The starray Package
Version 1.3
Alceu Frigeri
June 2023

Abstract
This package implements vector like 'structures', like in '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. For most of the provided functions there is a companion 'branching version'.

Contents
1 Introduction 1
2 Class 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 9
7 Additional Commands and Conditionals 10
8 Showing (debugging) starrays 11

1 Introduction
The main idea is to have an array like syntax when setting/recovering structured information, e.g. \texttt{\textbackslash starray\_get\_prop:nn} \{\langle \texttt{student[2].work[3].reviewer[4]} \rangle \} \{\langle \texttt{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 \texttt{name} \rangle being a property of "reviewer". Moreover one can iterate over the structure, for instance \texttt{\textbackslash starray\_get\_prop:nn} \{\langle \texttt{student\_work\_reviewer} \rangle \} \{\langle \texttt{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 a time) or with a keyval interface or both, either way, one has to first "create a root starray" (\texttt{\textbackslash 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".

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

\textit{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.

\textit{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 Class Options

The package options (\texttt{key=value}) are:

\texttt{prefix} (default: \texttt{\_\_starray\_}). Set the \texttt{prefix} used when declaring the property lists associated with any \texttt{starray}.

\texttt{msg-err} By default, the \texttt{starray} package only generates ”warnings”, with msg-err one can choose which cases will generate ”package error” messages. There are 3 message classes: 1. \texttt{strict} relates to \texttt{\textbackslash starray\_new:n} cases (\texttt{starray} creation); 2. \texttt{syntax} relates to ”term syntax” errors (student\_work\_reviewer in the above examples); finally 3. \texttt{reference} relates to cases whereas the syntax is correct but referring to non-existent terms/properties.

\begin{itemize}
  \item \texttt{none} (default) no package message will raise an error.
  \item \texttt{strict} will raise an error on \texttt{strict} case alone.
  \item \texttt{syntax} will raise an error on \texttt{strict} and \texttt{syntax} cases.
  \item \texttt{reference} will raise an error on \texttt{strict}, \texttt{syntax} and \texttt{reference} cases.
  \item \texttt{all} will raise an error on all cases.
\end{itemize}

\texttt{msg-suppress} ditto, to suppress classes of messages:

\begin{itemize}
  \item \texttt{none} (default) no package message will be suppressed.
  \item \texttt{reference} only \texttt{reference} level messages will be suppressed.
  \item \texttt{syntax} \texttt{reference} and \texttt{syntax} level messages will be suppressed.
  \item \texttt{strict} \texttt{reference}, \texttt{syntax} and \texttt{strict} level messages will be suppressed.
  \item \texttt{all} all messages will be suppressed.
\end{itemize}

3 Creating a starray

\texttt{\textbackslash starray\_new:n} \texttt{\{}\texttt{\textbackslash starray\}} \texttt{\}\texttt{\{}\texttt{if-true}\}\texttt{\{}\texttt{if-false}\}

Creates a new \texttt{\{}\texttt{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 \texttt{\{}\texttt{starray\_root}\} or just \texttt{\{}\texttt{root}\}.

\textit{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 ({\{\texttt{array}\}})
\starray_if_exist:nTF ★ \starray_if_exist:nTF {\{\texttt{array}\}} {\{\texttt{if-true}\}} {\{\texttt{if-false}\}}
\starray_if_valid_p:n ★ \starray_if_valid:nTF {\{\texttt{array}\}} {\{\texttt{if-true}\}} {\{\texttt{if-false}\}}

\starray_if_exist:n ★
\starray_if_valid:n ★

\new: 2023/05/20

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

\textbf{Note:} The predicate versions, \_\_p, expand to either \texttt{prg\_return\_true}: or \texttt{prg\_return\_false}:.

4 Defining and initialising a \texttt{array} structure

\starray_def_prop:nnn \starray_def_prop:nnn {\{\texttt{array-ref}\}} {\{\texttt{prop-key}\}} {\{\texttt{initial-value}\}}
\starray_def_prop:nnnTF \starray_def_prop:nnnTF {\{\texttt{array-ref}\}} {\{\texttt{prop-key}\}} {\{\texttt{initial-value}\}} {\{\texttt{if-true}\}} {\{\texttt{if-false}\}}

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

The definition/assignment of a \texttt{(prop-key)} to a \texttt{(array-ref)} is global.

