The starray Package
Version 1.8
Alceu Frigeri*
March 2024

Abstract

This package implements vector like ‘structures’, alike ‘C’ and other programming languages. It’s based on expl3 and aimed at ‘package writers’, and not end users. The provided ‘functions’ are similar the ones provided for property (or sequence, or token) lists. Most of the provided functions have a companion ‘branching version’.

Contents
1 Introduction 1
2 Package Options 2
3 Creating a starray 2
  3.1 Conditionals 3
4 Defining and initialising a starray structure 3
  4.1 Fixing an ill-instantiated starray 4
5 Instantiating starray terms 4
  5.1 referencing terms 5
  5.2 iterators 7
6 Changing and recovering starray properties 10
7 Additional Commands and Conditionals 11
8 Showing (debugging) starrays 12

1 Introduction

The main idea is to have an array like syntax when setting/recovering structured information, e.g. \texttt{\starray{\{\langle\text{student}\[2\].work\[3\].reviewer\[4\]\rangle\} \{\langle\text{name}\rangle\}}} where "student" is the starray root, "work" is a sub-structure (an array in itself), "reviewer" is a sub-structure of "work" and so on, \langle\text{name}\rangle\} being a property of "reviewer". Moreover one can iterate over the structure, for instance \texttt{\starray{\{\langle\text{student\cdotwork\cdotreviewer}\rangle\} \{\langle\text{name}\rangle\}}} is also a possible reference in which one is using "student’s", "work’s" and "reviewer’s" iterators.

Internally, a starray is stored as a collection of property lists. Each starray can contain a list of property pairs (key/value as in any expl3 property lists) and a list of sub-structures. Each sub-structure, at it’s turn, can also contain a list of property pairs and a list of sub-structures.

The construction/definition of a starray can be done piecewise (a property/sub-structure at a time) or with a keyval interface or both, either way, one has to first “create a root starray” (\texttt{\starray{\{\}}}), define it’s elements (properties and sub-structures), then instantiate them "as needed". An instance of a starray (or one of it’s sub-structures) is referred, in this text, as a "term".

*https://github.com/alceu-frigeri/starray
Finally, almost all defined functions have a branching version, as per expl3 T, F and TF (note: no \_p variants, see below). For simplicity, in the text below only the TF variant is described, as in \arraynew{nTF}, keep in mind that all 3 variants are defined, e.g. \arraynew{nT}, \arraynew{nF} and \arraynew{nTF}.

**Note:** Could it be implemented with a single property list? It sure could, but at a cost: 1. complexity; 2. access time. The current implementation, albeit also complex, tries to reach a balance between inherent structure complexity, number of used/defined auxiliary property lists and access time.

**Important:** Expandability, unfortunately most/all defined functions are not “fully expandable”, in particular, most conditional/branching functions aren’t, with just a few exceptions (marked with a star ★, as per expl3 documentation convention).

### 2 Package Options

The package options (key=value) are:

- **prefix**
  - (default: \_\_starray\_). Set the prefix used when declaring the property lists associated with any \array.

- **msg-err**
  - By default, the \array package only generates "warnings", with msg-err one can choose which cases will generate "package error" messages. There are 3 message classes: 1. strict relates to \arraynew cases (\array creation); 2. syntax relates to "term syntax" errors (student.work.reviewer in the above examples); finally 3. reference relates to cases whereas the syntax is correct but referring to non-existent terms/properties.

  - none: (default) no package message will raise an error.
  - strict: will raise an error on strict case alone.
  - syntax: will raise an error on strict and syntax cases.
  - reference: will raise an error on strict, syntax and reference cases.
  - all: will raise an error on all cases.

- **msg-suppress**
  - ditto, to suppress classes of messages:
    - none: (default) no package message will be suppressed.
    - reference: only reference level messages will be suppressed.
    - syntax: reference and syntax level messages will be suppressed.
    - strict: reference, syntax and strict level messages will be suppressed.
    - all: all messages will be suppressed.

### 3 Creating a starray

\arraynew{n} \arraynew{nTF}

Creates a new \array or raises a warning if the name is already taken. The declaration (and associated property lists) is global. The given name is referred (in this text) as the \arrayroot or just \arrayroot.

