The starray Package
Version 2.1

Alceu Frigeri*

February 2026

Abstract

This package implements vector like 'structures', alike 'C' and other programming languages. It's based on *expl3*[3] and aimed at 'package writers', and not end users. The provided commands 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

The main idea is to have an array like syntax when setting/recovering structured information, e.g. `\starray_get_prop:nn {⟨student[2].work[3].reviewer[4]⟩}{⟨name⟩}` 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, ⟨name⟩ being a property of "reviewer". Moreover one can iterate over the structure, for instance `\starray_get_prop:nn {⟨student.work.reviewer⟩}{⟨name⟩}` is also a possible reference in which one is using "student's", "work's" and "reviewer's" iterators.

*https://github.com/alceu-frigeri/starray

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*[3] 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 a time) or with a keyval interface or both, either way, one has to first "create a root starray" (`\starray_new:n`), 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".

Finally, almost all defined functions have a branching version, as per *expl3*[3]: `T`, `F` and *TF* (note: no `_p` variants, see below). For simplicity, in the text bellow only the *TF* variant is described, as in `\starray_new:nTF`, keep in mind that all 3 variants are defined, e.g. `\starray_new:nT`, `\starray_new:nF` and `\starray_new: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 an inherent structure complexity, number of used/defined auxiliary property lists and access time.
>
> ***Important:*** *Expandability*, unfortunately most/all defined functions are not expandable, in particular, most conditional/branching functions aren't, with just a few exceptions (marked with a star ★).

1.1 Prerequisites

Besides a fairly recent kernel (as recent as 2025/06/01, see [4]), the packages *pkginfograb*[1] (for package documentation) and *tokglobalstack*[2] (stack implementation for the iterate over commands see 6.2.1) are also used.

2 Package Options

The package options (*key*=*value*) are:

`msg-err` By default, the *starray* 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 `\starray_new:n` cases (*starray* creation);
2. *syntax* relates to "term syntax" errors (student.work.reviewer in the above examples); and
3. *reference* relates to cases whereas the syntax is correct but referring to a non-existent term or property.

`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.

`parsed check` By default (false) the many `\starray_parsed_` commands won't check if the last `\starray_term_parser:` was successful. With this option, they will test it (with a performance hit) raising a warning/error accordantly.

`NN names` Compatibility option. See 9.2 for more details. Possible values are:

`none`	(default) None of the old `...parsed..:NN..` functions will be defined.
`no warnings`	The old `...parsed..:NN..` functions will be defined. No warnings will be issued

`strict` The old `...parsed..:NN..` functions will be defined, issuing a deprecation warning at each use.

`iter cascade` By default, since version 2.0, when setting the value of an iterator, none of the substructure's iterators will be affected. With this option set, all substructure's iterators will be reset to 1 (or zero), see 6.2.

3 Demo package(s)

Given the inherent complexity of this package, one can find at `https://github.com/alceu-frigeri/starray/tree/main/demo` an example, *stdemo.sty*, package and documentation *stdemo.pdf*. Since the aforementioned package, and documentation, are just an example of use, it doesn't make sense to add them to CTAN.

4 Creating a starray

`\starray_new:n` `\starray_new:n {⟨starray⟩}`
`\starray_new:nTF` `\starray_new:nTF {⟨starray⟩} {⟨if-true⟩} {⟨if-false⟩}`

Creates a new ⟨starray⟩ 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 ⟨starray-root⟩ or just ⟨root⟩.

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

4.1 Conditionals

`\starray_if_exist_p:n` ⋆ `\starray_if_exist_p:n {⟨starray⟩}`
`\starray_if_exist:nTF` ⋆ `\starray_if_exist:nTF {⟨starray⟩} {⟨if-true⟩} {⟨if-false⟩}`
`\starray_if_valid_p:n` ⋆ `\starray_if_valid_p:n {⟨starray⟩}`
`\starray_if_valid:nTF` ⋆ `\starray_if_valid:nTF {⟨starray⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2023/05/20
updated: 2024/03/28

`\starray_if_exist:nTF` only tests if ⟨starray⟩ (the base array) is defined. It doesn't verifies if it really is a *starray*. `\starray_if_valid:nTF` is functionally equivalent, since release 1.9. See `\starray_term_parser:nTF`, section 8, for a more reliable validity test.

Note: The predicate versions, `_p`, expand to either `\c_true_bool` or `\c_false_bool`

5 Defining and initialising a starray structure

`\starray_def_prop:nnn` `\starray_def_prop:nnn {⟨starray-ref⟩} {⟨prop-key⟩} {⟨initial-value⟩}`
`\starray_def_prop:nnnTF` `\starray_def_prop:nnnTF {⟨starray-ref⟩} {⟨prop-key⟩} {⟨initial-value⟩} {⟨if-true⟩} {⟨if-false⟩}`

Adds an entry, ⟨prop-key⟩, to the ⟨starray-ref⟩ (see 6.1) definition and set its initial value. If ⟨prop-key⟩ is already present its initial value is updated. Both ⟨prop-key⟩ and ⟨initial-value⟩ may contain any ⟨balanced text⟩. ⟨prop-key⟩ is an (*expl3*[3]) property list ⟨key⟩ meaning that category codes are ignored.
The definition/assignment of a ⟨prop-key⟩ to a ⟨starray-ref⟩ is global.

