The stdemo Package
for starray version 1.9b

Alceu Frigeri*

June 2025

Abstract

As an example of how a package writer could use the *starray* package, this documents a demo package, *stdemo*, which defines a set of commands aiming to describe a set of activities and students' work associated with them. A companion document, *stdemodoc*, shows how to use these commands.

Contents

1 Introduction

The purpose of this is to create an example of how to use a *starray* in a complete setup. That for, this demo has a few parts: 1. A companion package *stdemo.sty*, 2. this document which documents not just the user level functions/commands, but also how the the companion package was created, and 3. a document using the demo package.

> ***Note:*** About the version number, since this is "part" of *starray*, and to keep tracking simple, the same version number (from *starray*) will be used.

2 Data Model

As an example, let's define two structures:

1. "Activities" (like a term project, course project, etc.) with associated elements:
   (a) name, acronym
   (b) coordinator
   (c) calendar events (presentation dates, exams...)
   (d) a check list.
2. Students, with associated date:
   (a) name, IDs, etc.
   (b) advisor and (perhaps) co-advisor.

*https://github.com/alceu-frigeri/starray/tree/main/demo

(c) reviewer(s).

(d) flags, etc.

Note: As in any "procedural language", one is advised to carefully design the data model, since this will shape the functions which will set and use said data.

Note: Pay attention to the use of the tildes, ~ , since those definitions will be made, most likely, in an *expl3* code régime, one has to remember that spaces are ignored, therefore, if needed, one has to explicitly use a tilde instead of a space.

2.1 Student Set

A student's structure might contain, besides student's name, work title, some flags, an advisor (and co-advisor, if needed), reviewer's list (with a provision for reviewer's grade, if needed).

Of course, one doesn't need to define a *starray* structure using `\starray_def_from_keyval:nn`, but, as in this, if the set of properties is known, it always makes for a cleaner definition.

Note: The fields/properties defaults can be anything, including usual LaTeX 2ε commands, like a `\rule` which is handy, for instance, when generating forms, e.g., if the fields are all set, a form can be created with the proper values, otherwise, it will be created with "rules" in place (no need to test if the properties were set).

```
\starray_new:n {student}
\starray_def_from_keyval:nn {student} {
  self = , %% this shall be self hash (if any)
  first = ,
  last = ,
  name = \rule{\l__stdemo_name_rule_dim}{.1pt} ,
  ID    = \rule{\l__stdemo_ID_rule_dim}{.1pt} ,
  email = \rule{\l__stdemo_email_rule_dim}{.1pt} ,
  worktitle = \rule{\l__stdemo_worktitle_rule_dim}{.1pt} ,
  remarks = ,
  board-local = {~local/place~} ,
  board-date   = {~date~} ,
  board-time  = {~time~} ,
  gradeavrg = 0,
  grade = ,
  flag-null = \c_false_bool , %% IF no grade was given
  flag-graded = \c_false_bool , %%% IF gradeavrg AND finalgrade already calculated (or defined)
  flag-approved = \c_false_bool ,
  flag-coadvisor = \c_false_bool ,
  advisor . struct = {
    first = ,
    last =  ,
    name = \rule{\l__stdemo_name_rule_dim}{.1pt},
    institution = \rule{\l__stdemo_name_rule_dim}{.1pt},
    title = \rule{\l__stdemo_title_rule_dim}{.1pt} ,
    email = \rule{\l__stdemo_email_rule_dim}{.1pt} ,
  } ,
  coadvisor . struct = {
    first = ,
    last =  ,
    name = \rule{\l__stdemo_name_rule_dim}{.1pt},
    institution = \rule{\l__stdemo_name_rule_dim}{.1pt},
    title = \rule{\l__stdemo_title_rule_dim}{.1pt} ,
    email = \rule{\l__stdemo_email_rule_dim}{.1pt} ,
  } ,
  reviewer . struct = {
    first = ,
    last =  ,
    name = \rule{\l__stdemo_name_rule_dim}{.1pt},
    institution = \rule{\l__stdemo_name_rule_dim}{.1pt},
    title = \rule{\l__stdemo_title_rule_dim}{.1pt} ,
    email = \rule{\l__stdemo_email_rule_dim}{.1pt} ,
    pointA = ,
    pointB = ,
    pointC = ,
    pointD = ,
    grade = 0 ,
    flag-set = \c_false_bool ,
  } ,
}
```