**Note:** A warning is raised (see 2) if the name is already taken. The branching version doesn’t raise any warning.
3.1 Conditionals

\starray_if_exist_p:n ★ \starray_if_exist_p:n \{\langle \text{starray} \rangle \}
\starray_if_exist:n \tf \{\langle \text{starray} \rangle \} \{\langle \text{if-true} \rangle \} \{\langle \text{if-false} \rangle \}
\starray_if_valid_p:n ★ \starray_if_valid:n \tf \{\langle \text{starray} \rangle \} \{\langle \text{if-true} \rangle \} \{\langle \text{if-false} \rangle \}

\starray_if_exist:n \tf only tests if \text{starray} (the base property) is defined. It doesn’t verifies if it really is a \text{starray}. \starray_if_valid:n \tf further tests if an internal boolean (is\textunderscore starray) is also defined. This doesn’t necessary mean it is a \text{starray} or if it’s a really valid one (chances are that it is, but...), see \texttt{\starray_term_syntax:n} \tf, section 7, for a more reliable validity test.

Note: The predicate versions, \_p, expand to either \texttt{\prg_return_true:} or \texttt{\prg_return_false:}.

4 Defining and initialising a \text{starray} structure

\starray_def_prop:nnn ★ \starray_def_prop:nnn \{\langle \text{starray-ref} \rangle \} \{\langle \text{prop-key} \rangle \} \{\langle \text{initial-value} \rangle \}
\starray_def_prop:nnn \tf \{\langle \text{starray-ref} \rangle \} \{\langle \text{prop-key} \rangle \} \{\langle \text{initial-value} \rangle \} \{\langle \text{if-true} \rangle \} \{\langle \text{if-false} \rangle \}

Adds an entry, \langle \text{prop-key} \rangle, to the \langle \text{starray-ref} \rangle (see 5.1) definition and set its initial value. If \langle \text{prop-key} \rangle is already present its initial value is updated. Both \langle \text{prop-key} \rangle and \langle \text{initial-value} \rangle may contain any \langle \text{balanced text} \rangle. \langle \text{prop-key} \rangle is an (expl3) property list \langle \text{key} \rangle meaning that category codes are ignored.

The definition/assignment of a \langle \text{prop-key} \rangle to a \langle \text{starray-ref} \rangle is global.

Note: A warning is raised (see 2) in case of a \langle \text{starray-ref} \rangle syntax/reference error. The branching version doesn’t raise any warning.

\starray_def_structure:nn ★ \starray_def_structure:nn \{\langle \text{starray-ref} \rangle \} \{\langle \text{struct-name} \rangle \}
\starray_def_structure:nnn \tf \{\langle \text{starray-ref} \rangle \} \{\langle \text{struct-name} \rangle \} \{\langle \text{if-true} \rangle \} \{\langle \text{if-false} \rangle \}

Adds a sub-structure (a \text{starray} in itself) to \langle \text{starray-ref} \rangle (see 5.1). If \langle \text{struct-name} \rangle is already present nothing happens. The definition/assignment of a \langle \text{struct-name} \rangle to a \langle \text{starray-ref} \rangle is global.

Note: Do not use a dot when defining a (sub-)structure name, it might seems to work but it will breaks further down (see 5.1).

Note 2: A warning is raised (see 2) in case of a \langle \text{starray-ref} \rangle syntax error. The branching version doesn’t raise any warning.

\starray_def_from_keyval:nn ★ \starray_def_from_keyval:nn \{\langle \text{starray-ref} \rangle \} \{\langle \text{keyval-lst} \rangle \}
\starray_def_from_keyval:nnn \tf \{\langle \text{starray-ref} \rangle \} \{\langle \text{keyval-lst} \rangle \} \{\langle \text{if-true} \rangle \} \{\langle \text{if-false} \rangle \}

Adds a set of \langle \text{keys} \rangle / \langle \text{values} \rangle and/or \langle \text{structures} \rangle to \langle \text{starray-ref} \rangle (see 5.1). The \langle \text{keyval-lst} \rangle is pretty straightforward, the construction \langle \text{key} \rangle. \text{struct} denotes a nested structure:
\starray_def_from_keyval:nn \{root.substructure\}
{
  keyA = valA ,
  keyB = valB ,
  subZ . struct =
  {
    keyZA = valZA ,
    keyZB = valZB ,
  }
  subY . struct =
  {
    keyYA = valYA ,
    keyYB = valYB ,
    subYYY . struct =
    {
      keyYYYa = valYYYa ,
      keyYYYb = valYYYb
    }
  }
}

The definitions/assignments to ⟨starray-ref⟩ are all global.

