Consensus

Reference Manual

Patrick Bouchaud, 2017-03-28

Table of Contents

1. Consensus entities

Each Consensus entity is uniquely identified by an `expression` which takes the general form:

```
source-medium->target
```

where

`source`, `medium` and `target` can be either

	nothing
`identifier`	an identifier (name) or group of identifiers enclosed in double-quotes (“)
`[ expression ]`	in the form `[ source-medium->target ]`

- identifiers are also entities.
- Each Consensus entity, once instantiated, can be in either one of two states: active, or inactive.
- Consensus entities are inactive by default.

2. Consensus queries

Consensus entities can be queried using a query expression which takes the general form:

```
mark : source-medium->target : instance
```

where

`mark` is optional and may appear only once in the overall expression

`source`, `medium`, `target` and `instance` can be either

	nothing
`.`	the dot (`.`) character, to represent ***any***
`identifier`	an identifier (name) or group of identifiers enclosed in double-quotes (")
`[ expression ]`	in the form `[ source-medium->target : instance ]` or `[ mark : source-medium->target : instance ]`

nothing translates

in the case of a `source`, `medium` or `target` term: into the Consensus ***nil*** entity.
in the case of an `instance` term, and in the absence of the tilde (~) prefix: into ***any***.

some or all of the terms can be, in that order

prefixed by the tilde (~) character, to represent ***not***

prefixed by either the star (`*`) or the underline (_) character, to represent respectively ***active*** and ***inactive***

- In the absence of any `mark`, the question mark (`?`) character, without any prefix, may also replace one, and only one, of either a `source`, `medium` or `target` term of the overall expression.

  In this case the query will return the list of entities matching that specific term, while keeping the overall expression successful.

- Consensus allows a variety of query expressions along the general principle presented here.

  The complete list of possible query expressions together with their detailed syntax is provided in the Appendix A of this document.

3. Consensus back-pointer notation

While entities are stored using the fore-pointer notation

```
source-medium->target
```

Consensus allows to reference, query and output entities using the back-pointer notation

```
target<-medium-source
```

and to mix fore- and back- pointer notations within the same expression.

This feature is key with respect to Consensus current proximity and future evolution towards natural language.

4. Consensus query operations

Consensus query expressions generalize the Consensus entity basic representation, and constitute the basis of the Consensus programming language.

They are used to perform the following operations:

Consensus entity base operations:

```
!! expression
!~ expression
!* expression
!_ expression
```

Consensus variable assignment operations:

```
: variable : expression
: variable : %[ expression ]
: variable : !! expression
: variable : !~ expression
: variable : !* expression
: variable : !_ expression
: variable : narrative()
: variable : %[ expression ].narrative()
: variable : !! %[ expression ].narrative()
: variable : !~ %[ expression ].narrative()
: variable : !* %[ expression ].narrative()
: variable : !_ %[ expression ].narrative()
```

Consensus output operations:

```
>: %[ expression ]
>: %[ expression ].$( account )
>: %[ expression ].narrative()
```

Consensus loop operations:

```
? : expression
? : %[ expression ]
```

Consensus conditional execution operations:

```
? %: expression
? %: %[ expression ]

?~ %: expression
?~ %: %[ expression ]
```

Consensus narrative base operations:

```
!! %[ expression ].narrative()
!~ %[ expression ].narrative()
!* %[ expression ].narrative()
!_ %[ expression ].narrative()
```

Consensus narrative events and conditions:

```
in ( expression )
on ( event: expression op )
on ( event < source >: expression op )
on ( event < source >: op expression )
```

Further descriptions, as well as the detailed syntax of each of these operations, are provided in the remainder of this document.

5. Consensus entity base operations

Consensus allows to perform the following base operations on entities:

`!! expression`	instantiates the entity represented by `expression`
`!~ expression`	releases the entities resulting from the query `expression`
`!* expression`	activates the entities resulting from the query `expression`
`!_ expression`	deactivates the entities resulting from the query `expression`

- The ***instantiate*** (`!!`) operation will instantiate - as needed - the whole graph of entities involved in `expression`, down to and including the base identifiers.

  This operation will return with an error if either or both of the terms ***any*** (`.`) or ***mark*** (`?`) are found in the expression. Otherwise it will return the top level entity created.

- The ***release*** (`!~`), ***activate*** (`!*`) and ***deactivate*** (`!_`) operations will apply only to the top level entity - aka. ***relationship instance*** - of each of the results of the query `expression`.

