G2 Language Reference Card

Gensym Corporation

Version Gensym.2015.16, May 2017

Table of Contents

G2 Language Reference Card, Version Gensym.2015.16

Gensym Corporation
401 Congress Avenue
Suite 2650
Austin, Texas USA 78701
Telephone: (781) 265-7100

G2 Language Reference

Syntax Notation

Syntax element	Description
`keyword`	Keyword(s) of the G2 Language
italic-term	User supplied term
{*choice*$_1$ | *choice*$_2$ | … | *choice*$_n$}	Choose one
[,…] [; …] …	Repeat 0 or more times
{}	Group elements
[]	Optional syntax
[]	Literal brackets

Actions

abort

`abort` {*procedure* | *method* | *procedure-invocation* | *method-invocation*}

activate/deactivate

{`activate` | `deactivate`} `the subworkspace of` *item*

change

`change` {*g2-array* | *g2-list*} **[** *integer-expression* **]** = *item-or-value*

`change` {*g2-array* | *g2-list*} **[** *integer-expression* **]** `to have no value`

`change the array-length of` *g2-array* `to` *quantity-expression*

`change the` *color-attribute-name* `of` *item* `to` {*color-name* | *symbolic-expression*}

`change the color-pattern of` *item* `so that` {*color-attribute-name* `is` *color-name*} [, …]

`change the` *region-name* `icon-color of` *object* `to` *color-name*

`change the name of` *item* `to` *symbolic-expression*

`change the size of` *kb-workspace* `to minimum`

`change the text of the` *attribute* `of` *item* `to` *text-expression*

`change the text of` {*procedure* | *statement* | *free-text* | *message*} `to` *text-expression*

conclude

`conclude that` {*g2-array* | *g2-list*} **[** *integer-expression* **]**
{ = *value-expression* | is *symbolic-expression* | `has no value`}

`conclude that the` *attribute* `of` *item* {= *value-expression* |
`is` *symbolic-expression* | `has no value`}

`conclude that the` { *class-name* | *type*} `that is an attribute of` *item*
`named by` *symbolic-expression* {= *value-expression* | `is` *symbolic-expression*}

`conclude that the` *icon-variables* `of` *item* = *value-expression*

`conclude that` {*variable* | *parameter*} = *value-expression*

`conclude that` {*symbolic-variable* | *symbolic-parameter*} `is` *symbol*

`conclude that` {*logical-variable* | *logical-parameter*} `is` {`true` | `false`}

`conclude that not` {*logical-variable* | *logical-parameter*}

`conclude that` *variable* `has no` [`current`] `value`

`conclude that` *item* `is` [{`not` | `now`}] *relation-name item*

create

`create` {`a` | `an`} *class-name* [*local-name*] [`by cloning` *item*]

`create an instance` [*local-name*] `of the class named by` *symbolic-expression*

`create a connection` [*local-name*] [`of` {`class` *connection-class-name* |
`the class named by` *symbolic-expression*}] `connected` {*between-spec* | *to-spec*}

delete

`delete` {*item* | *connection* [`removing connection stubs`] }

focus

`focus on` {*object* | *object-class*}, `awaiting completion`

halt

halt [with *text-expression*] if breakpoints are enabled

hide

hide {the workspace of *item* | the item superior to *item* |
the subworkspace of *item* | this workspace} [on *g2-window*]

inform

inform {the operator | *item*} [on *kb-workspace* [below *item* | *above item*]]
[for the next *time-interval*] that *text-expression*

insert

insert *item-or-value* {at the {beginning | end} of } | { {before | after} *item-or-value* in} *g2-list*

insert *item-or-value* {before | after} element *integer-expression* of *g2-list*

invoke

invoke *rule-category-name* [{, | or} *rule-category-name* rules]
[for {*object* | *object-class*}], awaiting completion

make

make *item* [permanent | transient]

make *kb-workspace* the subworkspace of *item*

move

move *item* [to | by] (*x*, *y*)