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

\starray_def_structure:nn \starray_def_struct:nn {\{\texttt{array-ref}\}} {\{\texttt{struct-name}\}}
\starray_def_structure:nnTF \starray_def_struct:nnTF {\{\texttt{array-ref}\}} {\{\texttt{struct-name}\}} {\{\texttt{if-true}\}} {\{\texttt{if-false}\}}

\texttt{Add} a sub-structure (a \texttt{array} in itself) to \texttt{(array-ref)} (see 5.1). If \texttt{(struct-name)} is already present nothing happens. The definition/assignment of a \texttt{(struct-name)} to a \texttt{(array-ref)} is global.

\textbf{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).

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

\starray_def_from_keyval:nn \starray_def_from_keyval:nn {\{\texttt{array-ref}\}} {\{\texttt{keyval-lst}\}}
\starray_def_from_keyval:nnTF \starray_def_from_keyval:nnTF {\{\texttt{array-ref}\}} {\{\texttt{keyval-lst}\}} {\{\texttt{if-true}\}} {\{\texttt{if-false}\}}

\texttt{Add} a set of \texttt{(keys) / (values)} and/or \texttt{(structures)} to \texttt{(array-ref)} (see 5.1). The \texttt{(keyval-lst)} is pretty straightforward, the construction \texttt{(key)} . \texttt{struct} denotes a nested structure :
\texttt{\textbackslash starray\_def\textunderscore from\_keyval\_nn \{root.substructure\}}

\begin{verbatim}
{keyA = valA ,
 keyB = valB ,
 subZ . struct =
 {keyZA = valZA ,
  keyZB = valZB ,
  }
 subY . struct =
 {keyYA = valYA ,
  keyYB = valYB ,
  subYYY . struct =
   {keyYYYa = valYYYa ,
    keyYYYb = valYYYb
   }
  }
}
\end{verbatim}

The definitions/assignments to \texttt{\{starray-ref\}} are all global.

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

\section*{4.1 Fixing an ill-instantiated starray}

When instantiating (see 5) a \texttt{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 \texttt{starray} (better said, if one adds a sub-structure), is a quark loop (from \texttt{l3quark}). To avoid a quark loop it is necessary to "fix" the structure of already instantiated terms.

\texttt{\textbackslash starray\_fix\_terms\_nn \{\{starray-ref\}\}}

The sole purpose of this function is to "fix" the already instantiated terms of a \texttt{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 \texttt{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.

\section*{5 Instantiating starray terms}

\texttt{\textbackslash starray\_new\_term\_nn \{\{starray-ref\}\}}
\texttt{\textbackslash starray\_new\_term\textunderscore nn \{\{starray-ref\}\} \{\{hash\}\}}
\texttt{\textbackslash starray\_new\_term\_nTF \{\{starray-ref\}\} \{\{if\_true\}\} \{\{if\_false\}\}}
\texttt{\textbackslash 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 \texttt{\{starray-ref\}}. Note that the newly created term will have all properties (key/values) as defined by the associated \texttt{\textbackslash starray\_prop\_def\_nn \{\{starray-ref\}\}}, with the respective "initial values". For instance, given the following
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 5.1). Also note that no term of kind (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.

5.1 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 5:
Note that, all it’s needed in order to be able to use \(\langle \text{starray-root} \rangle\) is that \(\langle \text{sub-A} \rangle\) is an already declared sub-structure of \(\langle \text{starray-root} \rangle\). 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:

\(\langle \text{starray-root} \rangle\langle \text{idx} \rangle\langle \text{sub-A} \rangle\langle \text{idxA} \rangle\langle \text{sub-B} \rangle\langle \text{idxB} \rangle\)

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:

\(\langle \text{starray-root} \rangle\langle \text{sub-A} \rangle\langle \text{idxA} \rangle\langle \text{sub-B} \rangle\)

in which case, one is using the "iterator" of \(\langle \text{starray-root} \rangle\) 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 \(\langle \text{starray-root} \rangle\) one has a first counter (representing, indexing the root structure terms), then for all sub-structures of \(\langle \text{starray-root} \rangle\) one will have an additional counter for every term of \(\langle \text{starray-root} \rangle\)!