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

\starray_def_structure:nn \starray_def_struct:nn {⟨starray-ref⟩} {⟨struct-name⟩}
\starray_def_structure:nnTF \starray_def_struct:nnTF {⟨starray-ref⟩} {⟨struct-name⟩} {⟨if-true⟩} {⟨if-false⟩}

Adds a sub-structure (a *starray* in itself) to ⟨starray-ref⟩ (see 6.1). If ⟨struct-name⟩ is already present nothing happens. The definition/assignment of a ⟨struct-name⟩ to a ⟨starray-ref⟩ 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 6.1).

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

\starray_def_from_keyval:nn \starray_def_from_keyval:nn {⟨starray-ref⟩} {⟨keyval-lst⟩}
\starray_def_from_keyval:nnTF \starray_def_from_keyval:nnTF {⟨starray-ref⟩} {⟨keyval-lst⟩} {⟨if-true⟩} {⟨if-false⟩}

Adds a set of ⟨keys⟩ / ⟨values⟩ and/or ⟨structures⟩ to ⟨starray-ref⟩ (see 6.1). The ⟨keyval-lst⟩ is pretty straightforward, the construction ⟨*key*⟩ . *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.

5.1 Fixing an ill-instantiated starray

When instantiating (see 6) a *starray*, the associated structure will be constructed based on it's "current definition" (see 5). A problem that might arise, when one extends the definition of an already instantiated *starray* (better said, if one adds a sub-structure to it), is that a *quark loop* will issue (from *l3quark*). To avoid that *quark loop* it is necessary to "fix" the structure of the 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, the reason this isn't automatically executed when adding a sub-structure, is that this is an expensive operation, because it has to craw over all the terms of an instantiated *starray* adding any missing sub-structure reference, but one doesn't need to run it "right away" it is possible to add a bunch of sub-structures and then run this just once.

6 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-)struture 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

```
\starray_new:n {st-root}

\starray_def_from_keyval:nn {st-root}
  {
    keyA = valA ,
    keyB = valB ,
    subZ . struct =
      {
        keyZA = valZA ,
        keyZB = valZB ,
      }
    subY . struct =
      {
        keyYA = valYA ,
        keyYB = valYB ,
        subYYY . struct =
          {
            keyYYYa = valYYYa ,
            keyYYYb = valYYYb
          }
      }
  }

\starray_new_term:n {st-root}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subY}
\starray_new_term:nn {st-root}{hash-A}
\starray_new_term:n {st-root.subZ}
```

One will have created 6 *terms*:

1. 2 ⟨st-root⟩ *terms*
   (a) the first one with index 1 and
      i. 2 sub-structures ⟨subZ⟩ (indexes 1 and 2)
      ii. 1 sub-structure ⟨subY⟩ (index 1)
   (b) the second one with indexes 2 and "hash-A" and
      i. 1 sub-structure ⟨subZ⟩ (index 1)

Note that, in the above example, it was used the "implicit" indexing (aka. iterator, see 6.1). Also note that no *term* ⟨subYYY⟩ was created.

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

6.1 Referencing Terms

When typing a ⟨starray-ref⟩ 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 ⟨starray-root⟩ will use a construct like ⟨starray-root⟩.⟨sub-struct⟩.⟨sub-struct⟩... For example, an equivalent construct to the one shown in 6 :

```
\starray_new:n {st-root}

\starray_def_struct:nn {st-root}{subZ}

\starray_def_prop:nnn {st-root}{keyA}{valA}
\starray_def_prop:nnn {st-root}{keyB}{valB}

\starray_def_prop:nnn {st-root.subZ}{keyZA}{valZA}
\starray_def_prop:nnn {st-root.subZ}{keyZB}{valZB}

\starray_def_struct:nn {st-root}{subY}
\starray_def_prop:nnn {st-root.subY}{keyYA}{valYA}
\starray_def_prop:nnn {st-root.subY}{keyYB}{valYB}

\starray_def_struct:nn {st-root.subY}{subYYY}
\starray_def_prop:nnn {st-root.subY.subYYY}{keyYYYA}{valYYYA}
\starray_def_prop:nnn {st-root.subY.subYYY}{keyYYYB}{valYYYB}
```

Note that, all it's needed in order to be able to use ⟨starray-root⟩.⟨sub-A⟩ is that ⟨sub-A⟩ is an already declared sub-structure of ⟨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 ⟨starray-ref⟩, is:

⟨starray-root⟩⟨idx⟩.⟨sub-A⟩⟨idxA⟩.⟨sub-B⟩⟨idxB⟩

In the case of term instantiation the last ⟨sub-⟩ cannot be indexed, after all one is creating a new term/index. Moreover, all ⟨idx⟩ are optional like:

⟨starray-root⟩.⟨sub-A⟩⟨idxA⟩.⟨sub-B⟩

in which case, one is using the "iterator" of ⟨starray-root⟩ and ⟨sub-B⟩ (more later, but keep in mind the ⟨sub-B⟩ iterator is the ⟨sub-B⟩ associated with the ⟨sub-A⟩⟨idxA⟩).

