



Atom-Based Software Engineering

ABSE Reference

ABSE Reference

The ABSE Reference help will describe in detail the ABSE command library (API).

Framework Commands and Functions

ABSE is supported by a framework of commands and functions, that acts just like an API (Application Programming Interface), implemented in the [Lua](#) programming language.

[Atom Templates](#) are composed of 7 independent [Sections](#) containing Lua code fragments. In each of these sections you can use Lua's own features, under certain restrictions, combined with ABSE's predefined commands and functions.

An ABSE command is understood as being a function call that returns no values.
An ABSE function is understood as being a function call that returns one or more values.

See the [Alphabetical ABSE Command & Function](#) list.

Admin Section Commands

The commands listed here are exclusively used on the Admin Section of Atom Templates.

Presentation commands:

[icon\(\)](#)

[label\(\)](#)

[help\(\)](#)

Constraint commands:

[neutral\(\)](#)

[under\(\)](#)

Organization commands:

[dict\(\)](#)

Parameter commands:

[param_area\(\)](#)

[param_block\(\)](#)

[param_bool\(\)](#)

[param_link\(\)](#)

[param_list\(\)](#)

[param_text\(\)](#)

Classification commands:

[abstract\(\)](#)

[doc\(\)](#)

Form Section Commands

The commands listed here are exclusively used on the Form Section of Atom Templates. The Form Section defines the structure and behavior of a specialized editor for an Atom Template's instances.

Form structure commands

[param_group\(\)](#)
[hide_param\(\)](#)

Form contents commands

[setdef\(\)](#)

Create Section Commands

The commands listed here are exclusively used on the Create Section of Atom Templates. The Create Section is executed when an Atom Instance is placed on a project tree, and can be compared to the constructor of an object in the object-oriented programming paradigm.

Organization commands

[list\(\)](#)

Library include commands

[include_lib\(\)](#)
[include_all_libs\(\)](#)

Preparation and Generation Commands

The commands listed here are used on the Create, Pre, Exec, Post and Functions Sections of Atom Templates. They are used to prepare or perform code generation in the context of their Atom Instances.

Counters

These commands can be used manage and obtain sequential unique numbers.

[counter_new\(\)](#)
[counter_get\(\)](#)

Virtual Functions

The following commands let you define [Atom Behaviors](#) (Atom Virtual Functions). An Atom Behavior is a function that conforms to a global specification, invoked through a unique string, allowing Atoms to work with other Atom types that were unknown to them at design time.

[vfunc\(\)](#)
[vfunc_exec\(\)](#)

Variables

The following commands and functions will act on Atom Instance variables.

Getting variables

[var\(\)](#)

[param\(\)](#)
[pvar\(\)](#)
[dvar\(\)](#)
[cvar\(\)](#)

Setting variables

[set\(\)](#)
[set_global\(\)](#)

Querying variables

[is_empty\(\)](#)
[var_exists\(\)](#)

Utility functions and commands

[eval\(\)](#)
[find_var\(\)](#)
[unset\(\)](#)

Atom Processing

These functions and commands will act on Atom Instances.

Querying Atoms

[is_type\(\)](#)
[is_under\(\)](#)
[is_child_of\(\)](#)
[is_neutral\(\)](#)
[is_relocated\(\)](#)
[get_type\(\)](#)

Getting Atoms

[get_atom\(\)](#)
[find_by_var\(\)](#)
[oparent\(\)](#)
[parent\(\)](#)

Child Atoms

[child\(\)](#)
[list_children\(\)](#)
[find_child\(\)](#)
[child_count\(\)](#)

Utility functions

[forall\(\)](#)
[exec\(\)](#)

Generation Commands

The commands listed here are exclusively used on the Exec and Post Sections of Atom Templates. They are used to perform code generation in the context of their Atom Instances.

Queries

[get_current_file\(\)](#)

Generation Cursor[cursor\(\)](#)**File Organization**[section\(\)](#)[unique_section\(\)](#)[common_section\(\)](#)**Text Generation**[line\(\)](#)[code\(\)](#)[block\(\)](#)**General Commands**

The commands listed here can be used in any of the Atom Instance sections (Form, Create, Pre, Exec, Post, Functions):

Environment-related commands and functions[log\(\)](#)**Atom-related commands and functions**[get_id\(\)](#)[get_atom_id\(\)](#)[tree_root\(\)](#)**Alphabetical ABSE Command & Function Quick Reference**

[abstract](#) - Specifies that the Atom Template cannot be directly instantiated.

[block](#) - Insert the given contents at the Generation Cursor's current position, using a mask.

[child](#) - Return the first direct child Atom from a specified type.

[child_count](#) - Return the number of direct child Atoms from a specified type.

[code](#) - Insert a block of code at the Generation Cursor's current position.

[common_section](#) - Insert a common-type file section at the Generation Cursor's current position.

[counter_get](#) - Get the current value of a counter and increment/decrement automatically.

[counter_new](#) - Create a counter that can then be used to obtain unique integers.

[cursor](#) - Place or move the [Generation Cursor](#).

[cvar](#) - Return the contents of a variable from a direct child Instance.

[dict](#) - Define a dictionary entry for every instance of an Atom Template.

[doc](#) - Specifies that the Template's Instances are for documentation purposes only.

[dvar](#) - Return the contents of a variable directly from a specified Instance.

[eval](#) - Translate a string into another by evaluating embedded variables.

[exec](#) - Calls a user-defined function in the context of a given Atom.

[find_by_var](#) - Return the first found child from a specified type that also has a variable whose

contents match those supplied.

