??), the string is delimited if contain column separator, comment markers, sequence of more than one !isgraph(), or string delimiters. It it contain lineseparator, output will cause an error.
- CAUTION: EOF char (normally Ctrl Z) will not used except for lineseparator, because the input methods assume as lineseparator.
- todo:
  - delimmited string will contain \?? as decimal speci£cation?

The each data block the form

[variable 1][separator][values 11]

...

[separator][values 1n]

...

[variable n][separator][values n1]

...

[separator][values nn]

- single primitive type

the primitive data with names is posted in the data as

[var name] [separator] [value]

and without names as

[value]

- vector of primitive type

The vector is posted as value separated by columnseparator, in the single line (if line is long, will use linewrap features, but remember that it meke unsupported by spreadseet application).

The vector data with name is the form

[var name] [separator] [values list]

or

[var name]

[separator] [values list]

The vector withou names is the form

[values list]

- vector of vector of the primitive type

The vector<vector> are formed by several lines, each of this containning data separated by column separator as vector

with names, it is posted as

[var name] [separator] [£rst line values list]

[separator] [second line values list]

...

[separator] [last line values list]

or

[var name] [separator]

[£rst line values list]

[separator] [second line values list]

...

[separator] [last line values list]

this data without names is the form

[£rst line values list]

[second line values list]

...

[last line values list]

[empty line]

Note:

The primitive data type and their vectors will added to loader using dataio::add() methods. If name is passed, assumed as £eld as name, and if name is omoted, is assumed as name less £eld.

- composed data type

The data corresponding to composed member data obedies the above requisites, but all member £eld need to one column shifted to indicate that is the member. for example, record ”person” with £elds: name=Paul, telephon = xxxxx, age=221, is soted as

person [separator] name [separator] Paul

[separator] telephon [separator] xxxxx

[separator] age [separator] 21

with item name, or

[separator] name [separator] Paul

[separator] telephon [separator] xxxxx

[separator] age [separator] 21

without names. The record £eld will or not use the name. for example, Above data with item name, but name of record without £eld name is the form

person [separator] Paul

[separator] telephon [separator] xxxxx

[separator] age [separator] 21

the only requeriments is that the empty one column is added in the left side of each lines of composed data (relative to current position). The other rules is same as non composed data.

For composed data£eld with names, the end of datas is assumed as ocurrence of line that is not shifted as current position. (that contain non empty data at left of corrent shift position - shift position is one shifted relative to name column)

for composed data without names, the end of data is detected with their £eld components. No special rule to determine end of speci£c composed data £eld.

- note The standard CSV skip all extra space (independent of the space is column separator or not) and while space that is not column separator or new line marker. The current stage of dataio skip extra spaces only if it is the column separator

- to think: Perphaps, is usefull to insert option so that the all empty column in the line will skiped?
- todo: except the case then column separator is space, the empty column at end of line is considered. In dataio, the empty column at end of line are ignored. Need to insert ¤ag to make on/off the empty column clearing.
- The string that contain more than one space between words, or contain special character or word, need to delimited with string delimiter. The most of spreadseet applications support only the input of this. The dataio does not support string delimiters, but in future, will support input/output of delimited string.
- To think: The incorpolate character delimiters? the standard spreadseet applications do not implement char, because is considered as string containning one character.

Note:

- The composed data £eld requires that is posted inside of one dataio class and added to dataio loader class. The recomendation is to use composed data only with names, in way to obtain more portability.
- The dataio is assumed to composed data (with £elds is the added itens) sometimes, the composed data is stored (or associated as item of the) extended class of dataio to perform validation, or manage complex data, such as table. in this case, the pointer casting to dataio is required. For example, if mydataio is derived class as

class mydataio : dataio {

....

};

Suppose that mydata is mydataio type and io is dataio type;

In way to add mydata as item of io with name ”sub£eld”, call

io.add(”sub£eld”, (dataio ∗)&mydata); // if derived class, need casting

important note:

- The data without names will appear only before the names data (independent of the order that the £eld is added to loader). The unnamed data is processed in order until this is complete. After this, start the named data £eld readding. The named data does not require that ordered.
- Empty column of end of row are ignored. If all of colun is empty (blank, or comment only), is assumed as blank line (used as data £eld separator).
- The dataio is very con£gurable, for example:
  - case sensive or not (using dataio::ignorecase())
  - de£ne column separator (using dataio::columnseparator)
  - change decimal denotatioon for ¤oat (use char dataio::decimal(char))
  - and much more.
- The lineseparator is con£gurable in way to support different data format, but the cross platform users require special caution. The new line marker differ dependding the operating system as
  - UNIX: ’\n’ (LF)
  - ANSI (DOS/WINDOW): ’\r\n’ (CR LF)
  - VAX/VMS: ’\r’ (CR)
  - UNKNOW (there exist?): ’\n\r’ (LF CR) The dataio will detect new line automatically (default)

multi-block style

If some name appear more that once, is assumed that the data is the list blocken in the elements. For example, the data item = {x, y, z) will writed in the form

item [separator] x

item [separator] y

item [separator] z

It is suitable to describe the list of composed data (see table features). Note that can not use this features without item name, or within table data.

Note:

To manage correctly this, need to add item as member of loader, and item is that associated to extended class of dataio, so the methods

void dataio::startinblock(unsigned i) // prepare to input column i

bool dataio::startoutblock(unsigned i) // prepare to input column i

and optional

void dataio::validate(unsigned i) // validate column i that inputed

are implemented.

column oriented data

The data will be column oriented. In this case, is assumed to the transposed one (column is row and row is column) relative to standard one. As the table format, the blank line is assumed as end of datablock

If activate columnoriented and istable, the data becames as most popular spreadseet like table format.

table data

In this case, vector<vector> will not used as £eld data (the vector is supported, and the element is itored, one in each column). For example, suppose that ';' is used as column separator, then

name ; Alfred ; Paul ; Mary

point ; 12 ; 15 ; 17

; 21 ; 52 ; 37

; 9 ; 7 ; 27

sum ; 123 ; 74 ; 81

store the values:

column 1:

- name = Alfred
- point = 12, 21, 9
- sum = 123

column 2:

- name = paul
- point = 15, 52, 7
- sum = 74

column 3:

- name = mary

- point = 17, 37, 27
- sum = 81

The blanck line are assumed as end of table.

Note:

In way to process table data format, need to extend dataio class and overwrite the methods:

void dataio::startinblock(unsigned i) // prepare to input column i

bool dataio::startoutblock(unsigned i) // prepare to input column i

and optional

void dataio::validate(unsigned i) // validate column i that inputed

attributor enabled mode

In the con£g £le, normaly the value is attributed using attributor To support this, dataio permit use of attributor to attribute value for name (replacing column separator) in the £rst level named variables.

Actually, the attributor enabled mode will input/output unnamed data too, but it will disabled in way to compatibilize as most standard.

The attributor mode similar to CVS, except the fact that the separator between variables and data is [attribseparator]. Actually, [column separator] and [attribseparator] will used as separator of £rst column.

If extendedmode is disabled, the variable without names will not used, due to most popular standard.

- todo: need to make cheking of correct use of this (only attributor will used to attribute values), in the standard mode (the extended mode, it is impossible, because will use unnamed data. will check, if unnamed is empty()).

Note: this mode is ignored by column oriented data and table data

section enabled mode

The windows ini £le use section and it is used to the applications for readding only the data associated to it. The data io assume that, if one of sectionnameopen or sectionnameclose is enabled (isgraph character), assume that the £rst level (not shifted data) name is used as section.

Corrently, will use unnamed data in the section enabled mode, but it will desactivated to standarize, i.e, On the £rst level (not sub£eld) the only named data will be used in the section activated mode. for example, windows ini£el do not permit use of variable outside of section de£nition, but the current one do not checking (and empty section de£nition ends the current section)

The standard are as same as CSV, that affect the composed data £eld in the following way. The name of composed data is posted, delimited by sectionname marker and the one column shifting of their data£eld is skipped:

[sectionnameopen][var name][sectionnameclose] [datalist 1] ... [datalist n]

The end of this block is assumed as occurrence of next section de£nition.

The section scape is made by empty section marker, and no subsection is suported. The dataio provides datade£nition outside of section (data£eld that is not composed data) only before compose data by convenience, but section scape will be disabled. Note that non section data before section is not recomended, because is not windows standard.

If extendedmode is disabled, the variable without names will not used to make standard. Note that the non-composed data type will be used because it is usefull, but is not the most popular standard.

- Todo: the output methods will changed to output the non composed data £rst and follows to composed data??.
- to think: On the input, emppty section is will ignored for compatibility for standard? or continue to supporting??

Note:
the section enabled mode is ignored by column oriented data and table data.

Note:
The section enabled, attrib separator enabled, exteddedmode disabled mode Assume that is standard sectionized style that each composed element does not contain unnamed data member and their members is simple, or vector. Thus, insert attrib separator between £rst and seccond column.

comment on data

- The two comment is supported:
  1. comment block delimited by commentopen and commentclose marker (if one of this is empty, assumed as disabled)
  2. the line comment: if commentline marker is found, assume taht rest of current line is comment.

- need to add the member before their comment. To add comment, search for arleady added members. If not found, the action is not performed. For example, to add comment to item foo, need to add foo £rst.
- comment output
  1. the global comment is outputed £rst, using one line for each item. The line commentting mode is used as prefered mode, and an extra blank line is inserted after end of global comment (if global comment is not empty).
  2. If istable and columnoriented is disabled, each item comment is outputed, using one line for each item. The line commentting mode is used as prefered mode.
  3. for comment of subitems, the block commenting mode is used. Each comment item is delimited by comment delimiter. The comment list is treated like as list of values and the output depend the columnoriented. If no columnoriented is applyed is single line. (Each columnoriented between subitem and main class make traspose of data)
  4. istable active mode: No member comment is outputed, but sub members comment is outputed.
  5. column oriented mode: The all member comment are outputed as subitems like comment mode.
- Todo: In future, is abled to read comment line (line that only non white space value is comment) as single comment and store in the comment list associated to next data to read.

line wrapping
if linewrap is enabled, will break long line as several one The line wrapmarker says to join the line at the next one.

For example, if '\' is linewrap marker,

a [separator] b \

[separator] 7 [separator] 9

is interpreted as

a [separator] b [separator] 7 [separator] 9

Notes

¤ags related: ———— 1. maped into member during input/outputsetvalue/getvalue process

are: lineseparator, columnseparator, linewrap, commentline, commentopenmark, commentclosemark, attribseparator, stringdelimiter, parseallinputstring, todo: charmarker used to specify the special characrer, like C/C++ (if is white, assumed disabled)

2. setted by method, and optionally, will map inside member during ¤ag setting: ignorecase, emptyisvalid, decimal, validateall, collectnames todo: collectunknowdata used to decide collect or not the unknow data

3. maped by maprecomemded¤agstomember method: ignorecase, emptyisvalid, decimal, validateall, collectnames

4. Not maped inside members (do manually, if desire): sectionnameopen, sectionnameclose, istable, columnoriented, maxcolumnonline, clearemptytail, extendedmode (is checked only by operator>>() and operator<<()??

obs.: dataio::setted(string const &name) do not apply parsestring() in name. do it, if desire.

string on cell ———— 1. The £eld that is not setted to string is considered as well as, if parseallinputstring is off. 2. On the name, or string item: The sequence of white space is replaced by one single space, and space on head or tail of strig is ignored. If delimiter string is used, the space before and after this is skiped. 3. if parseallinputstring is on, all item is parsed as in (2) 4. The name and string item is outputed as delimited string, if it contain markers: commentline, commentopen, commentclose, lineseparator, columnseparator, attribseparator, stringdelimiter, linewrap (if last non white of string), section name marker (if is name on £rst column) The lineseparator is deleted from string item, but on name or others, is keeped (atempty for, this, because linesepaator inside name, or other data that is not string will cause errors).

todo: implement extended text mode that word with non white char representations, as like C/C++ style epeci£cation.

add/delete: ——— 1. Add with name that arleady exist, replace old one. If need to output record list, use multiblock features.

2. Can not delete £elds. Only delete features is clear() that delete all £elds. Note that in the istable enabled mode, inside of startinblock, validate, delete will cause errors (setvalue do not check variable type)

6.4 binary data format draft 0.1 and note

The project page is `http://dataio.sourceforge.net/`.

See also:

data format draft 0.1 and note, dataio

Note:

supported primitive data type

The supported primitive data type are:

bool, char, wchar_t (wchat_t does not work?), string, int, long, unsigned, unsigned long, ¤oat, double, long double.