pause

pause knowledge-base

post

post [for the next *time-interval*] *text-expression*

print

print *kb-workspace*

remove

remove {*item-or-value* | the {first | last} *type*} from *g2-list*

remove element *integer-expression* from *g2-list*

reset

reset knowledge-base

rotate

rotate *item* {by | to the heading} *quantity-expression* degrees

set

set *variable* to *value-expression*

show

show {*kb-workspace* | *item*}
[{on window at {*scale* | (*x*, *y*) in the screen} |
{at the *item-location* of the screen} |
{scaled by *numeric-expression*} |
{scaled by its current scale times
{*quantity-expression* | (*x*, *y*) } } |
{with its *item-location* at the *window-location* of the screen} |
{with focal point (*x*, *y*) at (*x*, *y*) in the screen} |
{with its *workspace-location* at the *window-location* of the screen}]

show the item superior to {*kb-workspace* | *item*}

show the {workspace | subworkspace} of {*kb-workspace* | *item*}

shut down G2

shut down g2

start

start *procedure* ([*argument* [, ...]]) [across *g2-to-g2-interface* | *gsi-interface*]
[at priority *integer-expression*] [after *time-expression*])

transfer

transfer *item* {to *kb-workspace* [at (*x*, *y*)] | off}

`transfer` *item* `to the mouse of` *g2-window*

`transfer` {`the` *attribute* `of` *object* | *object*} `to` {*item* | `the` *attribute* `of` *object*}

update

`update` *item*

Procedure Syntax

The term *statement* in the syntax that follows refers to the statements under Procedure Statements.

procedure-name ([*argument*:{ {`class` *class-name*} | *type*}] [, ...])
[= ({`class` *class-name*} | *type*} [, ...])]

[*local-name* [, ...]:{`class` *class-name* [= *item*] } | *type* [= *value-expression*] } ;] ...

`begin`

[*statement-label*:] *statement* [; ...]

`end`

[`on error` (*local-name*) [*statement* [; ...]] `end`]

Procedure Statements

Allow Other Processing

`allow other processing`

Assignment

local-name = *value-expression*

Begin ... End

`begin`
[*statement-label*:] *statement* [; ...]
`end`

Call

[local-name [, ...] =] `call` *procedure* (*item-or-value* [, ...])
[`across` {*g2-to-g2-interface* | *gsi-interface*}]

Case

```
case (value-expression) of
 { {quantity | text | symbol}: statement [; ...] } ...
 [otherwise: statement] [; ...]
end
```

Collect Data

```
collect data [ (timing out after time-expression) ]
  local-name = value-expression [; ...]
 [; if timeout then statement] [;]
end
```

Do in Parallel

```
do in parallel [until one completes]
  statement [; ...]
end
```

Exit

```
exit if truth-value-expression
```

For

```
for local-name = each generic-reference-expression do [in parallel [until one completes] ]
statement [; ...]
end
```

```
for local-name = quantity-expression {to | down to} quantity-expression [by increment]
do [in parallel [until one completes] ]
statement [; ...]
end
```

Go To

```
go to statement-label
```

If-Then

```
if truth-value-expression then statement [else statement]
```

On Error

This statement can follow the `end` keyword of any kind of block.

`on error` (*local-name*) [*statement* [; ...]] `end`

Repeat

`repeat` *statement* [; ...] `end`

Return

`return` [*value-expression* [, ...]]

Signal

`signal` *error-object*

Wait

`wait` { {`for` *time-expression*} |
 {`until` *truth-value-expression* `checking every` *time-expression*} |
 {`until` {*variable* | *parameter* |
 {`the` *attribute* [*local-name*] `of` *item*} |
 {`the` {*class-name* | *type*} `that is an attribute of` *item*
  [`named by` *symbolic-expression*] } receives a value}}}

Rules Syntax

For an explanation of *generic-reference-expression*, see Generic References.

for prefix

for-prefix := {`for` {`any` | `the`} *generic-reference-expression*} ...

if rule

[*for-prefix*] `if` *truth-value-expression* `then` [`in order`] *action* [`and` *action*] ...

initially rule

`initially` [*for-prefix* [`unconditionally`]]
 [`if` *truth-value-expression* `then`]
 [`in order`] *action* [`and` *action*] ...

unconditionally rule

[*for-prefix*] `unconditionally` [`in order`] *action* [`and` *action*] ...

when rule

[*for-prefix*] `when` *truth-value-expression* `then` [`in order`] *action* [`and` *action*] ...

whenever rule

[*for-prefix*] `whenever` *event-expression* [`or` *event-expression*]
[`and when` *truth-value-expression*] `then`
[`in order`] *action* [`and` *action*] ...