2.2 Activity Set

For the activities one could set an "starray" as follow:

```
\starray_new:n {activity}
\starray_def_from_keyval:nn {activity} {
    name = Activity's~ name ,
    acronym = ACRO ,
    coord . struct =  {
        name = Coordinator's~ name,
        title = Coordinator's~ title ,
      } ,
    calendar . struct = {
        date = {-day-} ,
        week = {-week-} ,
        event = {-event-} ,
      } ,
    chkID = ,         %%% 'unique ID' for checklists
    chkmarked = ,     %%% This shall be a prop list of   marked itens
    chkunmarked = ,   %%% This shall be a prop list of unmarked itens
    chkref = ,        %%% This shall be a prop list of ref      itens
  }
```

Whereas, the "coord" sub-structure is for the activity's coordinator, whilst "calendar" shall (for instance) contains a list of calendar events, and, finally, the many "chk* " will be used for a "check list".

> ***Note:*** The "chkID" (and checklists). In many cases it's handy to have an unique identifier for a given structure. That can be obtained with `\starray_get_unique_ID:nN`, and to avoid calling this function time and time again, one can just store that ID as a field for later use.
>
> ***Note:*** Could the Coordinator's name and title be a direct property (dismissing the "coord" sub-structure)? of course, that's a matter of taste/choice, on how to model it.

3 Auxiliary Functions

Once the data layout is set (see 2) the next step is to define a set of (document level) functions, so the data can be initialized and used by the end user.

3.1 Generic Recovery Functions

`\DataField` `\DataField` {⟨starray⟩} {⟨item⟩}
`\DataGet` `\DataGet` {⟨starray⟩} {⟨item⟩} {⟨store-var⟩}

`\DataField` will recovery an item from any *starray*, for instance, ⟨starray⟩ might be ⟨activity⟩, ⟨activity.coord⟩, ⟨activity.calendar⟩, ⟨student⟩ or ⟨student.advisor⟩ etc. whilst ⟨item⟩ might be any corresponding field. The `\DataGet` will store said value in an auxiliary ⟨store-var⟩.

```
\NewDocumentCommand{\DataField }{mm}{
  \starray_get_prop:nn{#1}{#2}
}

\NewDocumentCommand{\DataGet}{mmm}{
  \starray_get_prop:nnN{#1}{#2}{#3}
}
```

3.2 Activity's Functions

One could define a single function to initialize all fields (using a key=val interface), but, in a more traditional approach one can set two functions to start the initialization process `\NewActivity` and `\ActivitySet`.

\NewActivity \NewActiviy {⟨act-ID⟩}
\ActivitySet \ActivitySet [⟨act-ID⟩] {⟨name⟩} {⟨acronym⟩}

\NewActivity will create a new one, ⟨act-ID⟩ will be the identifier of it. \ActivitySet will set the ⟨name⟩ and ⟨acronym⟩ of an activity. If not given, the current ⟨act-ID⟩ will be used.
The idea is to (normally) use one right after the other, though, once created with \NewActivity, an activity can be initialized/changed at a later point using the optional argument from \ActivitySet.