To add new primitive type T, specialize the following two template methods:

bool _dataiorec<T>::readitem(istream &is, T &iten); void _dataiorec<T>::writeitem(ostream &os, T const &item);

If T is supported primitive type (that above two methods is epecialized) vector <T> and vector<vector<T> > are automatically supported.

The named and unnamed data is undistinguiched, and input/putput follows the added order.

primitive type binary formats:

- string is treated as ASC Z string (null terminates sequence of chars) and write '\0' at end of string.
- The followings data assume to more signi£cant to less signi£cant byte order. is reversebyteorder is actived, the byte is less signi£cant more signi£cant. The numerical binary format use the IEEE 754 binary standard:
  - wchar_t (2 bytes): it is not IEEE 754 standard.
  - int, unsigned (4 bytes)
  - long, unsigned long (8 bytes)
  - long long, unsigned long long (usa same as long and unsigned long): it is not the IEEE 754 standard
  - ¤oat (4 bytes - 32 bits): single precision ¤oat.
    - 8 bits for expoent in biased representation: expoent value = expoent - 127.
    - 32 - 8 bits for fraction part (£rst bit is sign).
    - the value is [fraction part]∗2^[expoent value]
  - double (8 bytes - 64 bits): double precision:
    - 11 bit for expoent in biased representation: expoent value = expoent - 1023.
  - 64 - 11 bits for fraction part (£rst bit is sign sign).
    - the value is [fraction part]∗2^[expoent value]
  - long double (use same as double): Not the IEEE standard.

To process ¤oating point binary format, useds following ANSU C functions: double frexp(double val, ∗n) to obtain expoent and fraction part: x = frexp(y,n) call set the values so that y = x∗2^n (0 <= x < 1)

double ldexp(double val, ∗n) to restore value using expoent and fraction part y = ldexp(x,n) calling set the values so that y = x∗2^n (0 <= x < 1)

- to think: implement size() that return size used by record
- to think: multi-block features in the £rst level record data

6.5 some todo listing

For compelte listing, see the ”Related Files”->”todo listing”.

Todo:

# in the current implementation, addition of £eld with same names will not work correctly. In future, add will replace if names exist?.
# implement partial comment input features?
# stringutil: implement function that parse the parameter option into the vector <vector <string> >, for dataio parsing.
# unknow data collector (data that is not tried to set)

Chapter 7

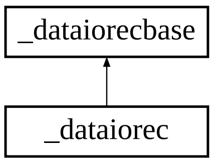
dataio Class Documentation

7.1 _dataiorec Class Template Reference

io record template class used by dataio.

`#include <dataio.h>`

Inheritance diagram for _dataiorec::



Public Methods

- bool stringtoitem (string &s, T &item)

  Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to read (from text stream). See dataio class for arlead specialized classes.

  See also:

  itemtostring, readitem, writeitem.

- void itemtostring (T const &item, string &s)

  Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to write (to text stream). See dataio class for arlead specialized classes.

  See also:

  stringtoitem, readitem, writeitem.

- bool readitem (istream &is, T &item)

  Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to read (from binary stream). See dataio class for arlead specialized classes.

  See also:

  writeitem, stringtoitem, itemtostring.

- void writeitem (ostream &os, T const &item)

  Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to write. See dataio class for arlead specialized classes.

  See also:

  readitem, stringtoitem, itemtostring.

7.1.1 Detailed Description

template<class T> class _dataiorec

io record template class used by dataio.

Internal use only. In the special case that add new class and pretend that is read/writed as primitive, need to specialize the methods: bool _dataiorec<T>::stringtoitem() and bool _dataiorec<T>::itemtostring()

De£nition at line 338 of £le dataio.h.

7.1.2 Member Function Documentation

7.1.2.1 template<class T> void _dataiorec<T>::itemtostring (T const & *item*, string & *s*)

Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to write (to text stream). See dataio class for arlead specialized classes.

See also:

stringtoitem, readitem, writeitem.

7.1.2.2 template<class T> bool _dataiorec<T>::readitem (istream & *is*, T & *item*)

Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to read (from binary stream). See dataio class for arlead specialized classes.

See also:

writeitem, stringtoitem, itemtostring.

Todo:

readitem

7.1.2.3 template<class T> bool _dataiorec<T>::stringtoitem (string & *s*, T & *item*)

Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to read (from text stream). See dataio class for arlead specialized classes.

See also:

itemtostring, readitem, writeitem.

7.1.2.4 template<class T> void _dataiorec<T>::writeitem (ostream & *os*, T const & *item*)

Specialize this methods of _dataiorec template class, if use you own primitive class that is not specialized by default and pretend to write. See dataio class for arlead specialized classes.

See also:

readitem, stringtoitem, itemtostring.

Todo:

writeitem

The documentation for this class was generated from the following £le:

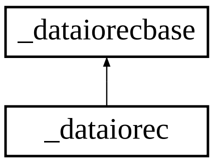
- dataio.h

7.2 _dataiorecbase Class Reference

io record base class used by ndataio class. Internal use only.

`#include <dataio.h>`

Inheritance diagram for _dataiorecbase::



7.2.1 Detailed Description

io record base class used by ndataio class. Internal use only.

De£nition at line 204 of £le dataio.h.

The documentation for this class was generated from the following £le:

- dataio.h

7.3 dataio Class Reference

dataio main class to perform data input/output features The primitive data that will read/write are: bool, char, wchar_t (wchar_t work? perphaps, no), string, int, long, unsigned, unsigned long, ﬂoat, double, long double.

```
#include <dataio.h>
```

Public Methods

- void addcomment (string const &rem)

  add an comment item to comment list
  See also:
  clearcomment, *comment*, *additemcomment*, *clearitemcomment*, *itemcomment*.

- void clearcomment (bool includemember=false)

  clear comment list
  See also:
  addcomment, *clearcomment*, *additemcomment*, *clearitemcomment*, *itemcomment*.

- vector<string>& comment ()

  return reference for comment list use carefully
  See also:
  addcomment, *clearcomment*, *additemcomment*, *clearitemcomment*, *itemcomment*.

- bool addcomment (string const &name, string const &rem)

  add an comment item to comment list of named item return false if not found
  See also:
  clearcomment, *comment*, *additemcomment*, *clearitemcomment*, *itemcomment*.

- bool additemcomment (void ∗item, string const &rem)

  add an comment item to comment list of item
  See also:
  addcomment, *clearcomment*, *comment*, *clearitemcomment*, *itemcomment*.

- void clearcomment (string const &name, bool includemember=false)

  clear comment list of named item
  See also:
  addcomment, *comment*, *additemcomment*, *clearitemcomment*, *itemcomment*.

- void clearitemcomment (void ∗item, bool includemember=false)

  clear comment list of item
  See also:
  addcomment, *clearcomment*, *comment*, *additemcomment*, *itemcomment*.

- void clearitemcomment (bool includemember=false)

  clear comment list of all items
  See also:
  addcomment, *clearcomment*, *comment*, *additemcomment*, *itemcomment*.

- vector<string>& comment (string const &name)

  return reference for comment list of named item use carefully
  See also:
  addcomment, clearcomment, additemcomment, clearitemcomment, itemcomment.

- vector<string>& itemcomment (void ∗item)

  return reference for comment list of item use carefully
  See also:
  addcomment, clearcomment, comment, additemcomment, clearitemcomment.

- string& commentline ()

  One line comment start marker. Ignore at end of current line. Default is _-DATAIODEFAULTCOMMENTLINE
  See also:
  _DATAIODEFAULTCOMMENTLINE, commentopen, commentclose.

- string& commentline (string const &com)
- string& linewrap ()

  This marker is used in the end of line (possible followed by blank space or comment) The next line is appended in the end of line. Caution: it is in sperimental stage and name will change. Default is _-DATAIODEFAULTLINEBREAK
  See also:
  _DATAIODEFAULTLINEBREAK.

- string& linewrap (string const &wrap)
- void disablesection ()

  disable section (empty sectionnameopen and sectionnameclose) to activate the section, use enablestdsection, or set manually the sectionnameopen and sectionnameclose
  See also:
  sectionnameopen, sectionnameclose, enablestdsection.

- void enablestdsection ()

  set as standard section mode (set the sectionnameopen and sectionnameclose) for non-standard section mode, set sectionnameopen and sectionnameclose manually
  See also:
  sectionnameopen, sectionnameclose, disablestdsection.

- bool throwexception ()

  if this ¤ag is true, throw exception on error default value is _DATAIODEFAULTTHROWEXCEPTION
  See also:
  printerrormessage, _DATAIODEFAULTTHROWEXCEPTION.

- bool throwexception (bool dec, bool includemember=true)
- bool printerror ()

  if this ¤ag is true, coutput error message on error default value is _DATAIODEFAULTPRINTEROR
  See also:
  throwexception, _DATAIODEFAULTPRINTEROR.

- bool printerror (bool dec, bool includemember=true)
- bool emptyisvalid ()

  if this ¤ag is true, consider empty value as valid value inside of several stringtoitem() conversion Default is _DATAIODEFAULTEMPTYISVALID

See also:

_DATAIODEFAULTEMPTYISVALID.

- bool emptyisvalid (bool ¤ag, bool includemember=true)
- bool ignorecase ()

  Returns:

  ignore case ¤ag.

- bool ignorecase (bool status, bool includemember=true)

  Set ignore case ¤ag. Default value of ignore case frag is _DATAIODEFAULTIGNORECASE

  See also:

  _DATAIODEFAULTIGNORECASE.

- char decimal ()

  Returns:

  decimal character used by ¤oating number.

- char decimal (char dec, bool includemember=true)

  set decimal character used by ¤oating number. Default is _DATAIODEFAULTDECIMAL

  See also:

  _DATAIODEFAULTDECIMAL.

- bool validateall ()

  Returns:

  true is all variable is validated.

- bool validateall (bool status, bool includemember=true)

  set ¤ag to decide if unrefered variables will be validated. Default is _DATAIODEFAULTVALIDATEALL

  See also:

  validate, _DATAIODEFAULTVALIDATEALL.

- string& commentopen ()
- string& commentopen (string const &s)

  See also:

  _DATAIODEFAULTCOMMENTOPEN, commentclose, commentline.

- string& commentclose ()

  Comment block endding marker. End the comment started by commentopen. Default is _-DATAIODEFAULTCOMMENTCLOSE

  See also:

  _DATAIODEFAULTCOMMENTCLOSE, commentopen, commentline.

- string& commentclose (string const &s)

  < return comment block endding marker Comment block endding marker. End the comment started by commentopen. Default is _DATAIODEFAULTCOMMENTCLOSE

  See also:

  _DATAIODEFAULTCOMMENTCLOSE, commentopen, commentline.

- void disablecommentblock ()

  disable comment block features (empty commentopen and commentclose).

- void enablestdcommentblock ()

enable standard comment block mode for non-standard comment block mode, set manually the commentopen and commentclose.

- bool setted ()

  Returns:

  true if all of value is correctly setted.

- bool setted (string const &name)

  Returns:

  true if the value of this, is correctly setted.

- bool itemsetted (void ∗item)

  Returns:

  true if the value of this is correctly setted.

- bool collectnames ()
- bool collectnames (bool ¤ag, bool includemember=true)

  set the ¤ag for the name collector on value setting stage. if true, the all dataio type class that the value is setted, collect the refered names, inclusive the dataio class added to this collect their own refered names. note that the £rst level (the class that operator>> is called, always collect their names). CAUTION: name of this methods will change!

  See also:

  refered, _DATAIODEFAULTREFERSUBNAMES.

- bool refered (string const &name)

  return true is name appeared in input data during operator>> data processing

  See also:

  refersubnames itemrefered.

- bool itemrefered (void ∗item)

  return true is item is tryed to set in the input stage, during operator>> data processing

  See also:

  refersubnames referedt.

- vector<string>& referednames ()
- vector<string>& unreferednames ()

  < return (reference of) referenced name list.