So, for example:

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:

\[ \text{starray} \text{new:n} \{ \text{st-root} \} \]
\[ \text{starray} \text{def:struct:nn} \{ \text{st-root}\{\text{subZ}\} \} \]
\[ \text{starray} \text{def:struct:nn} \{ \text{st-root}\{\text{subY}\} \} \]
\[ \text{starray} \text{def:struct:nn} \{ \text{st-root.subY}\{\text{subYYY}\} \} \]

\[ \text{starray} \text{new:term:n} \{ \text{st-root}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[1].subZ}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[1].subY}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[1].subY}\{\text{subYYY}\} \} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[1].subY}\} \]

\[ \text{starray} \text{new:term:n} \{ \text{st-root}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[2].subZ}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[2].subZ}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[2].subY}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root[2].subY}\{\text{subYYY}\} \} \]

\[ \text{starray} \text{new:term:n} \{ \text{st-root}\{\text{hash-A}\} \} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root}\{\text{subZ}\}\} \]
\[ \text{starray} \text{new:term:n} \{ \text{st-root}\{\text{hash-A}\}.\{\text{subZ}\}\} \]

Will create 3 \(\text{subZ}\) terms associated with the first (index = 1) \(\text{st-root}\).

### 5.2 iterators

\[ \text{starray} \text{set:iter:nn} \{\langle \text{starray-ref}\rangle\}\{\langle \text{int-val}\rangle\}\{\langle \text{if-true}\rangle\}\{\langle \text{if-false}\rangle\} \]
\[ \text{starray} \text{reset:iter:nn} \{\langle \text{starray-ref}\rangle\}\{\langle \text{if-true}\rangle\}\{\langle \text{if-false}\rangle\} \]
\[ \text{starray} \text{next:iter:nn} \{\langle \text{starray-ref}\rangle\}\{\langle \text{if-true}\rangle\}\{\langle \text{if-false}\rangle\} \]

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 \(\text{starray-ref}\) syntax error. The branching version doesn’t raise any warning.
Before the reset \(\langle \text{st-root.subY.subYYY}\rangle\) was equivalent to \(\langle \text{st-root[2].subY[2].subYYY[2]}\rangle\), 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 \(\langle \text{subYYY}\rangle\)) where reseted, and therefore \(\langle \text{st-root.subY.subYYY}\rangle\) was then equivalent to \(\langle \text{st-root[2].subY[1].subYYY[1]}\rangle\).

**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.

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

**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.
\texttt{\textbackslash starray\_parsed\_get\_cnt:}\star \texttt{\textbackslash starray\_parsed\_get\_cnt:}

\texttt{\textbackslash starray\_parsed\_get\_cnt:} will place the current number of terms, using \texttt{\textbackslash int\_use:}\texttt{N}, of the last parsed term, in the input stream.

\textbf{Warning:} This can be used after any command which 'parses a term', for instance \texttt{\textbackslash 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.

### 6 Changing and recovering starray properties

\begin{align*}
\texttt{\textbackslash starray\_set\_prop:nnn} & \quad \texttt{\textbackslash starray\_set\_prop:nnn} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\} \\
\texttt{\textbackslash starray\_set\_prop:nnV} & \quad \texttt{\textbackslash starray\_set\_prop:nnV} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\} \\
\texttt{\textbackslash starray\_set\_prop:nnnTF} & \quad \texttt{\textbackslash starray\_set\_prop:nnnTF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\} \\
\texttt{\textbackslash starray\_set\_prop:nnnVF} & \quad \texttt{\textbackslash starray\_set\_prop:nnnVF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\} \\
\texttt{\textbackslash starray\_gset\_prop:nnn} & \quad \texttt{\textbackslash starray\_gset\_prop:nnn} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\} \\
\texttt{\textbackslash starray\_gset\_prop:nnV} & \quad \texttt{\textbackslash starray\_gset\_prop:nnV} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\} \\
\texttt{\textbackslash starray\_gset\_prop:nnnTF} & \quad \texttt{\textbackslash starray\_gset\_prop:nnnTF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\} \\
\texttt{\textbackslash starray\_gset\_prop:nnnVF} & \quad \texttt{\textbackslash starray\_gset\_prop:nnnVF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash prop\_key}\}\{\texttt{\textbackslash value}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\}
\end{align*}

Those are the functions that allow to (g)set (change) the value of a term's property. If the \{\texttt{\textbackslash prop\_key}\} isn’t already present it will be added just for that term \texttt{\textbackslash starray\_ref}. The \{\texttt{\textbackslash nnV}\} variants allow to save any variable like a token list, property list, etc...