Note: The non-branching version raises a warning (see 2) in case of a ⟨starray-ref⟩ syntax error. The branching version doesn’t raise any warning. Also note that, syntax errors on the ⟨keyval-lst⟩ might raise low level (\TeX) errors.

4.1 Fixing an ill-instantiated starray

When instantiating (see 5) a starray, the associated structured will be constructed based on it’s “current definition” (see 4). A problem that migh arise, when one extends the definition of an already instantiated starray (better said, if one adds a sub-structure), is a quark loop (from \l3quark). To avoid a quark loop it is necessary to ”fix” the structure of already instantiated terms.

\starray_fix_terms:n  \starray_fix_terms:n \{(starray-ref)\}

The sole purpose of this function is to ”fix” the already instantiated terms of a starray. Note, this can be an expensive operation depending on the number of terms (it has to craw over all the terms of an instantiated starray adding any missing sub-structure references), but one doesn’t need to run it ”right away” it is possible to add a bunch of sub-structures and than run this just once.

5 Instantiating starray terms

\starray_new_term:n  \starray_new_term:n \{(starray-ref)\}
\starray_new_term:nn \starray_new_term:nn \{(starray-ref)\}\{\{hash\}\}
\starray_new_term:nTF \starray_new_term:nTF \{(starray-ref)\}\{\{if-true\}\}\{\{if-false\}\}
\starray_new_term:nnTF \starray_new_term:nnTF \{(starray-ref)\}\{\{hash\}\}\{\{if-true\}\}\{\{if-false\}\}

This create a new term (in fact a property list) of the (sub-)structure referenced by ⟨starray-ref⟩. Note that the newly created term will have all properties (key/values) as defined by the associated \starray_prop_def:nn \{(starray-ref)\}, with the respective ”initial values”. For instance, given the following
One will have created 6 terms:

1. 2 \(\langle\text{st-root}\rangle\) terms
   a. the first one with index 1 and
      i. 2 sub-structures \(\langle\text{subZ}\rangle\) (indexes 1 and 2)
      ii. 1 sub-structure \(\langle\text{subY}\rangle\) (index 1)
   b. the second one with indexes 2 and "hash-A" and
      i. 1 sub-structure \(\langle\text{subZ}\rangle\) (index 1)

Note that, in the above example, it was used the "implicit" indexing (aka. iterator, see 5.1). Also note that no term of kind \(\langle\text{subYYY}\rangle\) was created.

Note: A warning is raised (see 2) in case of a \(\langle\text{starray-ref}\rangle\) syntax error. The branching version doesn’t raise any warning.

### 5.1 referencing terms

When typing a \(\langle\text{starray-ref}\rangle\) there are 3 cases to consider:

1. structure definition
2. term instantiation
3. getting/setting a property

The first case is the simplest one, in which, one (starting by \(\langle\text{starray-root}\rangle\)) will use a construct like \(\langle\text{starray-root}\rangle.\langle\text{sub-struct}\rangle.\langle\text{sub-struct}\rangle\)… For example, an equivalent construct to the one shown in 5:
Note that, all it’s needed in order to be able to use \(\text{starray-root}\). \(\langle\text{sub-A}\rangle\) is that \(\langle\text{sub-A}\rangle\) is an already declared sub-structure of \(\text{starray-root}\). The property definitions can be made in any order.