  These operations will return with an error if `expression` contains any term which does not match an instantiated entity, up to and including the overall relationship instance. Otherwise

  the ***activate*** and ***deactivate*** operations will return the results of the query `expression`, regardless of whether the operation actually triggered any state change of the entities

  the ***release*** operation will return null.

6. Consensus variables

Consensus allows to use variables in order to

1. build complex query expressions while keeping these expressions readable.
2. transport and reuse query results.

Consensus variables values can be set to either one of the following:

`: variable : %[ expression ]`	assigns `expression` query results to `variable`
`: variable : expression`	assigns `expression` to `variable`

the difference being, that in the first case the entities corresponding to the variable value are evaluated once and for all upon the variable assignment, whereas in the second they are evaluated each and every time the variable value has to be resolved.

- `%variable` can then be used as a term of query expressions, that is: the variable name prefixed by the character percent (%) is used to reference the variable value.

Consensus allows a variety of variable value assignment operations along the general principle presented here.

The complete list of possible variable value assignment operations together with their detailed syntax is provided in the next section.

7. Consensus variable operations

Consensus supports the following operations on variables:

`: variable : value`	assigns `value` to `variable`
`:~ variable`	resets variable value to null (non-existing)
`%variable`	retrieves the value of `variable`

where

`variable` can be any identifier (name) except those enclosed in double-quotes (").

`value` can be either one of the following:

`expression`	sets `variable` value to the provided expression
`%[ expression ]`	sets `variable` value to the `expression` query results
`!! expression`	sets `variable` value to `expression` instantiation result
`!~ expression`	sets `variable` value to null (non-existing)
`!* expression`	sets `variable` value to the `expression` query results
`!_ expression`	sets `variable` value to the `expression` query results
`narrative()`	sets `variable` value to the specified `narrative` from the Consensus ***nil*** entity `narratives` value account

the following assignment operations set the variable value to the specified `narrative` from the `narratives` value account of the entities resulting from the query `expression`:

```
: variable : %[ expression ].narrative()
: variable : !! %[ expression ].narrative()
: variable : !~ %[ expression ].narrative()
: variable : !* %[ expression ].narrative()
: variable : !_ %[ expression ].narrative()
```

the following assignment operation sets the variable value to the ***literal*** value of the query `expression` results. It is used mainly for testing purpose:

```
: variable : %[ expression ].$( literal )
```

- When used as term of an expression, `%variable` will cause the expression resolution to return with an error if the variable value is narrative.

- In the absence of a variable name, the ***reset*** variable operation (`:~`) will reset all variables in the current scope to null (non-existing).

8. Consensus value accounts

In the future, Consensus will allow any entity to act as a Value Bank (`$$`) and hold value accounts (`$`) for other entities.

To date however the implementation only supports the Consensus default Value Bank, which holds Consensus system-specific value accounts.

Consensus allows to set these accounts values for any entity, as follows:

`: %[ expression ].$( account ): value`	assigns `value` to the `account` of the entities resulting from the query `expression`
`: %[ expression ].$( account ): %variable`	assigns `variable` value to the `account` of the entities resulting from the query `expression`

where

`account` can be either one of the following:

`narratives`	`value` (resp. variable value) must then be a list of instantiated narratives
`url`	`value` (resp. variable value) must then be an identifier enclosed in double-quotes (")
`hcn`	`value` (resp. variable value) must then be an identifier enclosed in double-quotes (")

These expressions will be extended in the future to support `%.$( account )` and cover all value accounts associated with the ***nil*** entity

The purpose and usage of these accounts and of their values are detailed in the following sections of this document.

9. Consensus input & output operations

In the future, Consensus will support the following input & output operations:

Input

`: format < stream`	reads input from `stream` and converts it according to the specified `format`

Output

`> stream: format`	produces and writes output to `stream` according to the specified `format`
`> variable: value`	adds `value` to the list of values held by `variable`
`> %[ expression ].$( account ): value`	adds `value` to the list of values held by the specified `account` (e.g. `narratives`) of the entities resulting from the query `expression`

To date however the implementation supports only a subset of these operations.

The subset of currently supported Consensus input & output operations are described in the following sections of this document.

10. Consensus pipe input operation

Consensus allows to execute a UNIX shell command and read in the command's standard output as Consensus commands, as follows:

`:< %( command )`	executes the UNIX shell `command` and reads in `command`'s standard output as Consensus commands

For security reasons it is doubtful whether this functionality should remain as-is in the final product.