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

So, for example:

```
\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}

\starray_new_term:n {st-root}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subY}
\starray_new_term:n {st-root.subY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root.subY}

\starray_new_term:n {st-root}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subY}
```

One has a single ⟨st-root⟩ iterator (pointing to one of the 3 ⟨st-root⟩ terms), then 3 "⟨subZ⟩ iterators", in fact, one ⟨subZ⟩ iterator for each ⟨st-root⟩ term. Likewise there are 3 "⟨subY⟩ iterators" and 4 (four) "⟨subYYY⟩ iterators" one for each instance of ⟨subY⟩.

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:

```
\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}

\starray_new_term:n {st-root}
\starray_new_term:n {st-root[1].subZ}
\starray_new_term:n {st-root[1].subZ}
\starray_new_term:n {st-root[1].subY}
\starray_new_term:n {st-root[1].subY}
\starray_new_term:n {st-root[1].subY[2].subYYY}
\starray_new_term:n {st-root[1].subY}

\starray_new_term:n {st-root}
\starray_new_term:n {st-root[2].subZ}
\starray_new_term:n {st-root[2].subZ}
\starray_new_term:n {st-root[2].subY}
```

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}

\starray_new_term:nn {st-root}{hash-A}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root[1].subZ}
\starray_new_term:n {st-root[hash-A].subZ}
```

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

6.2 Iterators

In the following commands, since version 2.0, when setting/resetting/incrementing the iterator of a (sub-)structure, only the given (sub-)structure iterator will be affected. With the package option `iter cascade` (see 2) all "descending" iterators will also be reset to *1* or *0*. All assignments to a structure's iterator are global.

`\starray_set_iter:nn`	`\starray_set_iter:nn` {⟨starray-ref⟩} {⟨int-val⟩}
`\starray_set_iter:nnTF`	`\starray_set_iter:nTF` {⟨starray-ref⟩} {⟨int-val⟩} {⟨if-true⟩} {⟨if-false⟩}
`\starray_reset_iter:nn`	`\starray_reset_iter:n` {⟨starray-ref⟩}
`\starray_reset_iter:nTF`	`\starray_reset_iter:nTF` {⟨starray-ref⟩} {⟨if-true⟩} {⟨if-false⟩}
`\starray_next_iter:n`	`\starray_next_iter:n` {⟨starray-ref⟩}
`\starray_next_iter:nnTF`	`\starray_next_iter:nTF` {⟨starray-ref⟩} {⟨if-true⟩} {⟨if-false⟩}

updated: 2026/02/01

Those functions allows to **set** an iterator to a given value, ⟨int-val⟩, **reset** it (i.e. assign 1 to it), or increase the iterator by one. An iterator might have a value between 1 and the number of instantiated terms. If the (sub-)structure wasn't instantiated, the iterator will be 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, otherwise values outside the valid range won't raise any warning/error.

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

In the following example, given:

```
\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}
```

```
\starray_new_term:n {st-root}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root.subY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subZ}
\starray_new_term:n {st-root.subY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root.subY}
\starray_new_term:n {st-root.subY.subYYY}
\starray_new_term:n {st-root.subY.subYYY}
```

```
\starray_set_prop:nnn {st-root.subY.subYYY}{key}{val}
\starray_set_prop:nnn {st-root[2].subY[2].subYYY[2]}{key}{val}