In all other cases, term instantiation, getting/setting a property, one has to address/reference a specific instance/term, implicitly (using iterators) or explicitly using indexes. The general form, of a \(\langle\text{starray-ref}\rangle\), is:

\[
\text{starray-root}\langle\text{idx}\rangle, \langle\text{sub-A}\rangle\langle\text{idxA}\rangle, \langle\text{sub-B}\rangle\langle\text{idxB}\rangle
\]

In the case of term instantiation the last \(\langle\text{sub-}\rangle\) cannot be indexed, after all one is creating a new term/index. Moreover, all \(\langle\text{idx}\rangle\) are optional like:

\[
\text{starray-root}\langle\text{sub-A}\rangle\langle\text{idxA}\rangle, \langle\text{sub-B}\rangle
\]

in which case, one is using the ”iterator” of \(\text{starray-root}\) and \(\langle\text{sub-B}\rangle\) (more later, but keep in mind the \(\langle\text{sub-B}\rangle\) iterator is the \(\langle\text{sub-B}\rangle\) associated with the \(\langle\text{sub-A}\rangle\langle\text{idxA}\rangle\)).

Since one has to explicitly instantiate all (sub)terms of a starray, one can end with a highly asymmetric structure. Starting at the \(\text{starray-root}\) one has a first counter (representing, indexing the root structure terms), then for all sub-strutures of \(\text{starray-root}\) one will have an additional counter for every term of \(\text{starray-root}\)!

So, for example:

\[
\text{starray-new}\langle\text{st-root}\rangle
\]
\[
\text{starray-def-struct}\langle\text{st-root}\rangle\langle\text{subZ}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{keyA}\rangle\langle\text{valA}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{keyB}\rangle\langle\text{valB}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{subZ}\rangle\langle\text{keyZA}\rangle\langle\text{valZA}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{subZ}\rangle\langle\text{keyZB}\rangle\langle\text{valZB}\rangle
\]
\[
\text{starray-def-struct}\langle\text{st-root}\rangle\langle\text{subY}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{subY}\rangle\langle\text{keyYA}\rangle\langle\text{valYA}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{subY}\rangle\langle\text{keyYB}\rangle\langle\text{valYB}\rangle
\]
\[
\text{starray-def-struct}\langle\text{st-root}\rangle\langle\text{subY}\rangle\langle\text{subYYY}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{subY}\rangle\langle\text{subYYY}\rangle\langle\text{keyYYYA}\rangle\langle\text{valYYYA}\rangle
\]
\[
\text{starray-def-prop}\langle\text{st-root}\rangle\langle\text{subY}\rangle\langle\text{subYYY}\rangle\langle\text{keyYYYB}\rangle\langle\text{valYYYB}\rangle
\]

One has a single \(\langle\text{st-root}\rangle\) iterator (pointing to one of the 3 \(\langle\text{st-root}\rangle\) terms), then 3 ”\(\langle\text{subZ}\rangle\) iterators”, in fact, one \(\langle\text{subZ}\rangle\) iterator for each \(\langle\text{st-root}\rangle\) term. Likewise there are 3 ”\(\langle\text{subY}\rangle\) iterators” and 4 (four) ”\(\langle\text{subYYY}\rangle\) iterators” one for each instance of \(\langle\text{subY}\rangle\).

Every time a new term is created/instantiated, the corresponding iterator will points to it, which allows the notation used in this last example, keep in mind that one could instead, using explicit indexes:
Finally, observe that, when creating a new term, one has the option to assign a "hash" to it, in which case that term can be referred to using an iterator, the explicit index or the hash:

```
\starray_new:n \{st-root\}
\starray_def_struct:nn \{st-root\}\{subZ\}
\starray_def_struct:nn \{st-root\}\{subY\}
\starray_def_struct:nn \{st-root\}.\{subY\}\{subYYY\}
```

Will create 3 ⟨subZ⟩ terms associated with the first (index = 1) ⟨st-root⟩.