> ***Note:*** Every time a *starray* is instantiated, up to two hashes are created: a numerical one (starting at one) and an "user defined one". In the \NewActivity function above, the given argument is that hash, so the just created instance can be later referenced by it. Of course, it must be an unique ID/hash.
>
> ***Note:*** One thing to be noticed about *starrays*: every structure has an associated internal index (iterator). When you create a new instance, this iterator always points to the newly created one, therefore, sparing the use of an explicit index.

```
\NewDocumentCommand{\NewActivity}{m} {
    \starray_new_term:nn {activity}{#1}
    \starray_new_term:nn {activity.coord}{}
    \starray_get_unique_id:nNTF {activity} \l__stdemo_tmpID_tl
      {}
      {}
    \starray_gset_prop:nnV {activity}{chkID} \l__stdemo_tmpID_tl
    \prop_new_linked:c {l__stdemo_ \l__stdemo_tmpID_tl _chkmarked_prop}
    \prop_new_linked:c {l__stdemo_ \l__stdemo_tmpID_tl _chkunmarked_prop}
    \prop_new_linked:c {l__stdemo_ \l__stdemo_tmpID_tl _chkref_prop}
}

\NewDocumentCommand{\ActivitySet}{O{}mm} {
  \tl_if_blank:nTF {#1}
    {
      \starray_set_prop:nnn {activity}{name}{#3}
      \starray_set_prop:nnn {activity}{acronym}{#2}
    }
    {
      \starray_set_prop:nnn {activity[#1]}{name}{#3}
      \starray_set_prop:nnn {activity[#1]}{acronym}{#2}
    }
}
```

Similarly, one can define some functions to set the activity's coordinator. Of course, it's up to the package programmer to choose if one, two (or more) functions for this.

\ActivitySetCoord \ActivitySetCoord [⟨act-ID⟩] {⟨name⟩}
\ActivitySetCoordTitle \ActivitySetCoordTitle [⟨act-ID⟩] {⟨title⟩}

For both, \ActivitySetCoord and \ActivitySetCoordTitle, the optional argument ⟨act-ID⟩ refers to an already create activity, and, if not given, will use the current one.

```
\NewDocumentCommand{\ActivitySetCoord}{O{}mO{}}{
  \tl_if_blank:nTF {#1}
    {
      \starray_gset_prop:nnn {activity.coord}{name}{#2}
    }
    {
      \starray_gset_prop:nnn {activity[#1].coord}{name}{#2}
    }
}

\NewDocumentCommand{\ActivitySetCoordTitle}{O{}m} {
  \tl_if_blank:nTF {#1}
    { \starray_set_prop:nnn {activity.coord}{title}{#2} }
    { \starray_set_prop:nnn {activity[#1].coord}{title}{#2} }
}
```

And the associated "Calendar Events", assuming there will be a fixed set of events (each semester/year), leaving the date to be set later on.

\ActivitySetNewEvent `\ActivitySetNewEvent` [⟨act-ID⟩] {⟨event-ID⟩} {⟨description⟩}
\ActivitySetEventDay `\ActivitySetEventDay` [⟨act-ID⟩] {⟨event-ID⟩} {⟨date⟩} {⟨week⟩}

The optional argument ⟨act-ID⟩ refers to an already create activity, and, if not given, will use the current one. ⟨event-ID⟩ can be any identifier. That way, the user can first define a set of events, and only later on, set the associated dates.

```
\NewDocumentCommand{\ActivitySetNewEvent}{O{}mm}{
  \tl_if_blank:nTF {#1}
    {
      \starray_new_term:nn {activity.calendar}{#2}
      \starray_gset_prop:nnn {activity.calendar}{event}{#3}
    }
    {
      \starray_new_term:nn {activity[#1].calendar}{#2}
      \starray_gset_prop:nnn {activity[#1].calendar}{event}{#3}
    }
}
```

```
\NewDocumentCommand{\ActivitySetEventDay}{O{}mmm}{
  \tl_if_blank:nTF {#1}
    {
      \starray_gset_from_keyval:nn {activity.calendar[#2]}
        {
            date = {#3} ,
            week = {#4} ,
        }
    }
    {
      \starray_gset_from_keyval:nn {activity[#1].calendar[#2]}
        {
            date = {#3} ,
            week = {#4} ,
        }
    }
}
```

In many cases, it's desirable to have a "check list". What such list could entice is always up to debate, the idea behind the few next functions is to allow the end user to define which items such a list (as a matrix) might have.

\checkdef `\checkdef` {⟨chkID⟩} {⟨chkPos⟩} {⟨chktext⟩}
\checklist `\checklist` [⟨act-ID⟩] {⟨chkID-list⟩}

⟨chkID⟩ is just an ID to reference the check list item. ⟨chkPos⟩ will relate the item to a position in a matrix (tabular environment, see `\StudentCheckListTabLine`) and ⟨chktext⟩ is the (assumed) short text. The command `\checkdef` defines/create a new item, whilst `\checklist` sets a list of ⟨chkID⟩s (note that ⟨chkID-list⟩ is a csv list).