Generic References

Precede a generic reference expression with one of the following five quantifiers, as appropriate.

the *generic-reference-expression*

In any expression, specifies a unique value.

any *generic-reference-expression*

In the antecedent of a rule, specifies a generic reference.

every *generic-reference-expression*

In the consequent of a rule, causes an action to execute on every item matching the reference; in a for every expression, iterates over a set.

each *generic-reference-expression*

In expressions over sets, computes a value based on the set; in procedures, iterates over items in a class.

a(n) *generic-reference-expression*

In there exists expressions, specifies a single element to test existence.

generic reference expression

generic-reference-expression :=
{*class-name* | *type*} [*local-name*] [*generic-reference-qualifier*]

generic-reference-qualifier :=
{upon *kb-workspace*} | {connected *connected-expression*} |
{at *at-expression*} | {nearest to *item*} |
{superior to *kb-workspace* | *object-attribute*} |
{that is *relation-name item*} |
{named by *symbolic-expression*} |
{in {*g2-list* | *g2-array*} } | {name of *item*} |
{that is {a superior-class | an inferior-class} of *symbolic-expression*}

Expressions

Generic reference qualifier expressions are indicated with a *. Use the, any, every, each, or a(n), as appropriate.

Attribute

the *attribute-name* [*local-name*] of {*item* | *structure*}
the text [*local-name*] of *item*
the text of the *attribute* of *item*
the attribute name of *subobject-expression*

the {*variable* | *parameter*} giving the *attribute-name* of
{*object* | *connection* | *message*}

*the {*class-name* | *type*} that is an attribute of {*item* | *structure*}
[named by symbolic-expression]

*the symbol [*local-name*] that is {a | an}
user-defined attribute name of {*item* | *structure*}

*the symbol [*local-name*] that is an attribute name of {*item* | *structure*}

Class

the class of *item*

the symbol [*local-name*] that is
{a superior-class | an inferior-class} of *symbolic-expression*

any instance of *class* is created

Connection

the *class-name* [*local-name*] connected to item

the *class-name* [*local-name*] connected at

an {input to | input of | output of} *item*

the *class-name* [*local-name*] connected at the
{*port-name* of | input to | input of | output of} *item*

the *class-name* [*local-name*] at {an input end | an output end |
either end} of *connection*

item is {connected to | disconnected from} *item*

connection {of class *class*] is directly
{connected to | disconnected from}
{an input of | an output of | the portname of} *item*

Item

Name

item-name

the *class-name* [*local-name*] named by *symbolic-expression*

the *class-name* [*local-name*] superior to *object*

the name [local-name] of item

Module Assignment

the containing-module of *item*

Scope

{the item | this procedure | this procedure-invocation | this rule |
this window | this workspace}

Status

{*item* | *subworkspace*} is {activated | deactivated}

item is {enabled | disabled}

Truth Value

item is [not] the same object as *item*

item has a name

item is {a | an} *class-name*

`item is an instance of the class named by` (*symbolic-expression*)

Representation

`the` {`item-height` | `item-width`} `of` *item*

`the icon-heading of` *item*

Location

`the distance between` *item* `and` {*item* | `the nearest` *class-name*}

`the` *class-name* [*local-name*] `nearest to` *item*

object `is moved` [`by` {`the user` | `G2`}]

`the` {`item-x-position` | `item-y-position`} `of` *item*

List and Array

List and Array

{*g2-array* | *g2-list* | *sequence* | *text*} **[** *integer-expression* **]**

`the` {*class-name* | *type*} [*local-name*] `in` {*g2-array* | *g2-list* | *sequence*}

List

`the` {`first` | `second` | `next to last` | `last`} {*class-name* | *type*}
[*local-name*] `in` {*g2-list* | *sequence*}

{*class-name* | *type*} `is` [`not`] `a member of` {*g2-list* | *sequence*}

`the number of elements in` {*g2-list* | *sequence*}

g2-list `that contains` *item*

Array

`the array-length of` *g2-array*

Numeric

Arithmetic Operations

quantity-expression { `+` | `-` | `*` | `/` } *quantity-expression*

quantity-expression `^` integer-expression

Comparison

quantity-expression { = | /= | > | >= | < | ⇐ } *quantity-expression*

Numeric Expressions Over Sets

the {sum | product | average | minimum | maximum} over each
generic-reference-expression of (*quantity-expression*)

the count of each *generic-reference-expression*
[such that (*truth-value-expression*)]