- void add (string const &name, dataio ∗item)

  add one data record (or data block) with names

  Parameters:

  item *is pointer to dataio or derived to dataio (the derived class need casting to dataio pointer).*

- void add (dataio ∗item)

  add one data record (or data block) without names

  Parameters:

  item *is pointer to dataio or derived to dataio (need casting to dataio pointer).*

- template<class T> void add (string const &name, T ∗item)

  add one variable with names

  Parameters:

  item *is pointer to class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().*

- template<class T> void add (T ∗item)

  add one item without names
  Parameters:
  item *is pointer to class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().*

- template<class T> void add (string const &name, vector< T > ∗item)

  add one vector of class with names
  Parameters:
  item *is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().*

- template<class T> void add (vector< T > ∗item)

  add one vector of class without names
  Parameters:
  item *is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().*

- template<class T> void add (string const &name, vector< vector< T > > ∗item)

  add one vector of vector of class with names
  Parameters:
  item *is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().*

- template<class T> void add (vector< vector< T > > ∗item)

  2 dimensional add one vector of vector of class without names
  Parameters:
  item *is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().*

- template<class T> void addnamed (string const &name, T ∗item, unsigned long &strsize=∗(unsigned long ∗) 0)

  TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence
  Parameters:
  item *is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().*

- template<class T> void addunnamed (T ∗item, unsigned long &strsize=∗(unsigned long ∗) 0)

  TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence
  Parameters:
  item *is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().*

- template<class T> void addnamed (string const &name, vector< T > ∗item, unsigned long &m=∗(unsigned long ∗) 0, unsigned long &strsize=∗(unsigned long ∗) 0)

  TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write vector size in binary i/o. itherwise, assume that is £xed size = m vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

Parameters:
: ***item*** *is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().*

- template<class T> void addunnamed (vector< T > ∗item, unsigned long &m=∗(unsigned long ∗) 0, unsigned long &strsize=∗(unsigned long ∗) 0)

  TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write vector size in binary i/o. itherwise, assume that is £xed size = m vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

  Parameters:
  : ***item*** *is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().*

- template<class T> void addnamed (string const &name, vector< vector< T > > ∗item, unsigned long &m=∗(unsigned long ∗) 0, unsigned long &n=∗(unsigned long ∗) 0, unsigned long &strsize=∗(unsigned long ∗) 0)

  TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write row number in binary i/o. itherwise, assume that is £xed row number = m. if &n == 0, read/write vector size in binary i/o. otherwise, assume that is £xed size = n vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

  Parameters:
  : ***item*** *is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().*

- template<class T> void addunnamed (vector< vector< T > > ∗item, unsigned long &m=0, unsigned long &n=∗(unsigned long ∗) 0, unsigned long strsize=∗(unsigned long ∗) 0)

  TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write row number in binary i/o. itherwise, assume that is £xed row number = m. if &n == 0, read/write vector size in binary i/o. otherwise, assume that is £xed size = n vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

  Parameters:
  : ***item*** *is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().*

- void setvalue (vector< vector< string > > &item)

  set values stored in vector<vector<string> >

  See also:
  : *getvalue,operator>>, operator<<, read, write.*

- vector<vector <string> >& getvalue (bool includecomment=false)

  get values as vector<vector<string> > The global comment is not included in the list!

  See also:
  : *setvalue, operator>>, operator<<, read, write.*

- istream& read (istream &is)

  In testing stage: This is to input cross platoform binary data from istream (for text stream input, use operator>>

  See also:
  : *write, reversebyteorder, operator>>, operator<<.*

- ostream& write (ostream &os)

In testing stage: This is to output cross platoform binary data to ostream (for text stream output, use operator<<

See also:

read, reversebyteorder, operator>>, operator<<.

- virtual void startinblock (unsigned i)

  method for input block preparator. The multi-block input, or table data input requires that extend dataio class and overwrite this so that this routine make block i-th preparation. (in this place, you need to set variables corresponding to i-th block.) See the samptable.cpp Note: The value of i increase in each call of startinblock() If pretend to auto validate the input, overwrite validate

  See also:

  istable, setted, refered, validate, startoutblock.

- virtual void validate (unsigned n)

  The data validator To validate the data, overwrite this Note that, in the multi-block input, or table data, the information as setted, refered, etc, is only for i-th block) - value of i increase in each call of validate - called only once for each value setting - the member that not tried to set value is not validated. is not validated. Will use refered() to check if spcial member is tried to validated - the main class that call operator>> call their validate

  See also:

  istable, setted, refered, startinbloc.

- virtual bool startoutblock (unsigned i)

  For multi-block output or table data output, need to overwrite this routine in way to to perform i-th data setting

  See also:

  istable, startinblock, validate.

- dataio ()

  default constructor.

- void clearinputbuffer ()

  clear the memory used by input/output transaction, After this, setted(), refered(), etc, will not work unless new input is performed.

- void reset¤ags (bool maprecomended¤agsintomember=true)

  reset all status ¤ags: istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator. Note that the decimal, commentopen, commentclose, and emptyisvalid, are recursivelly applied into the member (dataio class element added in current class).

  See also:

  add(), clear(), istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator, emptyisvalid, throwexception, errormessageon.

- void clear ()

  clear all variables added by add(). The status of istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator. are preserved.

  See also:

  add(), reset().

- void reset ()

  reset status: Clear all variables and reset all status ¤ags.

  See also:

  add(), reset¤ags(), clear(), istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator.

- virtual ~dataio ()

  default destructor.

Public Attributes

- unsigned maxcolumnonline

  If number of column exceds this number and linewrap is non empty, the operator>> perform line wrapping. Default is _DATAIODEFAULTMAXCOLUMNONLINE

  See also:

  _DATAIODEFAULTMAXCOLUMNONLINE, linewrap.

- bool istable

  table data ¤ag (if true, data will be table). Default is _DATAIODEFAULTISTABLE Commonly, the columnoriented is used together

  See also:

  _DATAIODEFAULTISTABLE, columnoriented.

- char sectionnameopen

  section name delimiter (used before section maem) for con£g £le. '\0' is assumed inactive. Default is _DATAIODEFAULTSECTIONNAMEOPEN If both of sectionnameopen and sectionnameclose is disabled (iswhite), con£g £le is assumed that was not section

  See also:

  _DATAIODEFAULTSECTIONNAMEOPEN, sectionnameclose, disablesection, enablestdsection.

- char sectionnameclose

  section name delimiter (used after section name) for con£g £le. Default is _DATAIODEFAULTSECTIONNAMECLOSE If both of sectionnameopen and sectionnameclose is disabled (iswhite), data is assumed that was not section

  See also:

  _DATAIODEFAULTSECTIONNAMECLOSE, sectionnameopen, enablestdsection, disablesection.

- bool columnoriented

  Column oriented data ¤ag. If true, the data is assumed as transpose of normal data. Default is _DATAIODEFAULTCOLUMNORIENTED. commonly, used together istable

  See also:

  _DATAIODEFAULTCOLUMNORIENTED, istable.

- char columnseparator

  The column separator of data £eld. Default is _DATAIODEFAULTCOLUMNSEPARATOR

  See also:

  _DATAIODEFAULTCOLUMNSEPARATOR, lineseparator.

- char lineseparator

  Line separator of data. Default is _DATAIODEFAULTLINESEPARATOR

  See also:

  _DATAIODEFAULTLINESEPARATOR, _DATAIODEFAULTAUTOLINESEPARATOR, columnseparator.

- char attribseparator

con£g attrib separator for data, if it is non zero, is assumed to use attribseparator to indicate atribution for veriable. Caution: The attribseparator will be used only for - istable and columnoriented is disabled - not sub£elds (non shifted data £eld only). The attribseparator and columnseparator will toghether in the data £le! os os ognored is istable or columnoriented ¤oag is true. Default is _-DATAIODEFAULTATTRIBSEPARATOR

See also:

istable, columnoriented, _DATAIODEFAULTATTRIBSEPARATOR.

- bool extendedmode

  if this ¤ag is true, work in extended format support mode that permit input/output the non standard data type for example, section enabled data will contain non composed data default value is _-DATAIODEFAULTEXTENDEDMODE

  See also:

  _DATAIODEFAULTEXTENDEDMODE.

- bool clearemptytail

  if this ¤ag is true, delete empty tail of each lines default value is _DATAIODEFAULTCLEAREMPTYTAIL

  See also:

  _DATAIODEFAULTCLEAREMPTYTAIL.

- bool parseallinputstring

  if this ¤ag is true, all item is parsed (using parsestring) in the input stage to solve delimited string problem, independent of the candidate for name or not. default value is _-DATAIODEFAULTPARSEALLITEMSTRING. This ¤ag is speciall that applied into all of members, independent of the map¤agstomember, in way to evit the double parsing in the stringtoitem (the default one that use it is the string type)

  See also:

  _DATAIODEFAULTPARSEALLINPUTSTRING.

- char stringdelimiter

  the string delimiter. Default is _DATAIODEFAULTSTRINGDELIMITER

  See also:

  scapechar, _DATAIODEFAULTSTRINGDELIMITER.

- char scapechar

  the scape char (to specify special char) for delimited string Default is _DATAIODEFAULTSCAPECHAR

  See also:

  stringdelimiter, _DATAIODEFAULTSCAPECHAR.

- bool reversebyteorder

  validate unreferenced variable?? if false, ignore validate calling for unreferenced members. if true, call validate for all members default value is _DATAIODEFAULTVALIDATEALL

  See also:

  validate, istartinblock, _DATAIODEFAULTVALIDATEALL used by binary input/output. If true, reverse byte order for numeric i/o default value is _DATAIODEFAULTREVERSEBYTEORDER , read, write, _DATAIODEFAULTREVERSEBYTEORDER.

- friend _dataiorec<T>

Friends

- istream& operator>> (istream &is, dataio &data)

  extractor: read data from istream (input from CSV text)

See also:
operator<<, setvalue, getvalue, read, write.

- ostream& operator<< (ostream &os, dataio &data)

  insertor for dataio: write data to ostream (output as CSV text)
  See also:
  operator>>, setvalue, getvalue, read, write.

7.3.1 Detailed Description

dataio main class to perform data input/output features The primitive data that will read/write are: bool, char, wchar_t (wchar_t work? perphaps, no), string, int, long, unsigned, unsigned long, ¤oat, double, long double.

See dataformat for detail. The cross platform user require special caution on the system diference of line breaking of text £le. See see dataformat for detail

See also:
data format draft 0.1 and note

Examples:
sampattrib.cpp, sampbinrec.cpp, samprec.cpp, sampreclist.cpp, sampsec.cpp, and samptable.cpp.

De£nition at line 497 of £le dataio.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 dataio::dataio () `[inline]`

default constructor.

De£nition at line 1218 of £le dataio.h.

7.3.2.2 dataio::~dataio () `[inline, virtual]`

default destructor.

De£nition at line 1253 of £le dataio.h.

7.3.3 Member Function Documentation

7.3.3.1 template<class T> void dataio::add (vector< vector< T > > ∗ *item*) `[inline]`

2 dimensional add one vector of vector of class without names

Parameters:

item is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().

De£nition at line 939 of £le dataio.h.

7.3.3.2 template<class T> void dataio::add (string const & *name*, vector< vector< T > > ∗ *item*) `[inline]`

add one vector of vector of class with names

Parameters:

item is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().

De£nition at line 928 of £le dataio.h.

7.3.3.3 template<class T> void dataio::add (vector< T > ∗ *item*) `[inline]`

add one vector of class without names

Parameters:

item is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().

De£nition at line 918 of £le dataio.h.

7.3.3.4 template<class T> void dataio::add (string const & *name*, vector< T > ∗ *item*) `[inline]`

add one vector of class with names

Parameters:

item is pointer to vector of class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().

De£nition at line 908 of £le dataio.h.

7.3.3.5 template<class T> void dataio::add (T ∗ *item*) `[inline]`

add one item without names

Parameters:

item is pointer to class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().

De£nition at line 896 of £le dataio.h.

7.3.3.6 template<class T> void dataio::add (string const & *name*, T ∗ *item*) `[inline]`

add one variable with names

Parameters:

item is pointer to class specialized by _dataiorec::stringtoitem() and _dataiorec::itemtostring().

De£nition at line 881 of £le dataio.h.

7.3.3.7 void dataio::add (dataio ∗ *item*) `[inline]`

add one data record (or data block) without names

Parameters:

item is pointer to dataio or derived to dataio (need casting to dataio pointer).

De£nition at line 863 of £le dataio.h.

7.3.3.8 void dataio::add (string const & *name*, dataio ∗ *item*) `[inline]`

add one data record (or data block) with names

Parameters:

item is pointer to dataio or derived to dataio (the derived class need casting to dataio pointer).

Examples:

sampattrib.cpp, sampbinrec.cpp, samprec.cpp, sampreclist.cpp, sampsec.cpp, and samptable.cpp.

De£nition at line 846 of £le dataio.h.