11. Consensus standard output operations

Consensus allows to write to standard output, using the following command:

`>: format` produces and writes output to standard output according to the specified `format`

where

`format` can be any sequence of characters (string), interpreted as follows:

Upon encountering the character percent (`%`) outside a character sequence enclosed in double-quotes, Consensus will:

if the next character is the character backslash (`\`):

output the character percent (`%`) and ignore the backslash.

if the next character is the question mark (`?`) resp. exclamation mark (`!`) character:

format the output according to the current variator (`%?`) resp. ***this*** (`%!`) variable value.

if the next character is the dot (`.`) character:

interpret the sequence starting from the percent character as a `%.narrative()` query, run the query, and format the output according to the query results.

if the next character is the opening bracket (`[`) character:

interpret the sequence starting from the percent character as either a

```
%[ expression ]
%[ expression ].$( account )
%[ expression ].narrative()
```

query, run the query, and format the output according to the query results.

if the subsequent character sequence is an acceptable variable identifier

interpret the sequence starting from the percent character as a %variable query and format the output according to the variable value.

otherwise

output both the character percent (`%`) and the subsequent character.

In all the cases above, Consensus resumes parsing the `format` string after the operation is performed.

Otherwise, Consensus will:

output the character to standard output. If the character is a newline character, Consensus will then return from the standard output command.

12. Consensus HTML-HCN output operation

Consensus allows to write the HTML value of an entity to standard output, as follows:

`>: %[ expression ].$( html )`	writes the HTML value of the entities resulting from the `expression` query to standard output, based on their `hcn` value.

The `hcn` value of an entity is a string of characters representing the complete path to the HCN file which is to be interpreted by the Consensus HCN parser when the Consensus HTML-HCN operation is performed on this entity.

The Consensus HCN file format consists of the standard HTML file format with the following two additions:

1. It supports Consensus tags, which are sequences of Consensus commands separated by the newline character.

   Consensus tags are specified by enclosing them in the double-characters << and >>, the same way standard HTML tags are enclosed in the single characters < and >.

   Upon encountering a Consensus tag, the Consensus HCN parser will execute the enclosed Consensus commands. The variable ***this*** (`%!`) in this case and outside any narrative-building operation references the entity whose HTML output is being produced.

2. It ascribes a special meaning to the percent (`%`) character used outside of Consensus tags.

   Upon encountering the percent (`%`) character in the text of an HCN file outside a Consensus tag, the Consensus HCN parser will interpret the subsequent characters and write the output as per its standard output operations. The variable ***this*** (`%!`) in this case references the entity whose HTML output is being produced.

The Consensus HCN parser will output any other character sequence from the HCN file to the standard output exactly as it is.

13. Consensus conditional execution mode & operations

Consensus allows to specify a condition for the execution of subsequent commands, as follows:

`? %variable`	
`?~ %variable`	allows resp. disallows execution of subsequent commands if `variable` has been `assigned`.
`? %[ expression ]`	
`?~ %[ expression ]`	allows resp. disallows execution of subsequent commands if the `query expression` results reference instantiated entities.
`? %: expression`	
`?~ %: expression`	allows resp. disallows execution of subsequent commands if the `query expression` yields result, which can be either entities or sub-expressions (literals) depending on whether a filter instance is used in `expression` (see Appendix A).
`/~`	sets the current condition to its contrary, that is: disables execution of subsequent commands if command execution was previously allowed, and enables execution of subsequent commands if command execution was previously disabled.
`/`	exits conditional execution mode.

This functionality in conjunction with the Consensus loop execution mode & operations (see next page) is particularly useful in the context of Consensus HTML-HCN output operations.

14. Consensus loop execution mode & operations

Consensus allows to execute commands in a loop over the results of a query expression, as follows:

`?: expression`	initiates loop over the query `expression` results, which can be either entities or sub-expressions depending on whether a filter instance is used in `expression` (see Appendix A).
`?: %[ expression ]`	initiates loop over the entities referenced by the query `expression` results
`/`	exits loop

- Expression results can be accessed individually through the Consensus variator variable (`%?`) in-between the ***initiate*** and ***exit loop*** commands.

- The Consensus variator variable (`%?`) supports all Consensus variable operations, e.g. it can be used as a term in query expressions, its value can be re-assigned to other variables, etc.

15. Consensus narratives

Consensus allows to build narratives and to assign them to entities via their `narratives` value account.