Fuzzy Truth

truth-value-expression {is more true than | is less true than |
is not more true than | is not less true than | = | /=}
truth-value-expression

quantity-expression {< | ⇐ | > | >= | =} *quantity-expression*
(`+-`*quantity-expression*)

Relation

the *class-name* [*local-name*] that is *relation-name item*

item is [not] *relation-name item*

the relationships of *item*

the items-in-this-relation of *relation-name*

item {becomes | ceases to be} *relation-name item*

item {becomes | ceases to be} related to *item*

Symbol

the symbol *symbol*

symbolic-expression {is | is not} *symbol*

Text

" [*text*] **[** *value-expression* **]** [*text*] "

text-expression { = | /= | > | >= | < | ⇐} *text-expression*

Formatting

quantity-expression [as {*ddd.dddd-format* | a time stamp | an interval}]

Time

the current [subsecond] [real] time

the current {year | month | day of the month | day of the week |
hour | minute | second}

the current system [real] time

Truth Value

Logical Operation

truth-value-expression {and | or} *truth-value-expression*

not *truth-value-expression*

Conditional

(if *truth-value-expression* then *item-or-value*
[else *item-or-value*])

For Every

for every *generic-reference-expression* (*truth-value-expression*)

Existence

there exists {a | an} *generic-reference-expression*
[such that (*truth-value-expression*)]

item-or-value {exists | does not exist}

Values

value-expression is a type

the [current] value of (*value-expression*)

value-expression has {a | no} [current] value

the first of the following expressions that has a [current] value
(*value-expression* [, ...])

Variable and Parameter

Variables

variable `loses its value`

Variables and Parameters

`the simulated value of` {*variable* | *parameter*}

`the value of` {*variable* | *parameter*} {`as of` *time-expression* `ago` |
`as of` *integer-expression* `datapoints ago`}

History (*quantity***,** *integer***,** *float variables* **and** *parameters* **only)**

`the` {`average value of` | `maximum value of` | `minimum value of` |
`sum of the values of` | `standard deviation of` | `number of history datapoints in`}
{*variable* | *parameter*} {`during the last` time-expression |
`between` *time-expression* `ago and` *time-expression* ago `}

`the` {`rate of change per` | `integral in`} *time-unit* `of` {*variable* | *parameter*}
{`during the last` *time-expression* | `between` *time-expression* `ago and` *time-expression* `ago`}

`the interpolated value of` {*variable* | *parameter*} `as of` *time-expression* `ago`

`the` {`history` | `history-using-unix-time`} `of` {*variable* | *parameter*}

`history-expression of` *variable-or-parameter* `during the` *time-expression*
`ending with the collection time`

Time

`the collection time of` {*variable* | *parameter*}
[`as of` *integer-expression* `datapoints ago`]

`the expiration time of` *variable*

Workspace

`the` {`workspace` | `subworkspace`} [*local-name*] `of` *item*

kb-workspace `has` [`not`] `been activated`

`the` *class-name* [*local-name*] `superior to` *kb-workspace*

`the` *class-name* [*local-name*] `upon` *kb-workspace*

`the module-assignment of` *kb-workspace*

Functions

Arithmetic Functions

`abs` (*quantity-expression*)

`arctan` (*quantity-expression* [, *quantity-expression*])

`average` (*quantity-expression*, *quantity-expression* [, *quantity-expression*...])

`ceiling` (*quantity-expression*)

`cos` (*quantity-expression*)

`exp` (*quantity-expression*)

`expt` (*quantity-expression*, *quantity-expression*)

`floor` (*quantity-expression*)

`ln` (*quantity-expression*)

`log` (*quantity-expression*)

`max` (*quantity-expression*, *quantity-expression* [, ...])

`min` (*quantity-expression*, *quantity-expression* [, ...])