7.3.3.9 bool dataio::addcomment (string const & *name*, string const & *rem*)

add an comment item to comment list of named item return false if not found

See also:

clearcomment, comment, additemcomment, clearitemcomment, itemcomment.

7.3.3.10 void dataio::addcomment (string const & *rem*) `[inline]`

add an comment item to comment list

See also:

clearcomment, comment, additemcomment, clearitemcomment, itemcomment.

Examples:

sampattrib.cpp, sampbinrec.cpp, samprec.cpp, sampreclist.cpp, sampsec.cpp, and samptable.cpp.

De£nition at line 576 of £le dataio.h.

7.3.3.11 bool dataio::additemcomment (void ∗ *item*, string const & *rem*)

add an comment item to comment list of item

See also:
addcomment, clearcomment, comment, clearitemcomment, itemcomment.

Examples:
sampbinrec.cpp, samprec.cpp, and sampsec.cpp.

7.3.3.12 template<class T> void dataio::addnamed (string const & *name*, vector< vector< T > > ∗ *item*, unsigned long & *m* = ∗(unsigned long ∗)0, unsigned long & *n* = ∗(unsigned long ∗)0, unsigned long & *strsize* = ∗(unsigned long ∗)0) `[inline]`

TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write row number in binary i/o. itherwise, assume that is £xed row number = m. if &n == 0, read/write vector size in binary i/o. otherwise, assume that is £xed size = n vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

Parameters:
item is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().

De£nition at line 1049 of £le dataio.h.

7.3.3.13 template<class T> void dataio::addnamed (string const & *name*, vector< T > ∗ *item*, unsigned long & *m* = ∗(unsigned long ∗)0, unsigned long & *strsize* = ∗(unsigned long ∗)0) `[inline]`

TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write vector size in binary i/o. itherwise, assume that is £xed size = m vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

Parameters:
item is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().

De£nition at line 1003 of £le dataio.h.

7.3.3.14 template<class T> void dataio::addnamed (string const & *name*, T ∗ *item*, unsigned long & *strsize* = ∗(unsigned long ∗)0) `[inline]`

TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

Parameters:

item is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().

De£nition at line 963 of £le dataio.h.

7.3.3.15 template<class T> void dataio::addunnamed (vector< vector< T > > ∗ *item*, unsigned long & *m* = 0, unsigned long & *n* = ∗(unsigned long ∗)0, unsigned long *strsize* = ∗(unsigned long ∗)0) `[inline]`

TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write row number in binary i/o. itherwise, assume that is £xed row number = m. if &n == 0, read/write vector size in binary i/o. otherwise, assume that is £xed size = n vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

Parameters:

item is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().

De£nition at line 1073 of £le dataio.h.

7.3.3.16 template<class T> void dataio::addunnamed (vector< T > ∗ *item*, unsigned long & *m* = ∗(unsigned long ∗)0, unsigned long & *strsize* = ∗(unsigned long ∗)0) `[inline]`

TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. if &m == 0, read/write vector size in binary i/o. itherwise, assume that is £xed size = m vector. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

Parameters:

item is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().

De£nition at line 1026 of £le dataio.h.

7.3.3.17 template<class T> void dataio::addunnamed (T ∗ *item*, unsigned long & *strsize* = ∗(unsigned long ∗)0) `[inline]`

TEST Purpose of addnamed and addunnamed are to perform binary i/o with £xed size vector or string manipulation. It is in testing stage. same as add for text mode. same as add for text mode. strsize is only for binary mode £xed size string input/output if &strsize == 0 use ASC Z string mode. otherwise, use £xed lenght=strsize of char sequence

Parameters:

item is pointer to class specialized by _dataiorec::readitem() and _dataiorec::writeutem().

De£nition at line 982 of £le dataio.h.

7.3.3.18 void dataio::clear ()

clear all variables added by add(). The status of istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator. are preserved.

See also:
add(), reset().

Examples:
sampreclist.cpp, and samptable.cpp.

7.3.3.19 void dataio::clearcomment (string const & *name*, bool *includemember* = false)

clear comment list of named item

See also:
addcomment, comment, additemcomment, clearitemcomment, itemcomment.

7.3.3.20 void dataio::clearcomment (bool *includemember* = false)

clear comment list

See also:
addcomment, clearcomment, additemcomment, clearitemcomment, itemcomment.

Examples:
sampbinrec.cpp, and sampreclist.cpp.

7.3.3.21 void dataio::clearinputbuffer ()

clear the memory used by input/output transaction, After this, setted(), refered(), etc, will not work unless new input is performed.

7.3.3.22 void dataio::clearitemcomment (bool *includemember* = false)

clear comment list of all items

See also:
addcomment, clearcomment, comment, additemcomment, itemcomment.

7.3.3.23 void dataio::clearitemcomment (void $*$ *item*, bool *includemember* = false)

clear comment list of item

See also:
addcomment, clearcomment, comment, additemcomment, itemcomment.

7.3.3.24 bool dataio::collectnames (bool ¤ag, bool *includemember* = true)

set the ¤ag for the name collector on value setting stage. if true, the all dataio type class that the value is setted, collect the refered names, inclusive the dataio class added to this collect their own refered names. note that the £rst level (the class that operator>> is called, always collect their names). CAUTION: name of this methods will change!

See also:
refered, _DATAIODEFAULTREFERSUBNAMES.

7.3.3.25 bool dataio::collectnames () `[inline]`

De£nition at line 816 of £le dataio.h.

Referenced by add().

7.3.3.26 vector< string > & dataio::comment (string const & *name*)

return reference for comment list of named item use carefully

See also:
addcomment, clearcomment, additemcomment, clearitemcomment, itemcomment.

7.3.3.27 vector< string > & dataio::comment () `[inline]`

return reference for comment list use carefully

See also:
addcomment, clearcomment, additemcomment, clearitemcomment, itemcomment.

De£nition at line 583 of £le dataio.h.

7.3.3.28 string & dataio::commentclose (string const & *s*) `[inline]`

< return comment block endding marker Comment block endding marker. End the comment started by commentopen. Default is _DATAIODEFAULTCOMMENTCLOSE

See also:
_DATAIODEFAULTCOMMENTCLOSE, commentopen, commentline.

De£nition at line 790 of £le dataio.h.

7.3.3.29 string & dataio::commentclose () `[inline]`

Comment block endding marker. End the comment started by commentopen. Default is _-DATAIODEFAULTCOMMENTCLOSE

See also:
_DATAIODEFAULTCOMMENTCLOSE, commentopen, commentline.

De£nition at line 785 of £le dataio.h.

7.3.3.30 string & dataio::commentline (string const & *com*) `[inline]`

De£nition at line 616 of £le dataio.h.

7.3.3.31 string & dataio::commentline () `[inline]`

One line comment start marker. Ignore at end of current line. Default is _-DATAIODEFAULTCOMMENTLINE

See also:
_DATAIODEFAULTCOMMENTLINE, commentopen, commentclose.

Examples:
sampbinrec.cpp, and samprec.cpp.

De£nition at line 615 of £le dataio.h.

7.3.3.32 string & dataio::commentopen (string const & *s*) `[inline]`

See also:
_DATAIODEFAULTCOMMENTOPEN, commentclose, commentline.

De£nition at line 779 of £le dataio.h.

7.3.3.33 string & dataio::commentopen () `[inline]`

De£nition at line 773 of £le dataio.h.

7.3.3.34 char dataio::decimal (char *dec*, bool *includemember* = true)

set decimal character used by ¤oating number. Default is _DATAIODEFAULTDECIMAL

See also:
_DATAIODEFAULTDECIMAL.

7.3.3.35 char dataio::decimal () `[inline]`

Returns:
decimal character used by ¤oating number.

Examples:
sampreclist.cpp, and samptable.cpp.

De£nition at line 739 of £le dataio.h.

Referenced by add().

7.3.3.36 void dataio::disablecommentblock () `[inline]`

disable comment block features (empty commentopen and commentclose).

De£nition at line 795 of £le dataio.h.

7.3.3.37 void dataio::disablesection () `[inline]`

disable section (empty sectionnameopen and sectionnameclose) to activate the section, use enablestdsection, or set manually the sectionnameopen and sectionnameclose

See also:
sectionnameopen, sectionnameclose, enablestdsection.

De£nition at line 656 of £le dataio.h.

7.3.3.38 bool dataio::emptyisvalid (bool ¤*ag*, bool *includemember* = true)

7.3.3.39 bool dataio::emptyisvalid () `[inline]`

if this ¤ag is true, consider empty value as valid value inside of several stringtoitem() conversion Default is _DATAIODEFAULTEMPTYISVALID

See also:
_DATAIODEFAULTEMPTYISVALID.

De£nition at line 729 of £le dataio.h.

Referenced by add().

7.3.3.40 void dataio::enablestdcommentblock () `[inline]`

enable standard comment block mode for non-standard comment block mode, set manually the commentopen and commentclose.

De£nition at line 802 of £le dataio.h.

7.3.3.41 void dataio::enablestdsection () `[inline]`

set as standard section mode (set the sectionnameopen and sectionnameclose) for non-standard section mode, set sectionnameopen and sectionnameclose manually

See also:
sectionnameopen, sectionnameclose, disablestdsection.

De£nition at line 663 of £le dataio.h.

7.3.3.42 vector< vector< string > > & dataio::getvalue (bool *includecomment* = false)

get values as vector<vector<string> > The global comment is not included in the list!

See also:
setvalue, operator>>, operator<<, read, write.

7.3.3.43 bool dataio::ignorecase (bool *status*, bool *includemember* = true)

Set ignore case ¤ag. Default value of ignore case frag is _DATAIODEFAULTIGNORECASE

See also:
_DATAIODEFAULTIGNORECASE.

7.3.3.44 bool dataio::ignorecase () `[inline]`

Returns:
ignore case ¤ag.

Examples:
sampbinrec.cpp, samprec.cpp, sampreclist.cpp, sampsec.cpp, and samptable.cpp.

De£nition at line 733 of £le dataio.h.

Referenced by add().

7.3.3.45 vector< string > & dataio::itemcomment (void ∗ *item*)

return reference for comment list of item use carefully

See also:
addcomment, clearcomment, comment, additemcomment, clearitemcomment.

7.3.3.46 bool dataio::itemrefered (void ∗ *item*) `[inline]`

return true is item is tryed to set in the input stage, during operator>> data processing

See also:
refersubnames referedt.

De£nition at line 836 of £le dataio.h.

7.3.3.47 bool dataio::itemsetted (void ∗ *item*)

Returns:
true if the value of this is correctly setted.

7.3.3.48 string & dataio::linewrap (string const & *wrap*) `[inline]`

De£nition at line 625 of £le dataio.h.

7.3.3.49 string & dataio::linewrap () `[inline]`

This marker is used in the end of line (possible followed by blank space or comment) The next line is appended in the end of line. Caution: it is in sperimental stage and name will change. Default is _-DATAIODEFAULTLINEBREAK

See also:
_DATAIODEFAULTLINEBREAK.

De£nition at line 624 of £le dataio.h.

7.3.3.50 bool dataio::printerror (bool *dec*, bool *includemember* = true)

7.3.3.51 bool dataio::printerror () `[inline]`

if this ¤ag is true, coutput error message on error default value is _DATAIODEFAULTPRINTEROR

See also:
throwexception, _DATAIODEFAULTPRINTEROR.

De£nition at line 721 of £le dataio.h.

7.3.3.52 istream & dataio::read (istream & *is*)

In testing stage: This is to input cross platoform binary data from istream (for text stream input, use operator>>

See also:
write, reversebyteorder, operator>>, operator<<.

Examples:
sampbinrec.cpp.

7.3.3.53 bool dataio::refered (string const & *name*) `[inline]`

return true is name appeared in input data during operator>> data processing

See also:
refersubnames itemrefered.

Examples:
sampbinrec.cpp, samprec.cpp, and sampreclist.cpp.

De£nition at line 830 of £le dataio.h.

7.3.3.54 vector< string > & dataio::referednames<string> () `[inline]`

Examples:
sampbinrec.cpp, and samprec.cpp.

De£nition at line 839 of £le dataio.h.

7.3.3.55 void dataio::reset () `[inline]`

reset status: Clear all variables and reset all status ¤ags.

See also:
add(), reset¤ags(), clear(), istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator.

De£nition at line 1246 of £le dataio.h.

7.3.3.56 void dataio::reset¤ags (bool *maprecomended¤agsintomember* = true)

reset all status ¤ags: istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator. Note that the decimal, commentopen, commentclose, and emptyisvalid, are recursivelly applied into the member (dataio class element added in current class).