### 5.2 iterators

```
\starray_set_iter:nn \{\langle \text{starray-ref} \rangle\}\{\langle \text{int-val} \rangle\}
\starray_set_iter:nTF \{\langle \text{starray-ref} \rangle\}\{\langle \text{int-val} \rangle\}\{\langle \text{if-true} \rangle\}\{\langle \text{if-false} \rangle\}
\starray_reset_iter:nn \{\langle \text{starray-ref} \rangle\}
\starray_reset_iter:nTF \{\langle \text{starray-ref} \rangle\}\{\langle \text{if-true} \rangle\}\{\langle \text{if-false} \rangle\}
\starray_next_iter:nn \{\langle \text{starray-ref} \rangle\}
\starray_next_iter:nTF \{\langle \text{starray-ref} \rangle\}\{\langle \text{if-true} \rangle\}\{\langle \text{if-false} \rangle\}
```

Those functions allows to set an iterator to a given ⟨int-val⟩, reset it (i.e. assign 1 to the iterator), or increase the iterator by one. An iterator might have a value between 1 and the number of instantiated terms (if the given (sub-)structure was already instantiated). If the (sub-)structure hasn’t been instantiated yet, the iterator will always end being set to 0. The branching versions allows to catch those cases, like trying to set a value past its maximum, or a value smaller than one.

**Important:** Please observe that, when setting/resetting/incrementing the iterator of a (sub-)structure, all "descending" iterators will be also be reset.

**Note:** A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error. The branching version doesn’t raise any warning.
After the reset \langle \text{st-root}.\text{subY}.\text{subYYY}\rangle was equivalent to \langle \text{st-root}[2].\text{subY}[2].\text{subYYY}[2]\rangle, given that each iterator was pointing to the "last term", since the reset was of the \langle \text{subY} \rangle iterator, only it and the descending ones (in this example just \langle \text{subYYY} \rangle) were reseted, and therefore \langle \text{st-root}.\text{subY}.\text{subYYY}\rangle was then equivalent to \langle \text{st-root}[2].\text{subY}[1].\text{subYYY}[1]\rangle.

Note: A warning is raised (see 2) in case of a \langle \text{starray-ref} \rangle syntax error or invalid \langle \text{hash} \rangle. The branching version doesn’t raise any warning.

\textbf{Warning:} This can be used after any command which 'parses a term', for instance \texttt{\text{starray_term_syntax:n}}, see section 7, but it only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.
\array parsed get iter:NN ★ \array parsed get iter:NN {{parsed-refA}} {{parsed-refB}}

new: 2023/11/28

\array parsed get iter: will place in the current iterator’s value associated with (parsed-refA) and (parsed-refB), using \int use:N, in the input stream.

Warning: (parsed-refA) and (parsed-refB) are the values returned by \array term syntax:nNN.

\array get cnt:n \array get cnt:n {{starray-ref}} \array get cnt:nN \array get cnt:n{{integer}} \array get cnt:nNNTF \array get cnt:nNNTF {{starray-ref}}{{integer}}{{if-true}}{{if-false}}

\array get cnt:n {{starray-ref}} will type in the current number of terms of a given (sub-)structure, whilst the other two functions will save it’s value in a integer variable (expl3).

Note: A warning is raised (see 2) in case of a (starray-ref) syntax error. The branching version doesn’t raise any warning.

\array parsed get cnt: ★ \array parsed get cnt:

new: 2023/05/20

\array parsed get cnt: will place the current number of terms, using \int use:N, of the last parsed term, in the input stream.

Warning: This can be used after any command which ’parses a term’, for instance \array term syntax:n, see section 7, but it only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\array parsed get cnt:NN ★ \array parsed get cnt:NN {{parsed-refA}} {{parsed-refB}}

new: 2023/11/28

\array parsed get cnt: will place in the current number of terms associated with (parsed-refA) and (parsed-refB), using \int use:N, in the input stream.

Warning: (parsed-refA) and (parsed-refB) are the values returned by \array term syntax:nNN.

\array iterate over:nn \array iterate over:nn {{starray-ref}}{{code}} \array iterate over:nnNTF \array iterate over:nnNTF {{starray-ref}}{{code}}{{if-true}}{{if-false}}

new: 2023/11/04

\array iterate over:nn will reset the (starray-ref) iterator, and then execute (code) for each valid value of iter. At the loop’s end, the (starray-ref) iterator will point to the last element of it. The (if-true) is executed, at the loop’s end if there is no syntax error, and the referenced structure was properly instantiated. Similarly (if-false) is only execute if a syntax error is detected or the referenced structure wasn’t properly instantiated

Note: \array iterate over:nn Creates a local group, so that one can recurse over sub-structures. Be aware, then, that (code) is executed in said local group.