> Narratives are essentially programs in the traditional sense of the term, except that, from both the programming and execution standpoints, time can no longer be relied upon to ensure that the program's instructions execute one after the other. In other words, Consensus narrative instructions cannot be assumed to execute linearly, one after the other, in sequence. No more - nor less - than this is the way things happen in reality. Here, instead of time, we have change. And change propagates not only in one but in all directions, from one entity to all connected others, first, within the system, and then outside - and then what goes around, as the saying goes, comes around...

To allow Consensus narratives, which are programs, to handle these changes, Consensus introduces the notion of frame.

> A Consensus frame is a snapshot at the present instant of both the current state of the system (that is, of all the entities in their present state), and of all the changes that happened to it since its own last frame. It is based on this information, and on this information only, that each entity narrative determines what changes must be performed and affect what part of the system, from the entity's standpoint, up until next frame.

This obviously requires serious consideration up-front of all the changes susceptible to affect the entity from others, and conversely of the effects of the entity's changes on all connected others. Which is fine, as it is precisely what programmers are supposed to do, up-front, so that their program has a chance to render any significant service when entering an information highway such as the Internet, or other.

Which is also the reason why this method is called Consensus.

16. Consensus narrative base operations

Consensus allows to perform the following base operations on narratives:

`!! %[ expression ].narrative()`	enters narrative-building mode and adds an instance of the built `narrative` to the `narratives` value account of each entity resulting from the query `expression`
`!~ %[ expression ].narrative()`	removes the specified `narrative` instance from the `narratives` value account of each entity resulting from the query `expression`
`!* %[ expression ].narrative()`	activates the specified `narrative` instance of the `narratives` value account of each entity resulting from the query `expression`
`!_ %[ expression ].narrative()`	deactivates the specified `narrative` instance of the `narratives` value account of each entity resulting from the query `expression`

where

`narrative` can be any identifier (name) except those enclosed in double-quotes (")

Consensus also allows to perform the above operations on the Consensus ***nil*** entity, as follows:

`!! narrative()`	enters narrative-building mode and adds an instance of the built `narrative` to the `narratives` value account of the Consensus ***nil*** entity
`!~ narrative()`	removes the specified `narrative` instance from the `narratives` value account of the Consensus ***nil*** entity
`!* narrative()`	activates the specified `narrative` instance of the `narratives` value account of the Consensus ***nil*** entity
`!_ narrative()`	activates the specified `narrative` instance of the `narratives` value account of the Consensus ***nil*** entity

Consensus allows to assign narratives to variables, and from there to the `narratives` value account of other entities.

17. Consensus narrative building operations

Upon the ***instantiate*** (`!!`) narrative command Consensus enters narrative-building mode, where the narrative conditions, events and actions can be specified, as follows:

`in` *`condition`*	specifies a narrative ***condition*** occurrence
`on` *`event`*	specifies a narrative ***event*** occurrence
`do` *`action`*	specifies a narrative ***action*** occurrence
`then`	specifies a narrative ***then*** occurrence
`/`	closes the latest narrative ***statement***

where

`condition` may consist of either

`expression`	a single query expression
`( expression1, expression2, ... )`	a comma-separated list of query expressions

`event` may consist of either

`event`	a single narrative event
`(` *`event1, event2,`* `...` `)`	a comma-separated list of narrative events

`action` may consist of either

`command`	1. a single Consensus command, or `exit`
	2. a newline character, followed by

`command1`
`command2`
`...`
	a newline-separated list of Consensus commands,
	closed on a new line by

`/.`	the narrative ***action block*** closing instruction.

- Narrative ***condition*** expressions and narrative ***events*** are evaluated as part of the Consensus system frame.

- Narrative actions are executed as part of the Consensus system frame. The entity whose narrative is being executed can be accessed from the narrative actions through the Consensus ***this*** variable (`%!`).

- The Consensus ***this*** variable (`%!`) supports all Consensus variable operations, e.g. it can be used as a term in query expressions, its value can be re-assigned to other variables, etc.

The narrative structure, narrative events and the Consensus narrative ***then*** occurrence are further detailed in the remaining sections of this document.

18. Consensus narrative statements

Narratives are graphs of occurrences, where each occurrence can be either one of the following:

`in` *`condition`*	specifies a narrative ***condition*** occurrence
`on` *`event`*	specifies a narrative ***event*** occurrence
`do` *`action`*	specifies a narrative ***action*** occurrence
`then`	specifies a narrative ***then*** occurrence

A Consensus narrative ***statement*** is a single narrative thread of the narrative graph, starting from the narrative root and finishing at a leaf (***action***) of the narrative graph.