See also:
add(), clear(), istable, ignorecase(), commentline, commentopen, commentclose, columnoriented, decimal(), columnseparator, lineseparator, emptyisvalid, throwexception, errormessageon.

7.3.3.57 bool dataio::setted (string const & *name*)

Returns:
true if the value of this, is correctly setted.

7.3.3.58 bool dataio::setted () `[inline]`

Returns:
true if all of value is correctly setted.

Examples:
sampbinrec.cpp, samprec.cpp, sampreclist.cpp, and samptable.cpp.

De£nition at line 807 of £le dataio.h.

7.3.3.59 void dataio::setvalue (vector< vector< string > > & *item*)

set values stored in vector<vector<string> >

See also:
getvalue,operator>>, operator<<, read, write.

7.3.3.60 void dataio::startinblock (unsigned *i*) `[inline, virtual]`

method for input block preparator. The multi-block input, or table data input requires that extend dataio class and overwrite this so that this routine make block i-th preparation. (in this place, you need to set variables corresponding to i-th block.) See the samptable.cpp Note: The value of i increase in each call of startinblock() If pretend to auto validate the input, overwrite validate

See also:
istable, setted, refered, validate, startoutblock.

Examples:
sampreclist.cpp, and samptable.cpp.

De£nition at line 1190 of £le dataio.h.

7.3.3.61 bool dataio::startoutblock (unsigned *i*) `[inline, virtual]`

For multi-block output or table data output, need to overwrite this routine in way to to perform i-th data setting

See also:
istable, startinblock, validate.

Examples:
sampreclist.cpp, and samptable.cpp.

De£nition at line 1214 of £le dataio.h.

7.3.3.62 bool dataio::throwexception (bool *dec*, bool *includemember* = true)

7.3.3.63 bool dataio::throwexception () `[inline]`

if this ¤ag is true, throw exception on error default value is _DATAIODEFAULTTHROWEXCEPTION

See also:
printerrormessage, _DATAIODEFAULTTHROWEXCEPTION.

De£nition at line 715 of £le dataio.h.

7.3.3.64 vector< string > & dataio::unreferednames ()

< return (reference of) referenced name list.

7.3.3.65 void dataio::validate (unsigned *n*) `[inline, virtual]`

The data validator To validate the data, overwrite this Note that, in the multi-block input, or table data, the information as setted, refered, etc, is only for i-th block) - value of i increase in each call of validate - called only once for each value setting - the member that not tried to set value is not validated. is not validated. Will use refered() to check if spcial member is tried to validated - the main class that call operator>> call their validate

See also:
istable, setted, refered, startinbloc.

Examples:
sampreclist.cpp, and samptable.cpp.

De£nition at line 1207 of £le dataio.h.

7.3.3.66 bool dataio::validateall (bool *status*, bool *includemember* = true)

set ¤ag to decide if unrefered variables will be validated. Default is _DATAIODEFAULTVALIDATEALL

See also:
validate, _DATAIODEFAULTVALIDATEALL.

7.3.3.67 bool dataio::validateall () `[inline]`

Returns:
true is all variable is validated.

De£nition at line 766 of £le dataio.h.

7.3.3.68 ostream & dataio::write (ostream & *os*)

In testing stage: This is to output cross platoform binary data to ostream (for text stream output, use operator<<

See also:
read, reversebyteorder, operator>>, operator<<.

Examples:
sampbinrec.cpp.

7.3.4 Friends And Related Function Documentation

7.3.4.1 ostream & operator<< (ostream & *os*, dataio & *data*) `[friend]`

insertor for dataio: write data to ostream (output as CSV text)

See also:
operator>>, setvalue, getvalue, read, write.

7.3.4.2 istream & operator>> (istream & *is*, dataio & *data*) `[friend]`

extractor: read data from istream (input from CSV text)

See also:
operator<<, setvalue, getvalue, read, write.

7.3.5 Member Data Documentation

7.3.5.1 friend dataio::_dataiorec<T>

De£nition at line 1260 of £le dataio.h.

7.3.5.2 char dataio::attribseparator

con£g attrib separator for data, if it is non zero, is assumed to use attribseparator to indicate atribution for veriable. Caution: The attribseparator will be used only for - istable and columnoriented is disabled - not sub£elds (non shifted data £eld only). The attribseparator and columnseparator will toghether in the data £le! os os ognored is istable or columnoriented ¤oag is true. Default is _DATAIODEFAULTATTRIBSEPARATOR

See also:
istable, columnoriented, _DATAIODEFAULTATTRIBSEPARATOR.

De£nition at line 692 of £le dataio.h.

7.3.5.3 bool dataio::clearemptytail

if this ¤ag is true, delete empty tail of each lines default value is _-
DATAIODEFAULTCLEAREMPTYTAIL

See also:
_DATAIODEFAULTCLEAREMPTYTAIL.

De£nition at line 702 of £le dataio.h.

7.3.5.4 bool dataio::columnoriented

Column oriented data ¤ag. If true, the data is assumed as transpose of normal data. Default is _-
DATAIODEFAULTCOLUMNORIENTED. commonsly, used together istable

See also:
_DATAIODEFAULTCOLUMNORIENTED, istable.

De£nition at line 671 of £le dataio.h.

7.3.5.5 char dataio::columnseparator

The column separator of data £eld. Default is _DATAIODEFAULTCOLUMNSEPARATOR

See also:
_DATAIODEFAULTCOLUMNSEPARATOR, lineseparator.

De£nition at line 675 of £le dataio.h.

7.3.5.6 bool dataio::extendedmode

if this ¤ag is true, work in extended format support mode that permit input/output the non standard data type for example, section enabled data will contain non composed data default value is _-
DATAIODEFAULTEXTENDEDMODE

See also:
_DATAIODEFAULTEXTENDEDMODE.

De£nition at line 698 of £le dataio.h.

7.3.5.7 bool dataio::istable

table data ¤ag (if true, data will be table). Default is _DATAIODEFAULTISTABLE Commonly, the columnoriented is used together

See also:
_DATAIODEFAULTISTABLE, columnoriented.

De£nition at line 634 of £le dataio.h.

7.3.5.8 char dataio::lineseparator

Line separator of data. Default is _DATAIODEFAULTLINESEPARATOR

See also:
_DATAIODEFAULTLINESEPARATOR, _DATAIODEFAULTAUTOLINESEPARATOR , columnseparator.

De£nition at line 679 of £le dataio.h.

7.3.5.9 unsigned dataio::maxcolumnonline

If number of column exceds this number and linewrap is non empty, the operator>> perform line wrapping. Default is _DATAIODEFAULTMAXCOLUMNONLINE

See also:
_DATAIODEFAULTMAXCOLUMNONLINE, linewrap.

De£nition at line 630 of £le dataio.h.

7.3.5.10 bool dataio::parseallinputstring

if this ¤ag is true, all item is parsed (using parsestring) in the input stage to solve delimited string problem, independent of the candidate for name or not. default value is _DATAIODEFAULTPARSEALLITEMSTRING. This ¤ag is speciall that applied into all of members, independent of the map¤agstomember, in way to evit the double parsing in the stringtoitem (the default one that use it is the string type)

See also:
_DATAIODEFAULTPARSEALLINPUTSTRING.

De£nition at line 710 of £le dataio.h.

7.3.5.11 bool dataio::reversebyteorder

validate unreferenced variable?? if false, ignore validate calling for unreferenced members. if true, call validate for all members default value is _DATAIODEFAULTVALIDATEALL

See also:
validate, istartinblock, _DATAIODEFAULTVALIDATEALL used by binary input/output. If true, reverse byte order for numeric i/o default value is _DATAIODEFAULTREVERSEBYTEORDER , read, write, _DATAIODEFAULTREVERSEBYTEORDER.

De£nition at line 764 of £le dataio.h.

7.3.5.12 char dataio::scapechar

the scape char (to specify special char) for delimited string Default is _DATAIODEFAULTSCAPECHAR

See also:
stringdelimiter, _DATAIODEFAULTSCAPECHAR.

De£nition at line 754 of £le dataio.h.

7.3.5.13 char dataio::sectionnameclose

section name delimiter (used after section name) for con£g £le. Default is _-DATAIODEFAULTSECTIONNAMECLOSE If both of sectionnameopen and sectionnameclose is disabled (iswhite), data is assumed that was not section

See also:

_DATAIODEFAULTSECTIONNAMECLOSE, sectionnameopen, enablestdsection, disablesection.

De£nition at line 651 of £le dataio.h.

7.3.5.14 char dataio::sectionnameopen

section name delimiter (used before section maem) for con£g £le. '\0' is assumed inactive. Default is _DATAIODEFAULTSECTIONNAMEOPEN If both of sectionnameopen and sectionnameclose is disabled (iswhite), con£g £le is assumed that was not section

See also:

_DATAIODEFAULTSECTIONNAMEOPEN, sectionnameclose, disablesection, enablestdsection.

De£nition at line 643 of £le dataio.h.

7.3.5.15 char dataio::stringdelimiter

the string delimiter. Default is _DATAIODEFAULTSTRINGDELIMITER

See also:

scapechar, _DATAIODEFAULTSTRINGDELIMITER.

De£nition at line 749 of £le dataio.h.

The documentation for this class was generated from the following £le:

- dataio.h

Chapter 8

dataio File Documentation

8.1 dataio.h File Reference

```
#include <string>
#include <iostream>
#include <strstream>
#include <vector>
#include <errno.h>
#include "stringutil.h"
```

Compounds

- class _dataiorec

  io record template class used by dataio.

- class _dataiorecbase

  io record base class used by ndataio class. Internal use only.

- class dataio

  dataio main class to perform data input/output features The primitive data that will read/write are: bool, char, wchar_t (wchar_t work? perphaps, no), string, int, long, unsigned, unsigned long, ¤oat, double, long double.

De£nes

- #de£ne _DATAIODEFAULTCOLUMNSEPARATOR '\t'

  default value of column separator (tab separated).

- #de£ne _DATAIODEFAULTAUTOLINESEPARATOR '\0' *default value of line separator that assume standard text £le that use one of folowwing new line conventions: 'n' (unix), rn' (ansi - DOS, windows), 'r' (VAX,VMS), 'rn' (unknow (there exist?).*

- #de£ne _DATAIODEFAULTLINESEPARATOR _DATAIODEFAULTAUTOLINESEPARATOR

  default value of line separator (standard text £le).

- #de£ne _DATAIODEFAULTATTRIBSEPARATOR '\0'

  default value for con£gattribseparator (disabled) if need to read con£g £le, active this as '=' (this is the con£guration for most applications) Observe that the activation of attrib separator do not disable the use of column separator to separate variable and data.

- #de£ne _DATAIODEFAULTDECIMAL '.'

  default value of decimal charecter (as C/C++ locale).

- #de£ne _DATAIODEFAULTIGNORECASE false

  default value of ignorere case frag (case sensive).

- #de£ne _DATAIODEFAULTCOMMENTLINE "//"

  default value of comment line marker (C++ line comment).

- #de£ne _DATAIODEFAULTCOMMENTOPEN "/∗"

  default value of comment block open delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTCOMMENTCLOSE "∗/"

  default value of comment block close delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTSTDCOMMENTOPEN "/∗"

  default value of standard comment block open delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTSTDCOMMENTCLOSE "∗/"

  default value of comment block close delimiter (C/C++/java comment block marker).

- #de£ne _DATAIODEFAULTLINEWRAP ""

  default value of line wrap (append next line) marker (disabled).

- #de£ne _DATAIODEFAULTMAXCOLUMNONLINE (∼0U)

  default value of maximun column on the line. It column exceds this and line wrap is actived (!linewrap.empty()), operator>>() perform line wrapping.

- #de£ne _DATAIODEFAULTCOLUMNORIENTED false

  default value of column oriented data ¤ag (is not column oriented).

- #de£ne _DATAIODEFAULTISTABLE false

  default value of istable data ¤ag (is not the table data).

- #de£ne _DATAIODEFAULTEMPTYISVALID true

  default value of emptyisvalid data ¤ag (empty is valid value, if suitable).

- #de£ne _DATAIODEFAULTEXTENDEDMODE true

  default value of extendedmode data ¤ag (work in extended mode).

- #de£ne _DATAIODEFAULTVALIDATEALL true

  default value of validateall() (the unrefered variables is validated).

- #de£ne _DATAIODEFAULTCLEAREMPTYTAIL true

  default value for clearemptytail.