`quotient` (*quantity-expression*, *quantity-expression*)

`random` (*quantity-expression* [, *quantity-expression*])

`remainder` (*quantity-expression*, *quantity-expression*)

`round` (*quantity-expression*)

`sin` (*quantity-expression*)

`sqrt` (*quantity-expression*)

`tan` (*quantity-expression*)

`truncate` (*quantity-expression*)

`truth-value` (*quantity-expression*)

Bitwise Functions

`bitwise-or` (*value-expression*, *value-expression*)

`bitwise-and` (*value-expression*, *value-expression*)

`bitwise-xor` (*value-expression*, *value-expression*)

`bitwise-not` (*value-expression*)

`bitwise-right-shift` (*value-expression*, *value-expression*)

`bitwise-left-shift` (*value-expression*, *value-expression*)

`bitwise-test` (*value-expression*, *value-expression*)

`bitwise-set` (*value-expression*, *value-expression*)

Call

`call-function` (*function-definition-expression* [*argument*] [, ...])

Character Support Functions

`character-codes-to-text` (*character-codes*: `sequence`)

`compare-text` (*text-1*: `text`, *text-2*: `text`)

`export-text` (*unicode-text*: `text`, *conversion-style*: `class text-conversion-style`)

`import-text` (*external-text*: `text`, *conversion-style*: `class text-conversion-style`)

`is-digit` (*character-code*: `integer`)

`is-lowercase` (*character-code*: `integer`)

`is-readable-digit` (*character-code*: `integer`)

`is-readable-digit-in-radix` (*character-code*: `integer`, *radix*: `integer`)

`is-titlecase` (*character-code*: `integer`)

`is-uppercase` (*character-code*: `integer`)

`readable-symbol-text` (*printed-text*: `text`)

`readable-text` (*printable-text*: `text`)

`readable-text-for-value` (*value-for-text*: `value`)

`text-to-character-codes` (*input-text*: `text`)

`to-lowercase` (*character-code*: `integer`)

`to-titlecase` (*character-code*: `integer`)

`to-uppercase` (*character-code*: `integer`)

`transform-text-for-unicode-comparison` (*text-to-transform*: `text`, *consider-case*: `truth-value`)

`transform-text-for-G2-4.0-comparison` (*text-to-transform*: `text`, *consider-case*: `truth-value`)

Connection Evaluator Functions

`connection-direction` (*object*, *connection*)

`connection-portname` (*object*, *connection*)

`connection-position` (*object*, *connection*)

`connection-side` (*object*, *connection*)

`connection-style` (*connection*)

`items-are-connected` (*item1*: `class item`, *item2*: `class item`)

`items-are-connected-with-direction` (*item1*: `class item`, *item2*: `class item`, *item1-direction*: `symbol`)

`items-are-connected-at-ports` (*item1*: `class item`, *item2*: `class item`, *portname1*: `symbol`, *portname2*: `symbol`)

Format Numeric Text Function

`format-numeric-text` (*quantity-text*: `text`, *formatting-expression*: `text`)

Great Circle Distance Function

`great-circle-distance` (*latitude-1*: `quantity`, *longitude-1*: `quantity`, *latitude-2*: `quantity`, *longitude-2*: `quantity`, *radius*: `quantity`)

Quantity Function

`quantity` ({*truth-value-expression* | *text-expression*})

RGB Symbol Function

`rgb-symbol` (*rr*, *gg*, *bb*)

Regular Expressions Text Functions

`find-next-pattern` (*search-pattern*: `text`, *source-text*: `text`, *start-position*: `integer`)

`find-next-substring-matching-pattern` (*search-pattern*: `text`,
source-text: `text`, *start-position*: `integer`)

`find-and-replace-pattern` (*search-pattern*: `text`,
text-to-substitute: `text`, *source-text*: `text`,
start-position: `integer`, *end-position*: `integer`)

Sequence Functions

`change-element` (*sequence*, *integer*, *item-or-value*)

`concatenate` (*sequence*, *sequence* [,...])

`insert-at-beginning` (*sequence*, *item-or-value*)

`insert-at-end` (*sequence*, *item-or-value*)

`insert-after` (*sequence*, *item-or-value*, *item-or-value*)

`insert-before-element` (*sequence*, *integer*, *item-or-value*)

`insert-after-element` (*sequence*, *integer*, *item-or-value*)

`remove` (*sequence*, *integer*)

`portion` (*sequence*, *integer*, *integer*)

`sequence` (*item-or-value* [,...])

Symbol Function

`symbol` (*text-expression*)

Structure Functions

`change-attribute` (*structure*, *attribute-name*, *item-or-value*)

`change-evaluated-attribute` (*structure*, *symbol-expression*, *item-or-value*)

`evaluated-structure` (*symbol-expression*, *item-or-value* [,...])