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

\begin{align*}
\texttt{\textbackslash starray\_set\_from\_keyval:nnn} & \quad \texttt{\textbackslash starray\_set\_from\_keyval:nnn} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash keyval\_lst}\} \\
\texttt{\textbackslash starray\_set\_from\_keyval:nnnTF} & \quad \texttt{\textbackslash starray\_set\_from\_keyval:nnnTF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash keyval\_lst}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\} \\
\texttt{\textbackslash starray\_set\_from\_keyval:nnnVF} & \quad \texttt{\textbackslash starray\_set\_from\_keyval:nnnVF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash keyval\_lst}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\} \\
\texttt{\textbackslash starray\_gset\_from\_keyval:nnn} & \quad \texttt{\textbackslash starray\_gset\_from\_keyval:nnn} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash keyval\_lst}\} \\
\texttt{\textbackslash starray\_gset\_from\_keyval:nnnTF} & \quad \texttt{\textbackslash starray\_gset\_from\_keyval:nnnTF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash keyval\_lst}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\} \\
\texttt{\textbackslash starray\_gset\_from\_keyval:nnnVF} & \quad \texttt{\textbackslash starray\_gset\_from\_keyval:nnnVF} \{\texttt{\textbackslash starray\_ref}\}\{\texttt{\textbackslash keyval\_lst}\}\{\texttt{\textbackslash if\_true}\}\{\texttt{\textbackslash if\_false}\}
\end{align*}

it is possible to set a collection of properties using a key/val syntax, similar to the one used to define a \texttt{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 \texttt{\textbackslash st\_root} and then \texttt{\textbackslash 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\langle\langle starray-ref\rangle\rangle \{\langle\langle key\rangle\rangle\}
\starray_get_prop:nnN\langle\langle starray-ref\rangle\rangle \{\langle\langle key\rangle\rangle\} \{\langle\langle tl-var\rangle\rangle\}
\starray_get_prop:nnTF\langle\langle starray-ref\rangle\rangle \{\langle\langle key\rangle\rangle\} \{\langle\langle if-true\rangle\rangle\} \{\langle\langle if-false\rangle\rangle\}

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

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

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

\starray_parsed_get_prop:n ★ \starray_parsed_get_prop:nn \{\langle\langle key\rangle\rangle\}
\starray_parsed_get_prop:n \{\langle\langle key\rangle\rangle\} places the value of \langle\langle key\rangle\rangle, 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.

7 Additional Commands and Conditionals

\starray_if_in:nnTF \starray_if_in:nnTF \{\langle\langle starray-ref\rangle\rangle\} \{\langle\langle key\rangle\rangle\} \{\langle\langle if-true\rangle\rangle\} \{\langle\langle if-false\rangle\rangle\}
\starray_term_syntax:nTF \starray_term_syntax:nTF \{\langle\langle starray-ref\rangle\rangle\} \{\langle\langle if-true\rangle\rangle\} \{\langle\langle if-false\rangle\rangle\}

The \starray_if_in:nnTF \{\langle\langle starray-ref\rangle\rangle\} \{\langle\langle key\rangle\rangle\} \{\langle\langle if-true\rangle\rangle\} \{\langle\langle if-false\rangle\rangle\} tests if a given \langle\langle key\rangle\rangle is present, whilst \starray_term_syntax:nTF \{\langle\langle starray-ref\rangle\rangle\} \{\langle\langle if-true\rangle\rangle\} \{\langle\langle if-false\rangle\rangle\} just verifies if the \langle\langle starray-ref\rangle\rangle syntax is valid or not, branching accordly.
This will parse a \texttt{starray-ref} reference, and set interval variables so that commands like \texttt{starray-parsed} can be used.

\textbf{Warning:} The main idea is to allow some expandable commands, but be aware that all \texttt{starray} commands that use a \texttt{starray-ref} use the very same parser variables.

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{\_prg\_return\_true}: or \texttt{\_prg\_return\_false}.

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

\section{Showing (debugging) starrays}

Displays the \texttt{starray} structure definition and initial property values in the terminal or directly in text.

Displays the \texttt{starray} instantiated terms and current property values in the terminal or directly in text.