- #de£ne _DATAIODEFAULTSTRINGDELIMITER '\"'

  default value of stringdelimiter() (C/C++ java mode delimiter).

- #de£ne _DATAIODEFAULTSCAPECHAR '\0'

  default value of scapechar (disabled: for C/C++ java mode char scape sequence, use ").

- #de£ne _DATAIODEFAULTTHROWEXCEPTION false

  default value of throwexception (disabled).

- #de£ne _DATAIODEFAULTPRINTERROR true

  default value of printerror (enabled).

- #de£ne _DATAIODEFAULTCOLLECTNAMES true

  default value for refersubnames() (collect refered names).

- #de£ne _DATAIODEFAULTPARSEALLINPUTSTRING false

  default values for parseallinputstring.

- #de£ne _DATAIODEFAULTSECTIONNAMEOPEN '\0'

  Default value for sectionameopen (disabled: for windows ini £le like, use '[').

- #de£ne _DATAIODEFAULTSECTIONNAMECLOSE '\0'

  Default value for sectionameclose (disabled: for windows ini £le like, use ']').

- #de£ne _DATAIODEFAULTSTDSECTIONNAMEOPEN '['

  Default value for sectionameopen for default section mode (windows ini line).

- #de£ne _DATAIODEFAULTSTDSECTIONNAMECLOSE ']'

  Default value for sectionameclose for default section mode (windows ini £le like).

- #de£ne _DATAIODEFAULTREVERSEBYTEORDER false

  for binary input/output usage default values for reversebyteorder.

- #de£ne _DATAIODEFAULTYESLIST {"true", "on", "yes", 0}

  used by bool type input: the item on list is assumed true Note: the empty value is assumed false.

- #de£ne _DATAIODEFAULTNOLIST {"false", "off", "no", 0}

  used by bool type input: the item on list is assumed false Note: if velue is not on yes list and on false list, is assumed false.

- #de£ne _DATAIODEFAULTYESVALUE "yes"

  defalt word for true value on bool type.

- #de£ne _DATAIODEFAULTNOVALUE "no"

  defalt word for false value on bool type.

8.2 formats.txt File Reference

8.3 formatsbin.txt File Reference

8.4 samples.txt File Reference

8.5 stringutil.h File Reference

```
#include <string>
#include <iostream>
#include <vector>
```

De£nes

- #de£ne _STRINGUTILDEFAULTPRINTERROR true

  default values for stringutilprinterror
  See also:
  stringutilprinterror.

- #de£ne _STRINGUTILDEFAULTTHROWEXCEPTION false

  defaultvalues for stringutilthrowexception
  See also:
  stringutilthrowexception.

- #de£ne _STRINGUTILSTRDELIMITER '\"'

  default string delimiter char speci£cation.

- #de£ne _STRINGUTILSCAPECHAR '\\'

  default scape char speci£cation.

- #de£ne _STRINGUTILCOMMENTCHAR '#'

  default comment char speci£cation.

- #de£ne _STRINGUTILCOLUMNSEPARATOR '\t'

  default column separator.

- #de£ne _STRINGUTILCNTRLMARK '^'

  speci£cator for cntrl char.

- #de£ne _STRINGUTILHEXAMARK 'X'

  speci£cator for hexadecimal.

- #de£ne _STRINGUTILCHARTABLE

  the spceial characters speci£cation table.

Functions

- bool stringutilprinterror ()

  print error message default value is _STRINGUTILDEFAULTPRINTERROR;
  See also:
  stringutilthrowexception, *_STRINGUTILDEFAULTPRINTERROR*.

- bool stringutilprinterror (bool status)
- bool stringutilthrowexception ()

  throw exception default value is _STRINGUTILDEFAULTTHROWEXCEPTION

  See also:

  stringutilprinterror, _STRINGUTILDEFAULTPRINTERROR.

- bool stringutilthrowexception (bool status)
- bool isextended (char c)
- bool iswhite (char c)

  white char detection, used by string util and dataio.

- char tocntrl (char c)
- unsigned stringcase£nd (string const &s1, string const &s2, unsigned pos=0)

  case insensive version of string::£nd().

- unsigned stringcase£nd_last_of (string const &s1, string const &s2, unsigned pos=0)

  case insensive version of string::£nd_last_of() does not used by dataio.

- string& parsestring (string &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  clear extra spaces and parse delimited string of s the sequence of iswhite() is replaced by single space, except inside of delimited string (inside delimiters, the sequence of two delimiters is assumed one delimiters).

- vector<string>& parsestring (vector< string > &strlist, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  for each element of list, clear extra spaces and parse delimited string.

- string& reverseparsestring (string &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  translate s to delimited string (is delimiter is found, substitute by double of one).

- vector<string>& reverseparsestring (vector< string > &strlist, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  for each element of list, convert to delimited string.

- unsigned £ndstringdelimiterclose (string const &s, unsigned pos=0, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  return the position of string delimiter to close, started in the position pos. case delimiter='\0', force that delimiter is s[pos] otherwise: if s[pos] is not delimiter, assume that £rst delimiter is to open, and start closer £ndding. return ~0U if not found. return s.size() if delimiter do not occur in the s (starting the pos) CAUTION: This routine will change in future in way to support C/C++ like cher speci£cation.

- bool stringdelimiterbalanced (string const &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, unsigned pos1=0, unsigned pos2=(~0U))

  return true, if string delimiter is closed correctly between [pos1, pos2).

- bool needdelimiter (string const &s, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR)

  return true, if string delimiters is required if contain two adjacent iswhite() or string delimiters.

- istream& gettextline (istream &f, string &line, char comment=_STRINGUTILCOMMENTCHAR, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, char newlinedelimiter='\0')

  *getline() that can read text £le of ansi, DOS, windows (newline == '\r
'), UNIX (newline=='n') and VAX/VMS (newline=='r') withouth speci£cation. if (newline == '\0'), auto detect new line format. otherwise, work as getline() Unless (newline == EOF), assume end of £le if EOF character is found or £sical eof occur It is necessary, because some text editor write EOF character at end of £le If (newline == EOF), read EOF char into memory, if found.*

- istream& getline (istream &f, vector< string > &dataline, char separator=_-STRINGUTILCOLUMNSEPARATOR, char comment=_STRINGUTILCOMMENTCHAR, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, char newline='\0')

  vectorized getline get vector <string> (one line separated as colmns) from istream does not used by dataio newline=='\0' is autodetect mode.

- istream& getlines (istream &f, vector< vector< string > > &datalist, char separator=_-STRINGUTILCOLUMNSEPARATOR, char comment=_STRINGUTILCOMMENTCHAR, char strdelimiter=_STRINGUTILSTRDELIMITER, char scapechar=_STRINGUTILSCAPECHAR, char newline='\0')

  vectorized getlines newline=='\0' is auto detect mode.

- ostream& putline (ostream &f, vector< string > &dataline, char separator=_-STRINGUTILCOLUMNSEPARATOR, char newline='\0')

  put the vector<string> to ostream dataio use it, only if _DATAIODEBUG_ is de£ned newline=='\0' is system newline.

- ostream& putlines (ostream &f, vector< vector< string > > &datalist, char separator=_-STRINGUTILCOLUMNSEPARATOR, char newline='\0')

  vectorized putlines newline=='\0' is system new lines.

- vector<vector <string> >& transpose (vector< vector< string > > &data)

  transpose the vector<vector <string> > data array (replace row and column).

8.6 todo.txt File Reference

Chapter 9

dataio Example Documentation

9.1 sampattrib.cpp

Example £le for value atribution format data (like con£g £le) read/write features.

```
// dataio 0.5.4 (beta) - Copyright (C) 2001, by Sadao Massago              //
// file: sampattrib.cpp (atrib indicator data examples)                           //
// requires: dataio.h, dataio.cpp, stringutil.h, stringutil.cpp              //
// ------------------------------------------------------------------//
// The dataio library and related files is licenced under the term of    //
// GNU Lesser General Public License version 2.1 or latter              //
// (see lesser.txt for detail).                                         //
// For information over GNU or GNU compatible license, visit the site    //
// http://www.gnu.org.                                                  //

//#include <string>
#include <iostream>
#include <strstream>

// #define _DEBUG_

#include "dataio.h"
// it want to copile for single linkable file (without using makefile)
// comment above include and active follow two includes:
// #include "stringutil.cpp"
// #include "dataio.cpp"

// for main()
#include <fstream>


class cfg {
  public:
    vector <string> path;
    unsigned maxiter;
    bool printlog;
    string defaultlogfile;

    bool read(istream &f) {
      dataio io;
      // configure as unix config file like
  io.attribseparator='=';
  io.columnseparator=' ';
  // cfg variable settings.
  io.add("path", &path);
  io.add("log-file", &defaultlogfile);
  io.add("print-log-on", &printlog); // prices with name fields
  io.add("iter-limit", &maxiter); // prices with name fields
  f >> io;
  return true;
}
bool read(const char *filename) {
  ifstream f(filename);
  if(!f)
    return false;
  // cfg variable settings.
  return read(f);
}
bool write(ostream &f)
{
  dataio io;
  // configure as unix config file like
  io.attribseparator='=';
  io.columnseparator=';';
  // cfg variable settings.
  io.add("path", &path);
  io.add("log-file", &defaultlogfile);
  io.add("print-log-on", &printlog); // prices with name fields
  io.add("iter-limit", &maxiter); // prices with name fields
  // some comments
  // item comment need to add after items
  io.addcomment("This is config file like data");  // global comment
  io.addcomment("path", "the path list");
  io.addcomment("log-file", "log file name");
  io.addcomment("iter-limit", "maximum number of iteration");
  f << io;
  return true;
}
bool write(const char *filename)
{
  ofstream f(filename);
  if(!f)
    return false;
  return write(f);
}
bool writeCSV(ostream &f)
// normal output
{
  dataio io;
  // // configure as unix config file like
  // io.columnattribseparator='=';
  // io.columnseparator=' ';
  // cfg variable settings.
  io.add("path", &path);
  io.add("log-file", &defaultlogfile);
  io.add("print-log-on", &printlog); // prices with name fields
  io.add("iter-limit", &maxiter); // prices with name fields
  f << io;
  return true;
}
bool writeCSV(const char *filename)
{
  ofstream f(filename);
  if(!f)
    return false;
  return writeCSV(f);
```

```
    }
}; // cfg

//    void main(int argc, char **argv)
//    {
int main()
{

  cfg progcfg;

  if(!progcfg.read("sampattrib.txt")){
         cout << "main:error: can’t read input files"<< endl;
         return 1;
  }

      // config file reading succeeds
  cout << "configuration loaded\n";
  cout << "progcfg.path.size(): " << progcfg.path.size() << endl;
  cout << "progcfg.maxiter = " << progcfg.maxiter << endl;
  progcfg.write(cout);
  cout << "now, config values as CSV data (with separator \’\\t\’ ):" << endl;
  progcfg.writeCSV(cout);
}
```

9.2 sampattrib.txt

Example data £le for sampattrib.cpp

```
// the file is in unix config format
// the current version of dataio does not split the list
// because need to use ’=’ as column separator...
// in future, implement readcfg that allow the correct list spliting
// or implement to use two column separator (between varable name and data, and
// between two data) ??
path=/usr/local/bin ./ usr/bin // this is tree element (does not spplited)
log-file= // no values
print-log-on /* print the transaction log */
// the iteration limit is setted to 1000
iter-limit=1000
```