The Consensus narrative-building interface will enter a new occurrence specification level upon encountering the newline character after either a narrative ***condition*** or a narrative ***event*** has been specified, in which case the Consensus narrative-building and Consensus narrative output interfaces will increase the indentation and specification levels of subsequent narrative occurrences by one. These same interfaces will decrease the narrative occurrence specification and indentation levels by one upon encountering the Consensus narrative statement closing instruction made, on a new line, of the character slash (`/`) followed by the newline character.

Note that by this rule it is possible e.g. to specify whole narrative statements (resp. a narrative condition followed by a narrative event) in one line, terminated by the newline character, without increasing the narrative occurrence specification level (resp. without increasing the narrative occurrence specification level by more than one level).

Consensus only allows one series of narrative ***events*** occurrences per narrative statement, provided that these are consecutive, and unless the narrative ***then*** occurrence separates the new narrative ***event*** occurrence from the last one.

Upon closing a narrative statement the Consensus narrative-building interface will return to the previous occurrence specification level or, if at root, exit narrative-building mode.

19. Consensus narrative events

In the future Consensus will support the following events:

Internal

Notifications

```
identifier: expression op
identifier: %[ expression ].$( account ): value op
```

External

Notifications

```
identifier < source >: expression op
identifier < source >: %[ expression ].$( account ): value op
```

Requests

```
identifier < source >: op expression
identifier < source >: op %[ expression ].$( account ): value
```

To date however Consensus only supports the subset of internal events which are not related to value banking operations.

These are described on the next page.

20. Consensus internal narrative events

Consensus allows to specify the following internal narrative events:

```
init
identifier: expression op
```

where

`init` is generated every time the narrative is activated.

`identifier` (optional) is a variable name.

op can be either one of the following:

`!!`	entities matching the query `expression` have been instantiated
`!~`	entities matching the query `expression` have been released
`!*`	entities matching the query `expression` have been activated
`!_`	entities matching the query `expression` have been deactivated

- The ***instantiated***, ***activated*** or ***deactivated*** entities can be accessed from the narrative actions using the narrative event variable `%identifier`, as illustrated below:

```
on digit: ?-is->[ digit<-has-%keyboard ] !*
       do !* %digit-is->[ value<-has-calculator ]
```

- In case of a ***release*** event notification the associated event variable will not hold entities, as these no longer exist. Instead, the narrative event variable will hold expressions representing the released entities. These expressions are called ***literals***.

- Consensus allows to perform the same operations on ***literal*** variables as are allowed on entity variables, e.g. they can be used as terms in query expressions, their value can be re-assigned to other variables, etc. See Appendix A for more information on literals.

Consensus internal events do not include requests, as Consensus narrative actions are executed without delay during the Consensus system frame. Proper synchronization can only be ensured by proper design of the overall system narratives.

21. Consensus narrative ***then***

Consensus allows a narrative ***then*** occurrence to follow a narrative action occurrence, as in

```
on event in condition do action then ...
```

or

```
on event
       in condition1 do action1
       in condition2 do action2
       then ...
```

- A Consensus narrative ***then*** occurrence must be followed by either a single narrative statement starting on the same line, or by a block of narrative statements starting on the next line - at the next narrative specification and indentation level.

- In narrative-building mode, Consensus will only return to the narrative ***then*** specification level if a narrative action occurrence exists on the same line prior to the narrative ***then*** occurrence. Otherwise - that is: if the narrative ***then*** occurrence started a new line - Consensus will return to the specification and indentation level immediately preceding the latest narrative ***then*** occurrence level.

`on event`
`  do action1 then`
`  in condition1 do action2`
`  /`
`  –`
	`on event`
	`  in condition1 do action1`
	`  in condition1 do action2`
	`  then do action3`
	`–`

- During system frame execution, the Consensus narrative statements immediately following a narrative ***then*** occurrence will become active only for the one frame immediately following the current frame, if at least one of the narrative actions immediately preceding the narrative ***then*** occurrence has been effectively triggered.

The purpose of the Consensus narrative ***then*** occurrences is to allow to build pipelines of operations spanning over several system frames. It is not the same as the sequential execution of instructions in a traditional programming language, as in Consensus the narrative statements preceding the narrative ***then*** occurrence are still active and possibly executing during the next frame.

Appendix A. 1. Consensus query expressions

Consensus entities can be queried using a query expression which can take one of the following forms:

```
source
source : instance
source-medium->target : instance
mark : source-medium->target : instance
```

where

`mark` is optional and may appear only once in the overall expression including its sub- expression terms, not considering those hidden in variables. `mark` can be either

	nothing
`?|???`	a combination of question mark (`?`) and/or dot (`.`) characters, indicating which of resp. the `instance`, `source`, `medium` and/or `target` results of this particular sub- expression should be returned by the query, while keeping the overall expression successful.