\starray_reset_iter:n {st-root[2].subY}

\starray_set_prop:nnn {st-root.subY.subYYY}{key}{val}
\starray_set_prop:nnn {st-root[2].subY[1].subYYY[1]}{key}{val}
```

Before the reset ⟨st-root.subY.subYYY⟩ was equivalent to ⟨st-root[2].subY[2].subYYY[2]⟩, given that each iterator was pointing to the "last term", since the reset was of the ⟨subY⟩ iterator, only it reset, and therefore ⟨st-root.subY.subYYY⟩ was then equivalent to ⟨st-root[2].subY[1].subYYY[2]⟩.

Note: With the package option `iter cascade` all sub-structure's iterators will also be set/reset, meaning, in the above example, that after resetting the ⟨subY⟩, the iterator of ⟨subYYY⟩ (it's only descendant) will also be reset, therefore ⟨st-root.subY.subYYY⟩ would be equivalent to ⟨st-root[2].subY[1].subYYY[1]⟩ in this case.

`\starray_set_iter_from_hash:nn` `\starray_set_iter_from_hash:nn {⟨starray-ref⟩}{⟨hash⟩}`
`\starray_set_iter_from_hash:nnTF` `\starray_set_iter_from_hash:nnTF {⟨starray-ref⟩}{⟨hash⟩}{⟨if-true⟩}{⟨if-false⟩}`

new: 2023/11/04

`\starray_set_iter_from_hash:nn {⟨starray-ref⟩}{⟨hash⟩}` will set iter based on the ⟨hash⟩ used when instantiating a term (see 6).

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

`\starray_get_iter:n` `\starray_get_iter:n {⟨starray-ref⟩}`
`\starray_get_iter:nN` `\starray_get_iter:nN {⟨starray-ref⟩}{⟨int-var⟩}`
`\starray_get_iter:nNTF` `\starray_get_iter:nNTF {⟨starray-ref⟩}{⟨int-var⟩}{⟨if-true⟩}{⟨if-false⟩}`

`\starray_get_iter:n {⟨starray-ref⟩}` will type in the current value of a given iterator, whilst the other two functions will save it's value in a integer variable (*expl3*[3]). The assignment is local.

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

`\starray_get_cnt:n` `\starray_get_cnt:n {⟨starray-ref⟩}`
`\starray_get_cnt:nN` `\starray_get_cnt:nN {⟨starray-ref⟩}{⟨integer⟩}`
`\starray_get_cnt:nNTF` `\starray_get_cnt:nNTF {⟨starray-ref⟩}{⟨integer⟩}{⟨if-true⟩}{⟨if-false⟩}`

`\starray_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*[3]). The assignment is local.

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

6.2.1 Iterating Over

\starray_iterate_over:nn \starray_iterate_over:nn {⟨starray-ref⟩} {⟨code⟩}
\starray_iterate_over:nnTF \starray_iterate_over:nnTF {⟨starray-ref⟩} {⟨code⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2023/11/04
updated: 2026/02/01

\starray_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. See 9.3 for a series of helper/companion commands when writing ⟨code⟩.

Note: \starray_iterate_over:nn is recurse aware, and can be nested to recurse over sub-structures. Be aware, though, that all iterators assignments are global.

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.

7 Changing and Recovering starray Properties

\starray_set_prop:nnn \starray_set_prop:nnn {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩}
\starray_set_prop:nnV \starray_set_prop:nnV {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩}
\starray_set_prop:nnnTF \starray_set_prop:nnnTF {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩} {⟨if-true⟩} {⟨if-false⟩}
\starray_set_prop:nnVTF \starray_set_prop:nnVTF {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩} {⟨if-true⟩} {⟨if-false⟩}
\starray_gset_prop:nnn \starray_gset_prop:nnn {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩}
\starray_gset_prop:nnV \starray_gset_prop:nnV {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩}
\starray_gset_prop:nnnTF \starray_gset_prop:nnnTF {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩} {⟨if-true⟩} {⟨if-false⟩}
\starray_gset_prop:nnVTF \starray_gset_prop:nnVTF {⟨starray-ref⟩} {⟨prop-key⟩} {⟨value⟩} {⟨if-true⟩} {⟨if-false⟩}

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 to that term, ⟨starray-ref⟩, only. The ⟨nnV⟩ variants allow to save the value of a variable like a token list, clist 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.

\starray_set_from_keyval:nn \starray_set_from_keyval:nnn {⟨starray-ref⟩} {⟨keyval-lst⟩}
\starray_set_from_keyval:nnTF \starray_set_from_keyval:nnnTF {⟨starray-ref⟩} {⟨keyval-lst⟩} {⟨if-true⟩} {⟨if-false⟩}
\starray_gset_from_keyval:nn \starray_gset_from_keyval:nnn {⟨starray-ref⟩} {⟨keyval-lst⟩}
\starray_gset_from_keyval:nnTF \starray_gset_from_keyval:nnnTF {⟨starray-ref⟩} {⟨keyval-lst⟩} {⟨if-true⟩} {⟨if-false⟩}

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 5), 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⟩.

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

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_get_prop:nn` {⟨starray-ref⟩} {⟨key⟩}
`\starray_get_prop:nnN` `\starray_get_prop:nnN` {⟨starray-ref⟩} {⟨key⟩} {⟨tl-var⟩}
`\starray_get_prop:nnNTF` `\starray_get_prop:nnNTF` {⟨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:nnNTF` version branches accordly. The assignment is local.

