The CSVStream Class

The CSVStream class extends the Stream class to read & write CSV files directly to Collection Objects.

The CSVStream Class is a subclass of the Stream Class.

Translation of data involved in the CSVStream class:
CSV file literals are surrounded by quotes "". These are removed by CSVLineIn and inserted by CSVLineOut. Quotes within CSV data are represented self escaped ie: " appears as "". These are translated by the CSVLineIn and CSVLineOut methods. CSVLineOut encapsulates non-numeric fields in "" unless they already are.
CSV literal strings can contain line-end sequences. To create multi-line fields use the line-end character provided by the operating system dependant ooRexx local variable .endofline.

Methods The CSVStream Class defines:
CLOSE
CSVLINEIN
CSVLINEOUT
INIT (overrides stream class method)
OPEN

Attributes of the CSVStream Class
HEADERS~FIELD(n)~NAME
HEADERS~FIELD(n)~LITERAL
SKIPHEADERS
DELIMITER
QUALIFIER
STRIPOPTION
STRIPCHAR

Methods Inherited from the Stream Class:
ARRAYIN
ARRAYOUT
CHARIN
CHAROUT
CHARS
COMMAND
DESCRIPTION
FLUSH
LINEIN
LINEOUT
LINES
MAKEARRAY
POSITION
QUALIFY
QUERY
SAY
SEEK
STATE
SUPPLIER

Methods inherited from the Object class:
NEW (Class method)
Operator methods: =, ==, \=, ><, <>, \==
CLASS
COPY
DEFAULTNAME>
HASMETHOD
OBJECTNAME
OBJECTNAME=

REQUEST
RUN
SETMETHOD
START
STRING
UNSETMETHOD

Note: The Stream class also has available class methods that its metaclass, the Class class, defines.

CLOSE

```
>>-CLOSE-------------------------------------------------><
```

Closes the stream that receives the message. CLOSE returns READY: if closing the stream is successful, or an appropriate error message. If you have tried to close an unopened file, then the CLOSE method returns a null string (""). If you specified headersExist when you created this instance then the headers will be updated to the stream at this point if they have been changed.

NB: Whereas the stream class can be closed implicitly (for example by using Lineout with no parameters) the CSVStream class must be closed explicitly.

CSVLineIn

```
>>-CSVLINEIN---------------------------------------------><
```

Reads and returns a row of CSV data from the stream. Note that a row of data may be stored in more than one logical line of the stream. An array is returned, the nth element of which contains the nth field from the Row.

Two other attributes exist after performing a CSVLineIn

Rawdata is a String Object containing the raw text that the row consists of.
Values is a Table Object mapping field data onto field-names. This is only available if headersExist is specified as .true on the init method.

CSVLineOut

```
>>-CSVLINEOUT-(-collection-Object-)-------------------------------><
```

Writes a row of CSV data to a stream. Note that a row of data may be stored in more than one logical line of the stream. If the stream was instantiated with headersExist as .true then the collection-object may be a table or stem object mapping headers onto CSV fields. Otherwise the collection-object must be an array or a collection with a makeArray method and the nth element of the collection will be placed in the nth field of the CSV file. Any occurrences of the Nil Object are stored as null strings in the file.

NB: if the collection object is a Stem then a tail of 0 is ignored as by convention the 0 tail stores the number of tails on the stem.

INIT

```
>>-INIT(name-+----------------+-)--------------------------------><
             +-,-HeadersExist-+
```

Initializes a stream object for a stream named *name*, but does not open the stream.

The second optional parameter if passed a value of 'H' (or .true) indicates that the first row of the stream is (or is to be) a row of headers containing the names of the CSV fields. Note that header fields are case sensitive. This means that 'name' and 'Name' and 'NAME' will all be treated as separate columns.

OPEN

`Parameters are as the Stream class Open Method`

Opens the stream to which you send the message and returns READY:. If the method is unsuccessful, it returns an error message string in the same form that the DESCRIPTION method uses.
See the Stream Class Open Method for a fuller description.

Changing the behaviour of a CSVStream object.

Before issuing the Open message, you can affect the CSVStream's behaviour by setting the attribute skipHeaders to .false. This will mean that the first row returned by CSVLineIn on a CSVStream where headers exist is the header row, rather than the default behaviour which is to return the first row of data.

