Column-based Data Readers

**Contents**

[Introduction 1](#_Toc319314617)

[Summary 1](#_Toc319314618)

[Data-loading Scripts 2](#_Toc319314619)

[Database Collection Loading 3](#_Toc319314620)

[Excel Spreadsheet Loading 3](#_Toc319314621)

[Magik collection creation 4](#_Toc319314622)

[Shared-constants 4](#_Toc319314623)

[Bulk-loading 4](#_Toc319314624)

## Introduction

These classes are designed as tools for loading data from various column-based sources. They should form the basis of any process that uses this kind of source data, to avoid the proliferation of custom-written data-loaders in the products & projects.

Using Excel spreadsheets and CSVs can facilitate data-import, since most customers are familiar with these formats. It is also easy to export data into this format from external systems.

## Summary

These classes will read from CSV/tabbed-text files or directly from Excel XLS spreadsheets (if Microsoft Office is installed).

Features are:

* Fields can be identified by Column Header labels, using internal or external names (minor punctuation errors being ignored)
* Data-loading scripts can be driven by these reader classes, with values converted to *float, integer, string, symbol, unit\_value, boolean, date, coordinate* etc.
* Database tables can be populated:
  + values are automatically converted to appropriate field-types
  + simple parent join fields handled (using aspect value)
  + support for unit\_value fields, dates etc
* Specifications can be loaded using the same mechanism; the IDs will be generated, :spec\_entity field set, Manufacturer created and Spec Group assigned
* Magik collections (*rope, property\_list, set* etc) can be created directly from files

## Data-loading Scripts

Data is read in rows from CSV or tabbed files. A list of exemplars can be provided, and the values will be converted automatically.

rwuk\_csv\_reader.new\_from\_tabbed(file\_name, \_optional column\_exemplars)

rwuk\_csv\_reader.new\_from\_csv   (file\_name, \_optional column\_exemplars)

rdr << rwuk\_csv\_reader.

new\_from\_csv(“data.csv”,

:symbol, 0.0,0.0,“name”,1,unit\_value.new(1, :m))

\_for tag,x,y,nam,j,u \_over rdr.values()

\_loop

      # tag,x,y,nam,j,u will have been converted to appropriate values

\_endloop

rdr.records() will yield vectors of values

Rows of data can be read as *property\_lists*, with the keys being derived from the column headers:

rdr.row\_exemplar << property\_list

\_for pl \_over rdr.records()

\_loop

...

\_endloop

### Examples

See *resources/base/data*

*data.csv* can be read in various ways – see *test.magik*

*terminal\_enclosure\_load.magik* is used to load *Cabinets.csv*

## Database Collection & Specification Loading

The loaders can be used to populate database collections:

tab << v.collections[:name]

tab.rwuk\_load\_from\_tabbed(file\_name, \_optional field\_names)

tab.rwuk\_load\_from\_csv   (file\_name, \_optional field\_names)

The table will be populated from the source file.  The values will automatically be converted to the correct type (including unit\_value fields).  Usually the field names will be derived from column headers (note that columns can be in any order), but the field names can be supplied if not available.  The column headers can be internal or external field names, and minor space, case & punctuation differences will be ignored.

If the table is a Specification (in the Template view), then the loader will:

* set the :spec\_entity field
* create the object\_spec & common\_spec record pair, with the common ID being generated via the spec UVA
* the Manufacturer will be created if it does not already exist – the check is case-independent, so “unknown” & “Unknown” won’t create two Manufacturers
* a Spec Group column can be used to define the Group for each Spec, which will be created if necessary

### Notes:

1. Enumerated values will be fixed for upper/lower-case and leading/trailing space differences

### Examples

In *resources/base/data, uub\_spec.csv* is Specification data, loaded with:

tv.collections[:uub\_spec].rwuk\_load\_from\_csv(“C:\data\uub\_spec.csv”)

or the bulk-loader can be used (see below):

rwuk\_csv\_reader.rwuk\_load\_from\_directory(tv, “C:\data”)

## Excel Spreadsheet Loading

Instead of writing out worksheets to CSVs, Excel spreadsheets can be loaded directly via the OLE mechanism:

rdr << rwuk\_excel\_reader.new\_from(file\_name)

The first worksheet will be used, or another can be selected using:

  rdr.sheet\_number << 2

or

  rdr.sheet\_name   << “tag”

The values() and records() iterator methods can be used as before.

Database collections and Specifications can be loaded from Excel spreadsheets with:

  ds\_collection.rwuk\_load\_from\_excel(file\_name, \_optional sheet\_name\_or\_number)

Field names are defined in column headers, as before.

### Examples

tv.collections[:copper\_cable\_spec].rwuk\_load\_from\_csv(“C:\data\copper\_cable\_spec.xlsx”)

### Notes:

1. The OLE mechanism yields all numeric values as floats, so you may need to convert them to integers if required, with integer.check\_type(value).

## Magik collection creation

Various Magik collections (keyed or not) can be created from source data files. Supported classes (plus sub-classes) are:

property\_list, hash\_table

equality\_property\_list, equality\_hash\_table

rope, simple\_vector

set, equality\_set, sorted\_collection

### Examples:

x << set.rwuk\_new\_from\_tabbed(file\_name, \_optional exemplars)

p << property\_list.rwuk\_new\_from\_csv(file\_name, \_optional exemplars)

z << hash\_table.rwuk\_new\_from\_excel(file\_name, sheet\_tag, \_optional exemplars)

### Note:

1. The keys will be symbols for identity-keyed collections, and strings for equality-keyed collections
2. The key is not included in the exemplars list
3. If the source file has multiple columns, then the collection elements will be simple-vectors; if only a single column is present, then the elements will be narrowed to a single value.

## Shared-constants

The data-loading classes can be used to initialise shared-constants from data files at build time. Instead of listing values in the Magik source code (where large blocks can break the compiler), they can be maintained in CSVs and loaded during the build process, like this:

\_pragma(classify\_level=restricted, topic={rwuk}, usage=external)

app\_landbase\_object.define\_shared\_constant(

:feature\_codes,

## Keyed-list of Feature-Codes and Descriptions

\_block

# load Feature Codes from data file

\_local file\_name <<

smallworld\_product.get\_data\_file("feature\_codes.csv",

:app\_datamodel)

\_local feature\_codes << equality\_hash\_table.

rwuk\_new\_from\_csv(file\_name)

# remove the header line entry

feature\_codes.remove\_key("CODE")

>> feature\_codes

\_endblock,

:public)

## Bulk-loading

rwuk\_csv\_reader.rwuk\_load\_from\_directory()

This loads all the XLS, CSV or TXT files in a directory, optionally controlled by a *file\_list.txt* file if the order is important, or iof only certain files are required

rwuk\_excel\_reader.rwuk\_load\_from\_spreadsheet()

This loads all the worksheets in a spreadsheet into collections defined by the worksheet tab names.