`source`, `medium`, `target` and `instance` can be either

	nothing
`.`	the dot (`.`) character, representing ***any***.
`identifier`	an identifier (name) or group of identifiers enclosed in double-quotes (").
`%variable`	a reference to the identified `variable` value, which can be either an expression or a list of entities or literals.
`[ expression ]`	where `expression` can take any query expression form, with the restriction on `mark` mentioned above.

nothing translates

in the case of a `source`, `medium` or `target` term: into the Consensus ***nil*** entity.
in the case of an `instance` term, and in the absence of the tilde (~) prefix: into ***any***.

some or all of the `source`, `medium`, `target` and/or `instance` terms can be, in that order

prefixed by the tilde (~) character, to represent ***not***

prefixed by either the star (*) or the underline (_) character, to represent respectively ***active*** and ***inactive***.

Consensus query expressions can be translated as:

> ***search the database for the entities matching the term*** `source` (***resp.*** `source-medium->target`) ***within the list of entities matching the term*** `instance` ***; then from within the results, search the database and return the entities matching the sub- expression term which is preceded by*** `mark`***.***

- In the absence of any `mark`, the question mark (`?`) character, without any prefix, may also replace one, and only one, of either a `source`, `medium` or `target` term of the overall expression.

  In this case the query will return the list of entities matching that specific term, while keeping the overall expression successful.

- When the `instance` term of an expression is a variable holding literals, and the expression has to be resolved, Consensus will extract and return literals from the variable value according to the complementary `source`, `medium` and `target` terms of the expression.

  Consensus solver will convert literals into entities - and vice-versa - appropriately in the course of the expression traversal. The end-result type of a query expression is determined by the highest level `instance` type of the expression: literal if the instance type is literal, entity otherwise.

- In addition to the forms mentioned above, `instance` can also take the following form:

`%[ ??? ]`	a combination of question mark (`?`) and/or dot (`.`) characters, enclosed in brackets, preceded by the percent (`%`) character.

  This indicates that the expression results should be resp. the `source`, `medium` and/or `target` component of another entity. As an example, the query expression

  ```
  .: ~%[ ??? ]
  ```

  will return the list of entities which are neither `source` nor `medium` nor `target` of any other entity.

- In addition to the forms mentioned above, Consensus allows to use the back-pointer notation

  ```
  target<-medium-source
  ```

  as an alternative to the fore-pointer notation

  ```
  source-medium->target
  ```

  and to mix fore- and back- pointer notations within the same expression.

  In the context of a Consensus output operation, Consensus will format the output according to the notation used in the query expression.

Appendix A. 2. Consensus shorties

Consensus allows to use a simplified representation, called ***shorty***, for the body of expressions where at least two of the terms `source`, `medium` and/or `target` are made of the dot (`.`) character, representing ***any***, without prefix.

The complete list of all ***shorties*** is presented below, together with their equivalent expanded form:

`source..`	equivalent to the expanded form `source-.->.`
`.medium.`	equivalent to the expanded form `.-medium->.`
`..target`	equivalent to the expanded form `.-.->target`
`target<-..`	equivalent to the expanded form `target<-.-.`

where

`source`, `medium` and `target` can be either

`.`	the dot (`.`) character, representing ***any***.
`?`	the question mark (`?`) character, without any prefix, and with the restriction presented in ***Appendix A. 1.*** concerning the usage of `mark` in the overall expression
`identifier`	an identifier (name) or group of identifiers enclosed in double-quotes (").
`%variable`	a reference to the identified `variable` value, which can be either an expression or a list of entities or literals.
`[ expression ]`	where `expression` can take any query expression form, with the restriction on `mark`.

`source`, `medium` and `target`, when not made of the dot (`.`) or the question mark (`?`) character, can be, in that order

prefixed by the tilde (~) character, to represent ***not***

prefixed by either the star (*) or the underline (_) character, to represent respectively ***active*** and ***inactive***.

- The rest of the expression can take any of the forms presented in ***Appendix A. 1.***

As an example, the query expression

`?.. : instance`	will return the source component of `instance`
`.?. : instance`	will return the medium component of `instance`
`..? : instance`	will return the target component of `instance`

~[...]	will return all the entities which have neither source nor medium nor target component, that is: all base entities in the database.