> ***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.

8 Additional Commands and Conditionals

`\starray_if_in:nnTF` `\starray_if_in:nnTF` {⟨starray-ref⟩} {⟨key⟩} {⟨if-true⟩} {⟨if-false⟩}

The `\starray_if_in:nnTF` {⟨starray-ref⟩} {⟨key⟩} {⟨...⟩} {⟨...⟩} tests if a given ⟨key⟩ is present.

`\starray_get_unique_id:nN` `\starray_get_unique_id:nN` {⟨starray-ref⟩} {⟨tl-var⟩}
`\starray_get_unique_id:nNTF` `\starray_get_unique_id:nNTF` {⟨starray-ref⟩} {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2024/03/10

Gets an 'unique ID' for a given ⟨starray-ref⟩ *term*, it should help defining/creating uniquely identified auxiliary structures, like auxiliary property or sequence lists, since one can't (better said shouldn't, as per l3kernel) store an anonymous property/sequence list using V-expansion. The assignment is local.

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

9 Parsed Commands

Since the parsing of a ⟨starray-ref⟩ is a non-expandable and expensive operation, the commands below allow for some coding speed up (by avoiding parsing the same ⟨starray-ref⟩ repeatedly) and offers expandable alternatives for a few commands.

The use pattern would be (1) to first parse the ⟨starray-ref⟩ with either `\starray_term_parser:n` or `\starray_term_parser:nN` and thereafter (2) use the many `\starray_parsed_` or `\starray_uparsed_` commands.

Note that, there are three sets of commands, one associated with `\starray_term_parser:n` and/or `\starray_term_parser:nTF` (which relies on internal variables), another set associated with `\starray_term_parser:nN` and/or `\starray_term_parser:nNTF` (which allows to save more than one ⟨starray-ref⟩ parsed term), and lastly one associated with `\starray_iterate_over:nn` (see 6.2.1).

9.1 Parsed Commands Based on Internal Variables

`\starray_term_parser:n` `\starray_term_parser:n {⟨starray-ref⟩}`
`\starray_term_parser:nTF` `\starray_term_parser:nTF {⟨starray-ref⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2023/05/20
updated: 2025/10/25

In case one needs to access the same term again and again, this will just parse a ⟨starray-ref⟩ reference once, and set interval variables so that commands like `\starray_parsed_` can be used thereafter (avoiding having to slowly parse the same term over and over).

Note: The internal variables used are exclusive, no other command (besides these two), set them. This allows to "parse a term" and call other `\starray_` commands before using the "parsed term" with one of the `\starray_parsed_` commands.

Warning: While it allows for some code speedup, and enables some commands to be fully expandable, be aware that the internal variables will only be set correctly if, and only if, the ⟨starray-ref⟩ is a valid term reference.

Note: By default, the many associated `\starray_parsed_` won't check the status of the last `\starray_term_parser:n` operation. This can be changed with the package option `parsed check` (see 2) in which case all associated `\starray_parsed_` will then verify the status of the last operation and raise a warning/error.

Note: A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error, in which case the internal variables won't be set correctly. The branching version doesn't raise any warning.

Note: The `\starray_term_syntax:n` and `\starray_term_syntax:nTF` have been deprecated (version 1.11), a warning is raised if a deprecated one is called.

`\starray_parsed_if_in_p:n` ⋆ `\starray_parsed_if_in_p:nTF {⟨key⟩}`
`\starray_parsed_if_in:nTF` ⋆ `\starray_parsed_if_in:nTF {⟨key⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2023/05/20

This will test if the given *key* is present in the "last parsed term".

Note: The predicate version, `_p`, expands to either `\c_true_bool` or `\c_false_bool`.

Warning: This can only be used after `\starray_term_parser:n` and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

`\starray_parsed_get_iter:` ⋆ `\starray_parsed_get_iter:`

new: 2023/05/20

`\starray_parsed_get_iter:` will place in the current iterator's value, using `\int_use:N`, of the last parsed term in the input stream.

Warning: This can only be used after `\starray_term_parser:n` and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\starray_parsed_get_iter:N \starray_parsed_get_iter:N {⟨int-var⟩}
\starray_parsed_get_iter:NTF \starray_parsed_get_iter:NTF {⟨int-var⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2025/10/25

These will save the iterator's value (of a parsed term) in a integer variable (*expl3*[3]). The ⟨if-true⟩ and ⟨if-false⟩ regards the status of the last \starray_term_parser: command, iff the option parsed check (see 2) is enable, otherwise it will always execute the ⟨if-true⟩ branch.