9.3 sampbinrec.cpp

Testing... Example £le for simple record data read/write as binary mode.

```
// dataio 0.5 (beta) - Copyright (C) 2001, by Sadao Massago              //
// file: sampbinrec.cpp (record data examples)                           //
// requires: dataio.h, dataio.cpp, stringutil.h, stringutil.cpp          //
// --------------------------------------------------------------------//
// The dataio library and related files is licenced under the term of    //
// GNU Lesser General Public License version 2.1 or latter               //
// (see lesser.txt for detail).                                          //
// For information over GNU or GNU compatible license, visit the site    //
// http://www.gnu.org.                                                   //

//#include <string>
#include <iostream>
#include <strstream>
#include <vector>

//  #define _DATAIODEBUG_

// #define _DEBUG_

#include "dataio.h"
// it want to copile for single linkable file (without using makefile)
// comment above include and active follow two includes:
 #include "stringutil.cpp"
 #include "dataio.cpp"

// for main()
#include <fstream>

//    void main(int argc, char **argv)
//    {
int main()
{

  dataio data1, data2;

  vector <vector <string> > products;
  vector <double> prices;

  vector <string> activeproducts;

  double moneyformonth;

  ifstream f("samprec.txt");
  fstream binf("samprec.bin", ios::in | ios::out);
  if(!f || !binf) {
       cout << "main:error: can’t open input files"<< endl;
       return 1;
  }

  // file open succeeds

  data1.add(&products);  // products withou field names

  // data1.add(&list3);
  data1.add("prices", &prices); // prices with name fields
  data1.additemcomment(&products, "the list of products");
```

```
data1.addcomment("prices", "the price list corresponding to product list");

data2.add("products", &data1); // data1 is sub data block for data2
data2.add("money-for-month", &moneyformonth); // one double item
// add item comment using variable reference for item
data2.additemcomment(&moneyformonth, "the cote for each month");
data2.add("responsible for", &activeproducts); // productlist with field names
// add comment using item names
data2.addcomment("responsible for", "product controled by this section");
data2.ignorecase(false); // ignore case
data2.commentline("");

#ifdef _DEBUG_
   cout << "main: readding data" << endl;
#endif
f >> data2;
cout << "main: data loaded...\n";
cout << "prices.size() =" << prices.size() << endl;
cout << "products.size() =" << products.size() << endl;

cout << "the nemes refered by data file (does not includes field names) are:\n";
putline(cout, data2.referednames());
if(!data2.refered("money-for-month")) {
  cout << endl << "The money-for-month is not on data base. ";
  cout << "Setting as 80\n";
  moneyformonth = 80;
}
else {
  cout << "Money-for-month is refered in database\n";
}

if(!data1.setted("prices"))
  cout << "prices correctly loaded\n";
else
  cout << "prices value is incorrect data\n";

#ifdef _DEBUG_
   cout << "main: writting data" << endl;
#endif
// cout << "products.size(): " << products.size() << endl;
// cout << "main: list1[0][0] = "<<list1[0][0]<<endl;
// cout << "main: x = " << x << endl;
// cout << "main: list2[0] = "<<list2[0]<<endl;
// x = strtod("2.0",NULL);
cout << "loaded data\n";
cout << data2 << endl;
cout << "now, products in columnoriented mode\n";
data1.columnoriented = true;
cout << data2;

data2.clearcomment(true);
cout << "writing to binary stream\n";
data2.write(binf);
cout << "readding from binary stream\n";
binf.seekg(0, ios::beg);
data2.read(binf);
cout << "data loaded\n";
cout << data2;

#ifdef _DEBUG_
  cout << "products: \n";
  for(int i=0; i < products.size(); i++)
    putline(cout, products[i]);
#endif
```

```
}
```

9.4 samprec.cpp

Example £le for simple record data read/write features.

```
// dataio 0.5 (beta) - Copyright (C) 2001, by Sadao Massago              //
// file: samprec.cpp (record data examples)                           //
// requires: dataio.h, dataio.cpp, stringutil.h, stringutil.cpp              //
// -------------------------------------------------------------------//
// The dataio library and related files is licenced under the term of    //
// GNU Lesser General Public License version 2.1 or latter               //
// (see lesser.txt for detail).                                          //
// For information over GNU or GNU compatible license, visit the site    //
// http://www.gnu.org.                                                   //

//#include <string>
#include <iostream>
#include <strstream>
#include <vector>

//  #define _DATAIODEBUG_

// #define _DEBUG_

#include "dataio.h"
// it want to copile for single linkable file (without using makefile)
// comment above include and active follow two includes:
// #include "stringutil.cpp"
// #include "dataio.cpp"

// for main()
#include <fstream>

//    void main(int argc, char **argv)
//    {
int main()
{

  dataio data1, data2;

  vector <vector <string> > products;
  vector <double> prices;

  vector <string> activeproducts;

  double moneyformonth;

  ifstream f("samprec.txt");
  if(!f) {
       cout << "main:error: can’t open input files"<< endl;
       return 1;
  }

  // file open succeeds

  data1.add(&products);  // products withou field names

  // data1.add(&list3);
  data1.add("prices", &prices); // prices with name fields
  data1.additemcomment(&products, "the list of products");
  data1.addcomment("prices", "the price list corresponding to product list");
  data2.add("products", &data1); // data1 is sub data block for data2
  data2.add("money-for-month", &moneyformonth); // one double item
  // add item comment using variable reference for item
  data2.additemcomment(&moneyformonth, "the cote for each month");
  data2.add("responsible for", &activeproducts); // productlist with field names
  // add comment using item names
  data2.addcomment("responsible for", "product controled by this section");
  data2.ignorecase(false); // ignore case
  data2.commentline("");

  #ifdef _DEBUG_
     cout << "main: readding data" << endl;
  #endif
  f >> data2;
  cout << "main: data loaded...\n";
  cout << "prices.size() =" << prices.size() << endl;
  cout << "products.size() =" << products.size() << endl;

  cout << "the nemes refered by data file (does not includes field names) are:\n";
  putline(cout, data2.referednames());
  if(!data2.refered("money-for-month")) {
    cout << endl << "The money-for-month is not on data base. ";
    cout << "Setting as 80\n";
    moneyformonth = 80;
  }
  else {
    cout << "Money-for-month is refered in database\n";
  }

  if(!data1.setted("prices"))
    cout << "prices correctly loaded\n";
  else
    cout << "prices value is incorrect data\n";

  #ifdef _DEBUG_
     cout << "main: writting data" << endl;
  #endif
  // cout << "products.size(): " << products.size() << endl;
  // cout << "main: list1[0][0] = "<<list1[0][0]<<endl;
  // cout << "main: x = " << x << endl;
  // cout << "main: list2[0] = "<<list2[0]<<endl;
  // x = strtod("2.0",NULL);
  cout << "loaded data\n";
  cout << data2 << endl;
  cout << "now, products in columnoriented mode\n";
  data1.columnoriented = true;
  cout << data2;

  #ifdef _DEBUG_
    cout << "products: \n";
    for(unsigned int i=0; i < products.size(); i++)
      putline(cout, products[i]);
  #endif

}
```

9.5 samprec.txt

Example data £le for samprec.cpp

```
// The products list is one column shifted, because is subfield of data.
products        paper   empty stock     buy  immediatelly!
        biscuit/* not the our responsability */ 5 on stock      discontinue if empty
        pencil  1 on stock      buy more!

        prices  1.2     1,0     0.2// check for new prices

responsible for  /* note that the biscuit does not
the responsability of this session */   paper   pencil

passed-to-other-section biscuit tea // now, it is not the our responsability
```

9.6 sampreclist.cpp

Example £le for multi-block record list read/write features.

```
// dataio 0.5 (beta) - Copyright (C) 2001, by Sadao Massago              //
// file: recordlist.cpp (record list (multi block) examples)             //
// requires: dataio.h, dataio.cpp, stringutil.h, stringutil.cpp              //
// ---------------------------------------------------------------------//
// The dataio library and related files is licenced under the term of    //
// GNU Lesser General Public License version 2.1 or latter               //
// (see lesser.txt for detail).                                          //
// For information over GNU or GNU compatible license, visit the site    //
// http://www.gnu.org.                                                   //

//#include <string>
#include <iostream>
#include <strstream>
#include <vector>

// #define _DEBUG_
// #define _DEBUG2_

#include "dataio.h"
// it want to copile for single linkable file (without using makefile)
// comment above include and active follow two includes:
// #include "stringutil.cpp"
// #include "dataio.cpp"



// for main()
#include <fstream>

   class stock
   {
   public:
      string product;
      unsigned unit;
      double price;
   };

   class stockio : public dataio {
   // vector <datarec> &data;
      vector <stock> *data;
   // datareclistio(vector <datarec> *datalist)
   // {&data = datalist;}
      bool status;
   public:
      void startinblock(unsigned i)
      {
      #ifdef _DEBUG2_
      cout << "startinblock: i = " << i  << endl;
      #endif
         if(!data)
            return;
         if(i==0) // first call
           status = true;
         clear();
         data->push_back(stock());
         add("product", &(*data)[i].product);
         add("unit", &(*data)[i].unit);
         add("price", &(*data)[i].price);
      }
      bool startoutblock(unsigned i)
      {
         #ifdef _DEBUG2_
            cout << "startoutblock: i = " << i  << endl;
         #endif
         if(!data || (i >= data->size()) )
            return false;
         clear();
         add("product", &(*data)[i].product );
         add("unit", &(*data)[i].unit);
         add("price", &(*data)[i].price);
         ostrstream ostr;
         ostr << i << "-th item" << ends;
         addcomment(ostr.str());
         return true;
      }
      void validate(unsigned i)
      {
      #ifdef _DEBUG2_
      cout << "validate: i = " << i  << endl;
      putline(cout, this->referednames());
      #endif
         if(!data || (i >= data->size()) ) { // if will not occu
            cout << "error: stockio::validate: dataio internal problem\n";
            return;
         }
         // if not correct setted, set as defaul values
         if(!refered("product")) {
           cout << "error: stockio::validate: " << i << "-th the product name missing...\n";
           status = false;
         } else if(!setted("product") || (*data)[i].product.empty()) {
           cout << "error: stockio::validate: " << i << "-th the product name is invalid...\n";
           status = false;
         }
         // if not correct setted, set as defaul values
         if(!refered("unit")) { // not refered. Assume as none
           (*data)[i].unit = 0;
         } else if(!setted("unit")) { // invalid value
           cout << "error: stockio::validate: " << i << "-th the unit value is invalid...\n";
           status = false;
         }
         if(!refered("price")) {
           cout << "warn: stockio: " << i << "-th price missing...\n";
           (*data)[i].price = 0; // o is unkonow price
           status = false;
         } else if(!setted("price")) {
           cout << "error: stockio: " << i << "-th the price is invalid...\n";
           status = false;
         }
      } // validate

      stockio():dataio() {
         data=0; status = false;}
      stockio(vector <stock> &stocklist) : dataio()
      {data = &stocklist; }
      void set(vector <stock> &stocklist)
      {data = &stocklist;}
   };


   //    void main(int argc, char **argv)
   //    {
```

```
int main()
{
   vector <stock> stocklist;

   stockio stockbase(stocklist);
   dataio io;

   ifstream f("sampreclist.txt");
   if(!f) {
      cout << "main:error: can’t open input files"<< endl;
      return 1;
   }

   // file open succeeds
   io.add("item", (dataio *)&stockbase);
   io.ignorecase(true);
   // io.colunoriented = true;
   // io.tabledata = true;
   io.decimal(’,’);

#ifdef _DEBUG_
  cout << "main: readding data" << endl;
#endif
f >> io;
cout << "main: data loaded...\n";

#ifdef _DEBUG_
  cout << "main: writting data" << endl;
#endif
// data.colunoriented = false;
// data.tabledata = false;
   cout << "loaded data...\n";
   cout << "stocklist.size() = " << stocklist.size() << endl;
   io.decimal(’.’);
   cout << io;

   cout << "now, stock as column oriented table\n";
   // note the diference of recordlist output and table output
   stockbase.clearcomment(); // clear comment setted by stockbase in above output
   stockbase.addcomment("table of product stock");
   stockbase.istable = true;
   stockbase.columnoriented = true;
   stockbase.decimal(’.’);
   cout << stockbase;
}
```

9.7 sampreclist.txt

Example data £le for sampreclist.cpp

```
// the food itens
item    product tea
        unit    3
        price   1,2
item    Product cofee
        unit    3
        price   5
// the next item omit product specification
// the validate will say: product not refered
item // product bread
        unit    3
        price   2

// is the bazar itens
item    Product notebook
        unit    2
        price   1
// the next item has invalid product value
// the price is invalid
item    product //      pencil
        unit    2
        price   xx//0,2
item    product eraser
        unit    5
        price   0,2
item    product pen
        unit    3
        price   0,8
```