> ***Note:*** In the implementation below, note that the check list items are associated with an activity, but the final list itself is a student by student one. Better said, each student will have a property list (constructed based on a student's unique ID) of the "checked items" (see `\StudentCheckListTabLine`).

```
\NewDocumentCommand{\checkdef}{mmm}{
    \starray_get_prop:nnN {activity}{chkID}\l__stdemo_chkID_tl
    \prop_gput:cnn {l__stdemo_ \l__stdemo_chkID_tl _chkmarked_prop} {#1}{\__stdemo_checkedbox:~\ #3}
    \prop_gput:cnn {l__stdemo_ \l__stdemo_chkID_tl _chkunmarked_prop} {#1}{\__stdemo_uncheckedbox:~
      \ #3}
    \prop_gput:cnn {l__stdemo_ \l__stdemo_chkID_tl _chkref_prop} {#2}{#1}
  }

\NewDocumentCommand{\checklist}{O{}m}{
    \tl_if_blank:nF {#1}
      {  \starray_set_iter_from_hash:nn {activity}{#1}  }

    \starray_get_prop:nnN {student}  {chkID} \l__stdemo_chkIDa_tl
    \starray_get_prop:nnN {activity} {chkID} \l__stdemo_chkIDb_tl

    \clist_map_inline:nn {#2}
      {
        \prop_get:cnNT {l__stdemo_ \l__stdemo_chkIDb_tl _chkref_prop} {##1} \l__stdemo_checkref_tl
          {
            \prop_get:ceN {l__stdemo_ \l__stdemo_chkIDb_tl _chkmarked_prop} {\l__stdemo_checkref_tl}
       \l__stdemo_checkB_tl
            \prop_gput:cee {l__stdemo_ \l__stdemo_chkIDa_tl _checklist_prop} {\l__stdemo_checkref_tl
      } {\l__stdemo_checkB_tl}
          }
      }
  }
```

3.2.1 Recovering/Using Activity's Data

`\ActivitySelect` `\ActivitySelect` {⟨act-ID⟩}

This will just select an activity, identified by ⟨act-ID⟩ as the current one. So that, in the following commands, one can avoid the first, optional, argument.

```
\NewDocumentCommand{\ActivitySelect}{m}
  {
    \starray_set_iter_from_hash:nn {activity}{#1}
  }
```

`\Activity` `\Activity` [⟨act-ID⟩] {⟨field⟩}
`\ActivityCoord` `\ActivityCoord` [⟨act-ID⟩] {⟨field⟩}

`\Activity` will recover the associated ⟨field⟩ value (from 2.2 it can (reasonably) be ⟨name⟩ or ⟨acronym⟩). `\ActivityCoord`, similarly, will recover the associated ⟨field⟩ value for the activity's coordinator (field can be, from 2.2, ⟨name⟩ or ⟨title⟩).

```
\NewDocumentCommand{\Activity}{O{}m}{
  \tl_if_blank:nTF {#1}
    { \starray_get_prop:nn {activity}{#2} }
    { \starray_get_prop:nn {activity[#1]}{#2} }
}

\NewDocumentCommand{\ActivityCoord}{O{}m}{
  \tl_if_blank:nTF {#1}
    { \starray_get_prop:nn {activity.coord}{#2} }
    { \starray_get_prop:nn {activity[#1].coord}{#2} }
}
```

`\ActivityCalendarIterate` `\ActivityCalendarIterate` {⟨code⟩}

This is a helper function, so that the end user is free to construct an "Event Calendar" with the (activity's) stored data. The suggested pattern is: 1. Select an activity with `\ActivitySelect`, then 2. execute the code for each item stored in the activity's calendar list.. The user is supposed to use (in ⟨code⟩) `\DataField` or `\DataGet` to retrieve and use the calendar's data.

```
\NewDocumentCommand{\ActivityCalendarIterate}{m}{
    \starray_iterate_over:nn{activity.calendar}{#1}
}
```

3.3 Student's Functions