Warning: This can only be used after \starray_term_parser:n and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\starray_parsed_get_cnt: ⋆ \starray_parsed_get_cnt:

new: 2023/05/20

\starray_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 only be used after \starray_term_parser:n and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\starray_parsed_get_cnt:N \starray_get_cnt:N {⟨integer⟩}
\starray_parsed_get_cnt:NTF \starray_get_cnt:NTF {⟨integer⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2025/10/25

Similarly to \starray_get_cnt:nN and \starray_get_cnt:nNTF these will save the number of terms (of the last parsed term) in a integer variable (*expl3*[3]). The ⟨if-true⟩ and ⟨if-false⟩ regards the status of the last \starray_term_parser: command, iff the option parsed check (see 2) is enable, otherwise it will always execute the ⟨if-true⟩ branch.

Warning: This can only be used after \starray_term_parser:n and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\starray_parsed_get_prop:n ⋆ \starray_parsed_get_prop:n {⟨key⟩}

new: 2023/05/20

\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 only be used after \starray_term_parser:n and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\starray_parsed_get_prop:nN \starray_parsed_get_prop:nN {⟨key⟩} {⟨tl-var⟩}
\starray_parsed_get_prop:nNTF \starray_parsed_get_prop:nNTF {⟨key⟩} {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2025/10/25

\starray_parsed_get_prop:nN {⟨key⟩} {⟨tl-val⟩} stores the value of ⟨key⟩, if it exists, from the last parsed term. The ⟨if-false⟩ branch is executed if ⟨key⟩ doesn't exist or (if the option parsed check, see 2, is enabled) if the last parser operation has failed.

Warning: This can only be used after \starray_term_parser:n and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

\starray_parsed_get_unique_id:nN \starray_parsed_get_unique_id:nN {⟨tl-var⟩}
\starray_parsed_get_unique_id:nNTF \starray_parsed_get_unique_id:nNTF {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2025/10/25

Gets an 'unique ID' from the last parsed term. The ⟨if-true⟩ and ⟨if-false⟩ regards the status of the last \starray_term_parser: command, iff the option parsed check (see 2) is enable, otherwise it will always execute the ⟨if-true⟩ branch.

Warning: This can only be used after \starray_term_parser:n and only makes sense (and returns a reliable/meaningful result) IF the last parser operation was successfully executed.

9.2 Parsed Commands Based on User Variables

Due to internal changes, since version 1.12, there is no longer the need to use two variables to save an 'internal reference', though just reducing the signatures of the original `_parsed_` commands by one N resulted in some name crashing which, unfortunately, obliged a series of commands renaming, as follows: all commands related in this section now are named `\starray_uparsed_` so they are completely distinct from those commands from the previous section, 9.1. By default, only the new names are defined. Using the package option `NN names`, see 2, the old command names get also defined. With `NN names = strict` the old commands will issue an warning pointing to the new names (this might result in low level errors if the commands are used in an expansion context). With `NN names = no warnings` the old commands won't issue any warning about their use.

`\starray_term_parser:nN` `\starray_term_parser:nN {⟨starray-ref⟩} {⟨parsed-refA⟩}`
`\starray_term_parser:nNTF` `\starray_term_parser:nNTF {⟨starray-ref⟩} {⟨parsed-refA⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2023/11/28
updated: 2025/10/25
updated: 2025/12/08

⟨parsed-refA⟩ (assumed to be a token list variable, ⟨tl-var⟩) will receive an 'internal reference' that can be used in commands like `\starray_uparsed_...:N` which expects such 'reference'. The assignment is global.

Note: Once correctly parsed, ⟨parsed-refA⟩ can be used at 'any time' (by those few `\starray_uparsed_...:N` associated commands).

Note: A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error (in which case ⟨parsed-refA⟩ will not hold a valid value). The branching version doesn't raise any warning.

Note: Commands `\starray_term_parser:nNN` and `\starray_term_parser:nNNTF`, besides `\starray_term_syntax:nNN` and `\starray_term_syntax:nNNTF`, have been deprecated (versions 1.11 and 1.12), a warning is raised if a deprecated one is called.

`\starray_uparsed_if_in_p:Nn` ⋆ `\starray_uparsed_if_in_p:nTF {⟨parsed-refA⟩} {⟨key⟩}`
`\starray_uparsed_if_in:NnTF` ⋆ `\starray_uparsed_if_in:nTF {⟨parsed-refA⟩} {⟨key⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2023/11/28
updated: 2025/12/08

This will test if the given *key* is present/associated with ⟨parsed-refA⟩.

Note: The predicate version, `_p`, expands to either `\c_true_bool` or `\c_false_bool`.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