Note: A warning is raised (see 2) in case of a (starray-ref) syntax error or the structure wasn’t yet instantiated. The branching version doesn’t raise any warning.
6 Changing and recovering starray properties

Those are the functions that allow to (g)set (change) the value of a term’s property. If the ⟨prop-key⟩ isn’t already present it will be added just for that term ⟨starray-ref⟩. The ⟨nnV⟩ variants allow to save any variable like a token list, property list, etc...

Note: A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error. The branching version doesn’t raise any warning.

It is possible to set a collection of properties using a key/val syntax, similar to the one used to define a starray from keyvals (see 4), with a few distinctions:

1. when referring a (sub-)structure one can either explicitly use an index, or
2. implicitly use it’s iterator
3. if a given key isn’t already presented it will be added only to the given term

Note that, in the following example, TWO iterators are being used, the one for ⟨st-root⟩ and then ⟨subY⟩.

Also note that the above example is fully equivalent to:

```
\starray_set_prop:nnn {st-root} {keyA} {valA}
\starray_set_prop:nnn {st-root} {keyB} {valB}
\starray_set_prop:nnn {st-root.subZ[2]} {keyZA} {valZA}
\starray_set_prop:nnn {st-root.subZ[2]} {keyZB} {valZB}
\starray_set_prop:nnn {st-root.subY} {keyYA} {valYA}
\starray_set_prop:nnn {st-root.subY} {keyYB} {valYB}
\starray_set_prop:nnn {st-root.subY.subYYY[1]} {keyYYYa} {valYYYa}
\starray_set_prop:nnn {st-root.subY.subYYY[1]} {keyYYYb} {valYYYb}

\starray_get_prop:nn {⟨starray-ref⟩} {⟨key⟩}

\starray_get_prop:nnN \starray_get_prop:nnN {⟨starray-ref⟩} {⟨key⟩} {⟨tl-var⟩}
\starray_get_prop:nnN \starray_get_prop:nN {⟨starray-ref⟩} {⟨key⟩} {⟨if-true⟩} {⟨if-false⟩}

\starray_get_prop:nn {⟨starray-ref⟩} {⟨key⟩} places the value of ⟨key⟩ in the input stream.
\starray_get_prop:nnN {⟨starray-ref⟩} {⟨key⟩} {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}

\starray_get_prop:nn {⟨starray-ref⟩} {⟨key⟩} places the value of ⟨key⟩ in the input stream.
\starray_get_prop:nnN {⟨starray-ref⟩} {⟨key⟩} {⟨tl-var⟩} recovers the value of ⟨key⟩ and places it in ⟨tl-var⟩ (a token list variable), this is specially useful in conjunction with \starray_set_prop:nnV, whilst the \starray_get_prop:nnN version branches accordly.

Note: In case of a syntax error, or ⟨key⟩ doesn’t exist, an empty value is left in the stream (or ⟨tl-var⟩).

Note: A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error. The branching version doesn’t raise any warning.

\starray_parsed_get_prop:n ★ \starray_parsed_get_prop:n {⟨key⟩}
\starray_parsed_get_prop:n {⟨key⟩} places the value of ⟨key⟩, if it exists, from the last parsed term, in the input stream.

Warning: This can be used after any command which ‘parses a term’, for instance \starray_term_syntax:n, see section 7, but it only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\starray_parsed_get_prop:NNn ★ \starray_parsed_get_prop:NNn {⟨parsed-refA⟩} {⟨parsed-refB⟩} {⟨key⟩}
\starray_parsed_get_prop:NNn places the value of ⟨key⟩, if it exists, associated with ⟨parsed-refA⟩ and ⟨parsed-refB⟩.

Warning: ⟨parsed-refA⟩ and ⟨parsed-refB⟩ should be the values returned by \starray_term_syntax:n.

7 Additional Commands and Conditionals

\starray_if_in:nnF \starray_if_in:nnF {⟨starray-ref⟩} {⟨key⟩} {⟨if-true⟩} {⟨if-false⟩}
The \starray_if_in:nnF {⟨starray-ref⟩} {⟨key⟩} {⟨if-true⟩} {⟨if-false⟩} tests if a given ⟨key⟩ is present.

\starray_term_syntax:n \starray_term_syntax:n {⟨starray-ref⟩}