9.8 sampsec.cpp

Example £le for con£g like section usage features.

```
// dataio 0.5 (beta) - Copyright (C) 2001, by Sadao Massago              //
// file: sampsec.cpp (section usage example)                           //
// requires: dataio.h, dataio.cpp, stringutil.h, stringutil.cpp              //
// ------------------------------------------------------------------//
// The dataio library and related files is licenced under the term of    //
// GNU Lesser General Public License version 2.1 or latter               //
// (see lesser.txt for detail).                                          //
// For information over GNU or GNU compatible license, visit the site    //
// http://www.gnu.org.                                                   //

//#include <string>
#include <iostream>
#include <strstream>
#include <vector>

// #define _DATAIODEBUG_

// #define _DEBUG_

#include "dataio.h"
// it want to copile for single linkable file (without using makefile)
// comment above include and active follow two includes:
// #include "stringutil.cpp"
// #include "dataio.cpp"

// for main()
#include <fstream>


class fileinfo {
  public:
  string fname;
  vector <string> path;
  string type;
};

//    void main(int argc, char **argv)
//    {
int main()
{
  string prog;

  fileinfo fi;

  unsigned runs;
  bool print_log;
  unsigned long imax;
  bool ask;

  bool nocomment;

  dataio cfg, fileinfoio, process;

  ifstream f("sampsec.txt");

  if(!f) {
        cout << "main:error: can’t open input files"<< endl;
        return 1;
     }

     // file open succeeds
     // setting io
     cfg.add("prog-name", &prog);
     cfg.add("files", &fileinfoio);
     fileinfoio.add("file-name", &fi.fname);
     fileinfoio.add("default-path", &fi.path);
     fileinfoio.add("default file type", &fi.type);

     cfg.add("process", &process);
     process.add(&runs);
     process.add("print-log-on", &print_log);
     process.add("max-iter", &imax); process.addcomment("max-iter", "iteration limit");
     process.add("ask-to-user", &ask); process.additemcomment(&ask, "if exception occur, ask to user");

     cfg.add("print-comment-off", &nocomment);

     cfg.addcomment("this is win ini like config example");
     cfg.addcomment("contain non win ini standard features");
     cfg.addcomment("print-comment-off", "no process comment printing mode");
     cfg.ignorecase(true);
     cfg.attribseparator=’=’;
     cfg.sectionnameopen = ’[’;
     cfg.sectionnameclose = ’]’;
     cfg.columnseparator=’ ’;
     cfg.extendedmode = false; // near as standard

     cout << "readding config file\n";
     f >> cfg;

     cout << "main: config loaded...\n";

     cout << "loaded data\n";
     cout << cfg << endl;
     cout << "now, cfg in CSV (with columnseparator=\’\\t\’)\n";
     cfg.ignorecase(true);
     cfg.attribseparator=’\0’; // disable
     cfg.sectionnameopen = ’\0’; // disable (need to disable both)
     cfg.sectionnameclose = ’\0’; // disable (need to disable both)
     cfg.columnseparator=’\t’;

     cout << cfg;
   }
```

9.9 sampsec.txt

Example data £le for sampsec.cpp

```
// Config file
// using section as windows ini standard, but use some non-standart
// only to demonstrate the features.
// RECOMEND to use only the standard data type.

//  config file standard is like this??
//  unix:
//    Normally is the sequence of lines in the form

// [variable]=[values]

//    where [variable] is the name of variable (will not use without names)
//    that does not contain spaces and [values] is the one value, or list of values
//    separated by space.
//    will use '\' as line wrapping (to break the long line)
//    note that the dataio is not token based and multiple space is treated as
//    multiple empty column (if use space as column separator)
//    Thus, the empty column clearing is required inside of program.
//    The section is not used
//    The comment line marker is '#' ??
//
//  windows:
//    all data list is the form

// [section]
// [variable 1]=[value 1]
// ...
// [variable n]=[value n]

//    where [section] is the section name delimited by
//    '[' and ']' (can not define variables outside of section),
//    [variable x] is variable name (can not use without names,
//    and [value x] is one value or values separated by comma (',').
//    comment line is ';' ??
//    how to make line wrapping??


// the comment line will not marked as '\\' that is C++ and java standard
// use '#' for unix like, and ';' for windows like

// this is outside section
// standard for unix, and invarid for windows
prog-name=myprog

// defining section
// invalid for unix and required by windows
[files] // starting section
file-name=myfile // ok for unix and windows
// space separated values
// standard of unix. The windows use comma (',') separated
default-path=./ /usr/local/ ./mydir
// will use delimited string as name??
"default file type"=".bak"

// changing the section. standard for windows
// unix do not use section
```

```
[process] // new section
12 // number of runs
print-log-on
max-iter=1000
ask-to-user=false

// the empty section will not used by standard of windows.
// unix will not use too (because will not use section)
[] // empty section ends section. it’s not the win ini standard
print-comment-off // it is outside section
```

9.10 samptable.cpp

Example £le for table data read/write features.

```
// dataio 0.5 (beta) - Copyright (C) 2001, by Sadao Massago             //
// file: samptable.cpp (table data examples)                              //
// requires: dataio.h, dataio.cpp, stringutil.h, stringutil.cpp           //
// -------------------------------------------------------------------//
// The dataio library and related files is licenced under the term of    //
// GNU Lesser General Public License version 2.1 or latter               //
// (see lesser.txt for detail).                                          //
// For information over GNU or GNU compatible license, visit the site    //
// http://www.gnu.org.                                                   //

#include <string>
#include <iostream>
#include <strstream>
#include <vector>

// #define _DEBUG_

#include "dataio.h"
// it want to copile for single linkable file (without using makefile)
// comment above include and active follow two includes:
// #include "stringutil.cpp"
// #include "dataio.cpp"

// for main()
#include <fstream>

class stock {
public:
  string product;
  int unit;
  double weight;
  double price;
};

class stockio : public dataio {
  // vector <datarec> &data;
  public:
  vector <stock> *data;
  // datareclistio(vector <datarec> *datalist)
  // {&data = datalist;}
  void startinblock(unsigned i)
  {
    #ifdef _DEBUG_
      cout << "startinblock: i = " << i  << endl;
    #endif
    if(!data)
      return;
    clear();
    data->push_back(stock());
    add("product", &(*data)[i].product );
    add("unit", &(*data)[i].unit);
    add("weight of unit", &(*data)[i].weight);
    add("price", &(*data)[i].price);
  }
  void validate(unsigned i)
  {
      if(!setted()) {
        cout << "warn: stockio::validate:column " << i <<
                " some value is missing or invalid\n";
      }
    }
    bool startoutblock(unsigned i)
    {
      #ifdef _DEBUG_
        cout << "startoutblock: i = " << i  << endl;
      #endif
           if(!data || (i >= data->size()) )
             return false;
           clear();

      ostrstream ostr;
      ostr << i << "-th item" << ends;
      addcomment(ostr.str());

      add("product", &(*data)[i].product );
      add("unit", &(*data)[i].unit);
      add("weight of unit", &(*data)[i].weight);
      add("price", &(*data)[i].price);
           return true;
    }
    stockio():dataio() {data=0;}
    stockio(vector <stock> &stocklist) : dataio()
    {data = &stocklist; }
    void set(vector <stock> &stocklist)
    {data = &stocklist;}
  };

  //    void main(int argc, char **argv)
  //    {
  int main()
  {
    vector <stock> food;
    vector <stock> bazar;

    stockio io(food);

    ifstream f("samptable.txt");

    if(!f) {
       cout << "main:error: can’t open input files"<< endl;
       return 1;
    }

    // file open succeeds
    // io settings
    io.ignorecase(true);
    io.columnoriented = true;
    io.istable = true;
    io.decimal(’,’);

    cout << "main: readding data" << endl;
    f >> io;
    cout << "main: data loaded. now, writing...\n";

    // io.columnoriented = false;
    // io.istable = false;
    cout << "food.size() = " << food.size() << endl;
    io.decimal(’.’);
    cout << io;

    io.set(bazar);
  // io.ignorecase(true);
  // io.columnoriented = true;
  // io.istable = true;
  io.decimal(’,’);
  f >> io;
  cout << "main: new data loaded...\n";
  cout << "bazar.size() = " << bazar.size() << endl;
  io.decimal(’.’);
  cout << io;


  cout << "now, outputting bazar as record list...\n";
  io.columnoriented = false;
  io.istable = false;
  dataio recio;
  recio.add("item", (dataio *)&io);
  cout << recio;
}
```

9.11 samptable.txt

Example data £le for sampteble.cpp

```
// the food table
Product price   unit    weight of unit
tea     1,2     3       100,0
coffe   5       3       500
bread   3       2       200

// the bazar table
Product price   unit    weight of unit
notebook        1,0     2       20,0
pencil  0,2     5       10
eraser  0,2     2       10
pen     0,8     3       5
```

9.12 samptablewithoutdataio.cpp

Example £le for simple table data read/write as string tables, (using only the stringutil)

```
// dataio 0.5 (beta) - Copyright (C) 2001, by Sadao Massago              //
// file: samptablewithoutdataio.cpp (table data examples)               //
// requires: stringutil.h, stringutil.cpp                               //
// ----------------------------------------------------------------------//
// The dataio library and related files is licenced under the term of    //
// GNU Lesser General Public License version 2.1 or latter               //
// (see lesser.txt for detail).                                          //
// For information over GNU or GNU compatible license, visit the site    //
// http://www.gnu.org.                                                   //

#include <string>
#include <iostream>
#include <fstream>
#include <vector>
#include <stdlib.h>

// This sample use only the stringutil (dataio does not used)

// #define _DEBUG_
//
#include "stringutil.h"
// it want to copile for single linkable file (without using makefile)
// comment above include and active follow include:
// #include "stringutil.cpp"

void stringlisttodoublelist(vector <string> const &l, vector <double> &x)
{
  if(!l.empty()) {
     x.clear();
     for(unsigned i=0; i<l.size(); i++)
        x.push_back(atof(l[i].c_str()));
  }
}

void stringlisttointlist(vector <string> const &l, vector <int> &x)
{
  if(!l.empty()) {
     x.clear();
     for(unsigned i=0; i<l.size(); i++)
        x.push_back(atoi(l[i].c_str()));
  }
}

//    void main(int argc, char **argv)
//    {
int main()
{
  vector <vector<string> > table;
  // vector <string> line;
  unsigned i;

  vector <string> product;
  vector <double> price;
  vector <int> unit;
  vector <double> weight;
```

```
   ifstream f("samptablewithoutdataio.txt");

   cout << "this sample read table without dataio (using only the stringutil)" << endl;
   if(!f) {
      cout << "main:error: can’t open input files"<< endl;
      return 1;
   }

   // file open succeeds
   cout << "main: readding data" << endl;
   getlines(f, table, ’\t’, ’#’, ’\0’);
   // clear empty lines
   i=0;
   while(i<table.size()) {
      if(table[i].empty())
           table.erase(&table[i], &table[i+1]);
      else i++;
   }
   transpose(table); // the data is in column oriented mode

   cout << "main: data loaded. now, converting...\n";

   cout << "table.size() = " << table.size() << endl;
   for(unsigned i=0; i<table.size(); i++) {
        // line = table[i];
        if(!table[i].empty()) {
           string name = table[i][0]; // first is name
           table[i].erase(&table[i][0], &table[i][1]); // delete name from this
           if(name == "product")
               product.swap(table[i]); // get name list
           else if (name == "price")
               stringlisttodoublelist(table[i], price); // convert to double list
           else if(name == "unit")
               stringlisttointlist(table[i], unit); // convert to int list
           else if(name == "weight of unit")
               stringlisttodoublelist(table[i], weight); // convert to double list
           else {
               cout << "unknow name: " << name << endl;
               cout << "associated data is : ";
               putline(cout, table[i]);
           }
        }
        // cout << endl;
   } // for

   // now, printing data.
   cout << "loaded data: " << endl;
   cout << "product:";
   for(i=0; i<product.size(); i++)
       cout << " " << product[i];
   cout << endl;
   cout << "price:";
   for(i=0; i<price.size(); i++)
       cout << " " << price[i];
   cout << endl;
   cout << "unit:";
   for(i=0; i<unit.size(); i++)
       cout << " " << unit[i];
   cout << endl;
   cout << "weight of unit:";
   for(i=0; i<weight.size(); i++)
       cout << " " << weight[i];
   cout << endl;

} // main()
```

9.13 samptablewithoutdataio.txt

Example data £le for samptablewithoutdataio.cpp

```
# this is simple table data
# in column oriented mode
product price   unit    weight of unit# field specifications
tea     1.2     3       100.0
coffe   5       3       500
bread   3       2       200
notebook        1.0     2       20.0
pencil  0.2     5       10
eraser  0.2     2       10
pen     0.8     3       5
```

Chapter 10

dataio Page Documentation

10.1 Todo List

Group some todo listing # in the current implementation, addition of £eld with same names will not work correctly. In future, add will replace if names exist?.

# implement partial comment input features?

# stringutil: implement function that parse the parameter option into the vector <vector <string> >, for dataio parsing.

# unknow data collector (data that is not tried to set)

Member _dataiorec::readitem(istream &is, T &item) readitem

Member _dataiorec::writeitem(ostream &os, T const &item) writeitem

Index