`\starray_uparsed_get_iter:N` ⋆ `\starray_uparsed_get_iter:N {⟨parsed-refA⟩}`

new: 2023/11/28
updated: 2025/12/08

`\starray_uparsed_get_iter:` will place in the current iterator's value associated with ⟨parsed-refA⟩, using `\int_use:N`, in the input stream.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

`\starray_uparsed_get_iter:NN` `\starray_uparsed_get_iter:NN {⟨parsed-refA⟩} {⟨int-var⟩}`
`\starray_uparsed_get_iter:NNTF` `\starray_uparsed_get_iter:NNTF {⟨parsed-refA⟩} {⟨int-var⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2025/10/25
updated: 2025/12/08

These will save the iterator's value in a ⟨int-var⟩. The `\starray_uparsed_get_iter:NNTF` is for symmetry only (with other commands), it will always execute the ⟨if-true⟩.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

`\starray_uparsed_get_cnt:N` ⋆ `\starray_uparsed_get_cnt:N {⟨parsed-refA⟩}`

new: 2023/11/28
updated: 2025/12/08

`\starray_uparsed_get_cnt:` will place in the current number of terms associated with ⟨parsed-refA⟩, using `\int_use:N`, in the input stream.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

`\starray_uparsed_get_cnt:NN` `\starray_get_cnt:NN {⟨parsed-refA⟩} {⟨int-var⟩}`
`\starray_uparsed_get_cnt:NNTF` `\starray_get_cnt:NNTF {⟨parsed-refA⟩} {⟨int-var⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2025/10/25
updated: 2025/12/08

Similarly to `\starray_get_cnt:nN` and `\starray_get_cnt:nNTF` these will save the number of terms in ⟨int-var⟩. The `\starray_uparsed_get_cnt:NNTF` is for symmetry only (with other commands), it will always execute the ⟨if-true⟩.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

`\starray_uparsed_get_prop:Nn` ⋆ `\starray_uparsed_get_prop:Nn {⟨parsed-refA⟩} {⟨key⟩}`

new: 2023/11/28
updated: 2025/12/08

`\starray_uparsed_get_prop:Nn` places the value of ⟨key⟩, if it exists, associated with ⟨parsed-refA⟩.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

`\starray_uparsed_get_prop:NnN` `\starray_uparsed_get_prop:NnN {⟨parsed-refA⟩} {⟨key⟩} {⟨tl-var⟩}`
`\starray_uparsed_get_prop:NnNTF` `\starray_uparsed_get_prop:NnNTF {⟨parsed-refA⟩} {⟨key⟩} {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2025/10/25
updated: 2025/12/08

`\starray_uparsed_get_prop:NnN {⟨key⟩} {⟨tl-val⟩}` stores the value of ⟨key⟩, if it exists, from the parsed term.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

`\starray_uparsed_get_unique_id:NN` `\starray_uparsed_get_unique_id:NN {⟨parsed-refA⟩} {⟨tl-var⟩}`
`\starray_uparsed_get_unique_id:NNTF` `\starray_uparsed_get_unique_id:NNTF {⟨parsed-refA⟩} {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}`

new: 2025/10/25
updated: 2025/12/08

Gets an 'unique ID' from the last parsed term. The `\starray_uparsed_get_unique_id:NNTF` is for symmetry only (with other commands), it will always execute the ⟨if-true⟩.

Warning: ⟨parsed-refA⟩ should be the value returned by `\starray_term_parser:nN`.

9.2.1 Deprecation Equivalence List

Bellow follows a complete list of all deprecated names and their corresponding new names. Note each command signature (one N shorter).

Deprecated	New
`\starray_term_parser:nNN`	`\starray_term_parser:nN`
`\starray_term_parser:nNNTF`	`\starray_term_parser:nNTF`
`\starray_parsed_if_in_p:NNn`	`\starray_uparsed_if_in_p:Nn`
`\starray_parsed_if_in:NNnTF`	`\starray_uparsed_if_in:NnTF`
`\starray_parsed_get_iter:NN`	`\starray_uparsed_get_iter:N`
`\starray_parsed_get_iter:NNN`	`\starray_uparsed_get_iter:NN`
`\starray_parsed_get_iter:NNNTF`	`\starray_uparsed_get_iter:NNTF`
`\starray_parsed_get_cnt:NN`	`\starray_uparsed_get_cnt:N`
`\starray_parsed_get_cnt:NNN`	`\starray_uparsed_get_cnt:NN`
`\starray_parsed_get_cnt:NNNTF`	`\starray_uparsed_get_cnt:NNTF`
`\starray_parsed_get_prop:NNn`	`\starray_uparsed_get_prop:Nn`
`\starray_parsed_get_prop:NNnN`	`\starray_uparsed_get_prop:NnN`
`\starray_parsed_get_prop:NNnNTF`	`\starray_uparsed_get_prop:NnNTF`
`\starray_parsed_get_unique_id:NNN`	`\starray_uparsed_get_unique_id:NN`
`\starray_parsed_get_unique_id:NNNTF`	`\starray_uparsed_get_unique_id:NNTF`

9.3 Parsed Commands on Iterate Over

When iterating over a ⟨starray-ref⟩, see 6.2.1, internal variables are set pointing the current (iterated over) starray term: This removes the need, for instance, to execute `\starray_term_parser:n` before using some 'parsed' commands (previous sections). The following commands can be used in combination with the ones from 9.1 and 9.2.

Observe that, in case of nested `\starray_iterate_over:nn`, the `_iparsed_` commands always refer to the immediate `\starray_iterate_over:nn`, meaning that, in 'case A' below, *key-Aa* is a key of the starray ⟨st-A⟩ (⟨st-A[iter st-A]⟩), whilst (case B), *key-Xa* is a key of the starray ⟨st-A.sub-X⟩ (better said: ⟨st-A[iter st-A].sub-X[iter sub-X]⟩