After issuing the OPEN message to a CSVStream which has been opened with headers exist, the class will attempt to learn the nature of the fields by analysing the data. You can teach it by setting the headers field attributes name and literal. For instance:

```
MyCsvStream~headers~field(2)~name='Height'  /* set the name of the second
                                                         field to 'Height' */
MyCsvStream~headers~field(3)~literal= .true /*  tell the stream to treat the
                          third column as literal data rather than numeric */
```

By default the delimiter CSVStream expects is a comma (after all CSV stands for Comma Separated Variables) and literals are qualified by a double inverted comma. However you can create and read files with other delimiters or qualifiers by changing the attributes delimiter and qualifier after instantiating A CSVStream object. For instance, to use ; as a delimiter and ' as a qualifier do the following:

```
MyCsvStream = .csvStream~new
MyCsvStream~delimiter=";"
MyCsvStream~qualifier="'"
```

If the attribute StripOption is set to 'L', 'T' or 'B' then data is stripped using that option before CSVLineIn inserts it in the returned array. The default of 'N' means no stripping is performed. One can specify which character to strip using the attribute stripChar which defaults to blank.

```
MyCsvStream~StripOption = 'T' /* strip trailing blanks */
```

or

```
MyCsvStream~StripOption = 'L'
MyCsvStream~stripChar = '0'   /* strip Leading zeroes  */
```

GetHeaders

Get headers will return a CSVStreamHeader object containing details of the column header names and whether they are literal values or not. Column header names that exist before the csvStream is opened are present as soon as the file is opened, but literal information will not be present till the first CSVLineIn or CSVLineOut is issued.

SetHeaders

Passed a CSVStreamHeader object will apply it to the CSVStream. Together with Get Headers this allows you to base one CSV file on another.

Attributes

SkipHeaders

See Changing the behaviour of a CSVStream object under the OPEN method

Headers

Access is available to the Field definition table for files with headers. There are two entries, NAME & LITERAL. NAME is the Name for that particular column. If LITERAL is .true then that column will be treated as a literal even if the data in it is numeric. If any entry in a column is non-numeric then the entire column is treated as a literal. See Changing the behaviour of a CSVStream object under the OPEN method for an example of accessing the table.

Delimiter

This is the character which delimits the fields (as long as it does not appear within a literal). In a standard CSV file it is a comma **,**. See Changing the behaviour of a CSVStream object under the OPEN method for an example of changing the delimiter.

QUALIFIER

The Qualifier is the character that surrounds literal fields. Delimiters that appear within literal fields are ignored. In a standard CSV file the qualifier is a double quotation mark **”**. See Changing the behaviour of a CSVStream object under the OPEN method for an example of changing the qualifier.

Examples:

Files without headers:

```
csv = .CSVStream~new('c:\MyData.csv') /* 2nd arg defaults to no headers */
/* csv~skipHeaders = .false             - UnNoOp to return header line */

csv~open('write')                              /*=File looks like this=*/
csv~CSVLineOut(.array~of('red','stop'))        /* "red","stop"         */
csv~CSVLineOut(.array~of('green','go'))        /* "green","go"         */
csv~close                                      /*======================*/

csv~open('read')                               /*=======Returns========*/
do while csv~chars > 0                         /* New record           */
   dataArr = csv~CSVLineIn                     /* field 1: red         */
   say 'New record'                            /* field 2: stop        */
   do I = 1 to dataArr~last                    /* New record           */
      say 'field' I':' dataArr[I]              /* field 1: green       */
   end                                         /* field 2: go          */
end                                            /*======================*/
csv~close
```

Files with headers:

```
csv = .CSVStream~new('c:\headered.csv',.true)
csv~open                   /* Stream class defaults to both ie:readWrite */
myTable = .table~new
myTable~put('red','colour')
myTable~put('stop','action')
csv~CSVLineout(myTable)
myTable~put('green','colour')
myTable~put('go','action')
```

```
csv~CSVLineout(myTable)
csv~close

Csv~open('read')                                /*=======Returns========*/
Do while csv~chars > 0                          /* new record           */
   Csv~csvLineIn                                /* colour: red          */
   Say 'new record'                             /* action: stop         */
   Do field over csv~values                     /* new record           */
      Say field':' csv~values~at(field)         /* colour: green        */
   End                                          /* action: go           */
End                                             /*======================*/
csv~close
```