\starray_term_syntax:n {⟨starray-ref⟩} {⟨if-true⟩} {⟨if-false⟩}
new: 2023/05/20

This will just parse a ⟨starray-ref⟩ reference, and set interval variables so that commands like \starray_parsed__ can be used.

Warning: The main idea is to allow some expandable commands, but be aware that all \starray commands that use a ⟨starray-ref⟩ use the very same parser variables. In case one needs are more permanent/resilient starray reference, one should use the \starray_term_syntax:n variant.

Note: A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error. The branching version doesn’t raise any warning.
Similar to the ones above (\texttt{\textbackslash{starray}_{syntax}:nn}). \texttt{(parsed-refA)} and \texttt{(parsed-refB)} (assumed to be two token list vars, (\texttt{tl-var})) will receive two 'internal references' that can be used in commands like \texttt{\textbackslash{starray}_{parsed}....:nn} which expects such 'references', without having to worry about using another \texttt{\textbackslash{starray}..} command.

\textbf{Note}: Once correctly parsed, \texttt{(parsed-refA)} and \texttt{(parsed-refB)} can be used at 'any time' (by those few \texttt{\textbackslash{starray}_{parsed}....:nn} associated commands).

\textbf{Note}: A warning is raised (see 2) in case of a \texttt{\textbackslash{starray}_{ref}} syntax error (in which case \texttt{(parsed-refA)} and \texttt{(parsed-refB)} will not hold a valid value). The branching version doesn’t raise any warning.

\begin{verbatim}
\texttt{\textbackslash{starray}_{parsed if in p}:nn} \texttt{\textbackslash{starray}_{parsed if in}:nn} \texttt{(key) \texttt{(if-true)} \texttt{(if-false)}}
\end{verbatim}

This will test if the given \texttt{key} is present in the "last parsed term".

\textbf{Note}: The predicate version, \texttt{.p}, expands to either \texttt{\textbackslash{prg}_{return true}}: or \texttt{\textbackslash{prg}_{return false}}:.

\textbf{Warning}: This can be used after any command which 'parses a term', for instance \texttt{\textbackslash{starray}_{syntax}:nn}, but it only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\begin{verbatim}
\texttt{\textbackslash{starray}_{parsed if in p}:nNn} \texttt{\textbackslash{starray}_{parsed if in}:nNn} \texttt{(parsed-refA)} \texttt{(parsed-refB)} \texttt{(key) \texttt{(if-true)} \texttt{(if-false)}}
\end{verbatim}

This will test if the given \texttt{key} is present/associated with \texttt{(parsed-refA)} and \texttt{(parsed-refB)}.

\textbf{Note}: The predicate version, \texttt{.p}, expands to either \texttt{\textbackslash{prg}_{return true}}: or \texttt{\textbackslash{prg}_{return false}}:.

\textbf{Warning}: \texttt{(parsed-refA)} and \texttt{(parsed-refB)} should be the values returned by \texttt{\textbackslash{starray}_{syntax}:nn}.

\begin{verbatim}
\texttt{\textbackslash{starray}_{get unique id}:nn} \texttt{\textbackslash{starray}_{get unique id}:nN} \texttt{(tl-var)}
\end{verbatim}

\textbf{Warning}: A warning is raised (see 2) in case of a \texttt{\textbackslash{starray}_{ref}} syntax error. The branching version doesn’t raise any warning.

\section{Showing (debugging) starrays}

\begin{verbatim}
\texttt{\textbackslash{starray}_{show def}:n} \texttt{\textbackslash{starray}_{show def in text}:n}
\end{verbatim}
Displays the ⟨starray⟩ structure definition and initial property values in the terminal or directly in text.

\arrayshowterms:n  \arrayshowterms{n}\{\\array-ref\}
\arrayshowterms:n{\array-ref} \arrayshowtermsinlinetext:n{\array-ref}

Displays the ⟨starray⟩ instantiated terms and current property values in the terminal or directly in text.