```
\starray_iterate_over:nn {st-A}
  {
    ... \starray_iparsed_get_prop:n {key-Aa}  %% case A

    \starray_iterate_over:NN {st-A.sub-X}
      {
        ... \starray_iparsed_get_prop:n {key-Xa}  %% case B
      }
  }
```

`\starray_iparsed_if_in_p:n` ⋆ `\starray_iparsed_if_in_p:nTF` {⟨key⟩}
`\starray_iparsed_if_in:nTF` ⋆ `\starray_iparsed_if_in:nTF` {⟨key⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2026/02/01

This will test if the given *key* is present in the current, iterated over, term.

> ***Note:*** The predicate version, `_p`, expands to either `\c_true_bool` or `\c_false_bool`.
>
> ***Warning:*** This should only be used in the ⟨code⟩ part of `\starray_iterate_over:nn` or `\starray_iterate_over:nnTF`.

`\starray_iparsed_get_iter:` ⋆ `\starray_iparsed_get_iter:`

new: 2026/02/01

`\starray_iparsed_get_iter:` will place in the current iterator's value, using `\int_use:N`, of the current, iterated over, term in the input stream.

> ***Warning:*** This should only be used in the ⟨code⟩ part of `\starray_iterate_over:nn` or `\starray_iterate_over:nnTF`.

`\starray_iparsed_get_iter:N` `\starray_iparsed_get_iter:N` {⟨int-var⟩}
`\starray_iparsed_get_iter:NTF` `\starray_iparsed_get_iter:NTF` {⟨int-var⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2026/02/01

These will save the iterator's value (of a parsed term) in a integer variable (*expl3*[3]). The ⟨if-true⟩ and ⟨if-false⟩ regards the status of the last `\starray_term_parser:` command, iff the option `parsed check` (see 2) is enable, otherwise it will always execute the ⟨if-true⟩ branch.

> ***Warning:*** This should only be used in the ⟨code⟩ part of `\starray_iterate_over:nn` or `\starray_iterate_over:nnTF`.

`\starray_iparsed_get_cnt:` ⋆ `\starray_iparsed_get_cnt:`

new: 2026/02/01

`\starray_iparsed_get_cnt:` will place the current number of terms, using `\int_use:N`, of the current, iterated over, term, in the input stream.

> ***Warning:*** This should only be used in the ⟨code⟩ part of `\starray_iterate_over:nn` or `\starray_iterate_over:nnTF`.

\starray_iparsed_get_cnt:N \starray_get_cnt:N {⟨integer⟩}
\starray_iparsed_get_cnt:NTF \starray_get_cnt:NTF {⟨integer⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2026/02/01

Similarly to \starray_get_cnt:nN and \starray_get_cnt:nNTF these will save the number of terms (of the current, iterated over, term) in a integer variable (*expl3*[3]). The ⟨if-true⟩ and ⟨if-false⟩ regards the status of the last \starray_term_parser: command, iff the option parsed check (see 2) is enable, otherwise it will always execute the ⟨if-true⟩ branch.

Warning: This should only be used in the ⟨code⟩ part of \starray_iterate_over:nn or \starray_iterate_over:nnTF.

\starray_iparsed_get_prop:n ⋆ \starray_iparsed_get_prop:n {⟨key⟩}

new: 2026/02/01

\starray_iparsed_get_prop:n {⟨key⟩} places the value of ⟨key⟩, if it exists, from the current, iterated over, term, in the input stream.

Warning: This should only be used in the ⟨code⟩ part of \starray_iterate_over:nn or \starray_iterate_over:nnTF.

\starray_iparsed_get_prop:nN \starray_iparsed_get_prop:nN {⟨key⟩} {⟨tl-var⟩}
\starray_iparsed_get_prop:nNTF \starray_iparsed_get_prop:nNTF {⟨key⟩} {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2026/02/01

\starray_iparsed_get_prop:nN {⟨key⟩} {⟨tl-val⟩} stores the value of ⟨key⟩, if it exists, from the current, iterated over, term. The ⟨if-false⟩ branch is executed if ⟨key⟩ doesn't exist or (if the option parsed check, see 2, is enabled) if the last parser operation has failed.

Warning: This should only be used in the ⟨code⟩ part of \starray_iterate_over:nn or \starray_iterate_over:nnTF.

\starray_iparsed_get_unique_id:nN \starray_iparsed_get_unique_id:nN {⟨tl-var⟩}
\starray_iparsed_get_unique_id:nNTF \starray_iparsed_get_unique_id:nNTF {⟨tl-var⟩} {⟨if-true⟩} {⟨if-false⟩}

new: 2026/02/01

Gets an 'unique ID' from the current, iterated over, term. The ⟨if-true⟩ and ⟨if-false⟩ regards the status of the last \starray_term_parser: command, iff the option parsed check (see 2) is enable, otherwise it will always execute the ⟨if-true⟩ branch.

Warning: This should only be used in the ⟨code⟩ part of \starray_iterate_over:nn or \starray_iterate_over:nnTF.

10 Showing (debugging) starrays

\starray_show_def:n \starray_show_def:n {⟨starray-ref⟩}
\starray_show_def_in_text:n \starray_show_def_in_text:n {⟨starray-ref⟩}

Displays the ⟨starray⟩ structure definition and initial property values in the terminal or directly in text.

Note: A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error or invalid ⟨hash⟩.

\starray_show_terms:n \starray_show_terms:n {⟨starray-ref⟩}
\starray_show_terms_in_text:n \starray_show_terms_in_text:n {⟨starray-ref⟩}

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

Note: A warning is raised (see 2) in case of a ⟨starray-ref⟩ syntax error or invalid ⟨hash⟩.

References

[1] Alceu Frigeri. *The pkginfograb Package*. 2025. URL: https://mirrors.ctan.org/macros/latex/contrib/pkginfograb/doc/pkginfograb.pdf (visited on 12/16/2025).

[2] Alceu Frigeri. *The tokglobalstack package*. 2026. URL: https://ctan.org/pkg/tokglobalstack (visited on 02/18/2026).

[3] The LaTeX Project. *The LaTeX3 Interfaces*. 2025. URL: https://mirrors.ctan.org/macros/latex/required/l3kernel/interface3.pdf (visited on 11/20/2025).

[4] The LaTeX Project. *The LaTeX3 Kernel*. 2025. URL: https://ctan.org/pkg/l3kernel (visited on 11/20/2025).