`remove-attribute` (*structure*, *attribute-name*)

`remove-evaluated-attribute` (*structure*, *symbol-expression*)

`structure` (*attribute-name*: *item-or-value* [,...])

Text Functions

`capitalize-words` (*text-expression*)

`get-from-text` (*source-text*: `text`, *start-index*: `integer`, *end-index*: `integer`)

`insert-in-text` (*text-expression*, *text-expression*, *integer*)

`is-contained-in-text` (*text-expression*, *text-expression*)

`length-of-text` (*text-expression*)

`lower-case-text` (*text-expression*)

`omit-from-text` (*text-expression*, *integer*, *integer*)

`position-of-text` (*text-expression*, *text-expression*)

`replace-in-text` (*text-to-substitute*: `text`, *source-text*: `text`,
start-index: `integer`, *end-index*: `integer`)

`text-begins-with-quantity` (*text-expression*)

`upper-case-text` (*text-expression*)

Text Tokenizing Function

`get-next-token` (*tokenizer*: `class G2-tokenizer`, *source-text*: `text`, *start-position*: `integer`)

Time Functions

{`year` | `month` | `day-of-the-month` | `day-of-the-week` | `minute` | `hour` | `second`} (*time-expression*)

`time` (*year*, *month*, *day*, *hour*, *minute*, *second*)

User-Defined Functions

function-name ([*argument*] [, ...])

Inspect Syntax

show on a workspace {*item* | the *class-name* named *item-name*} |
every *class-name* [*filter*] | the workspace hierarchy
[of {*kb-workspace-name* | *object-name*}] |
the class hierarchy [of *class-name*] |
the module hierarchy [of *module-name*] |
the method hierarchy of *method-name* |
the method inheritance path for *class-name*
[and the method *method-name*] }

write to the file *filename* {*item* | the *class-name named item-name*} |
every *class-name* [*filter*] | the class hierarchy [of *class-name*] }

go to *symbol*

display a table [columnwise] of [the *attribute-name* [, ...] of]
{*item* | the *class-name* named *item-name* |
every *class-name* [*filter*] }

replace | highlight [the {word | symbol}] {*text* | *symbol*} with
{*text* | *symbol*} in {*item* | the *class-name* named *item-name* |
every *class-name* [*filter*] }

check for consistent modularization

recompile {*item* | the *class-name* named *item-name* |
every *class-name* [*filter*] }

Filters

such that *truth-value-expression*

containing [the {word | symbol}] *text-expression*

with notes

whose status is *status*

where *attribute-name relational-operator quantity-expression*

where *attribute* is {*truth-value-expression* | *symbolic-expression*}

in the category *symbolic-expression*

which has the focal {class | object} *symbolic-expression*

found on the workspace *kb-workspace*

`assigned to module` *module-name* [`or` *module-name*]

`assigned to the hierarchy of module` *module-name*

Version Control

`show on a workspace the change log entry of the` *attribute* `of` *item*
{`as of` *timestamp*} | {`with revision` *num*} | {`with tag` *tag*}

`show on a workspace the differences between`
`the change log entry of the` *attribute* `of` *item*
{`as of` *timestamp*} | {`with revision` *num*} | {`with tag` *tag*}
`and the change log entry of the` *attribute* `of` *item*
{`as of` *timestamp*} | {`with revision` *num*} | {`with tag` *tag*}

`use version control to tag the change log entry of every logged attribute of`
`every item in module` *module-name* [`as of` *timestamp*] `using tag` *tag*

`use version control to tag the change log entry of the` *attribute* `of` *item*
[`as of` *timestamp* | `with revision` *num*] `using tag` *tag*

`use version control to revert the text of every logged attribute of every`
`item in module` *module-name* `to the change log entry` {`as of` *timestamp*}
| {`using tag` *tag*}

`use version control to revert the text of the` *attribute* `of` *item* `to the`
`change log entry` {`as of` *timestamp*} | {`with revision` *num*} |
{`with tag` *tag*}

`use version control to delete the change log entry of the` *attribute* `of` *item*
{`as of` *timestamp*} | {`with revision` *num*} | {`with tag` *tag*}

`use version control to enable change logging on` *item*

`use version control to disable change logging on` *item*