[find_child](#) - Return the first found child from a specified type under the given parent.

[find_var](#) - Search for an Atom Instance given a variable that is defined on that Atom.

[forall](#) - Iterates on child Atom Instances and returns the resulting text, according to a specified format.

[get_atom](#) - Return a reference to an Atom, given its ID.

[get_atom_id](#) - Returns the ID of a given Atom Instance.

[get_current_file](#) - Return the name of the file where the Generation Cursor is currently placed.

[get_id](#) - Return the ID of the Atom Instance being executed.

[get_type](#) - Return an Atom Instance's type (the name of its Atom Template).

[help](#) - Define help text to describe the use of an Atom Template.

[hide_param](#) - Hides a parameter, removing it from the Instance's form (but not from the Atom Template).

[icon](#) - Set the icon to display on all Instances of a Template.

[include_all_libs](#) - Makes an Atom Instance the insertion point of Auto-Include and Optional-Include Atoms from all Atom Libraries.

[include_lib](#) - Makes an Atom Instance the insertion point of Auto-Include and Optional-Include Atoms from a given Atom Library.

[is_child_of](#) - Check if an Atom Instance is under another, specified Instance.

[is_empty](#) - Check if the specified variable is empty.

[is_neutral](#) - Check if the supplied Atom Instance is neutral.

[is_relocated](#) - Check if the supplied Atom Instance is relocated.

[is_type](#) - Check the type of a given Atom Instance.

[is_under](#) - Check if the supplied Atom Instance is on a branch headed by an Instance of the given type.

[label](#) - Sets the formulas to use when computing the Atom Instance's label and description.

[line](#) - Insert a single line of code at the Generation Cursor's current position.

[list](#) - Classifies Atom Instances by adding them to an Atom List.

[list_children](#) - Return a list of Atom Instances that conform to the specified criteria.

[log](#) - Append a message to the specified log.

[neutral](#) - Specifies that the atoms derived from this Atom Template are invisible to several ABSE rules.

[oparent](#) - Returns the original parent of the supplied Atom.

[param](#) - Returns the contents of a Parameter-bound variable.

[param_area](#) - Define a code area for an Atom Template.

[param_block](#) - Define a text block (multi-line text) parameter for an Atom Template.

[param_bool](#) - Define a boolean parameter for an Atom Template.

[param_group](#) - Groups together several Atom Template parameters that have been tagged with the same group ID.

[param_link](#) - Define a link-type parameter for an Atom Template.

[param_list](#) - Define a list-selection parameter for an Atom Template.

[param_text](#) - Define a single-line text parameter for this Atom Template.

[parent](#) - Return the parent Atom of the current Atom Instance or of another Atom if specified.

[pvar](#) - Get the contents of a variable, but ignore local variables (start searching at the parent Atom).

[section](#) - Insert a file section at the [Generation Cursor](#)'s current position.

[set](#) - Create or change a variable on the current Atom Instance.

[set_global](#) - Create or change a variable on the Tree's own variable table.

[setdef](#) - Sets a default value to be presented on the control that corresponds to the given parameter on the Atom Instance form.

[tree_root](#) - Return the invisible Atom Instance that represents the root of the tree being executed.

[under](#) - Defines where the Instances of an Atom Template can be created as children.

[unique_section](#) - Insert a unique-type file section at the Generation Cursor's current position.

[unset](#) - Remove a temporary variable from the current Atom Instance.

[var](#) - Return the contents of a variable, following ABSE's variable seek rules.

[var_exists](#) - Check if a given variable exists in the variable seek path.

[vfunc](#) - Declare a function as the handler of a globally defined [Atom Behavior](#).

[vfunc_exec](#) - Invoke an Atom Behavior on a designated Atom Instance.

abstract

abstract()

Command category : Classification

Allowed in sections : Admin

Inherited from base class(es) : no

Specifies that the Atom Template cannot be directly instantiated, therefore, preventing the corresponding Atom Instances from being created on a project tree. You can compare this to virtual classes in an object-oriented programming language.

block

block(contents, mask)

contents : Text to insert.

mask : Insertion mask where "{}" denotes the place where text is to be placed on each line.

Command category : File

Allowed in sections : Exec, Post, Functions

This command will insert the given contents at the [Generation Cursor](#)'s current position. For each supplied line, the command will apply the given mask (template) string.

Example:

```
set("description", "This is the description\n of this object.")
block("description", "/* {} */")
```

The resulting generated text would be:

```
/* This is the description */
/* of this object */
```

child

atom = *child(child_type [,parent_atom])*

atom : Requested Atom Instance, or *nil* if no child of the given type was found.

child_type : Type of child to get (Atom Template name).

parent_atom : Atom Instance to use as parent. If *nil*, the current Atom Instance is used.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return the first direct child from the specified type under the given parent. If no parent was specified, the returned Atom Instance is a direct child of the current Atom.

child_count

count = *child_count([child_type [,atom]])*

count : The number of children that were found.

child_type : Type of child to find (Atom Template name). If empty, count all children.

atom : Atom Instance where the search starts. If *nil*, starts at the current Atom Instance.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return the number of direct children of the specified type that were found under the given atom. If no Atom is given, the current Atom Instance is considered.

code

code(contents [,callback_func])

contents : Line(s) to insert at the current generation cursor position.

callback_func : Name of a line-processing function to call (a function that accepts a single string argument and returns a string).

Command category : File

Allowed in sections : Exec, Post, Functions

This command will insert the given contents at the [Generation Cursor](#)'s current position. The difference from the [line\(\)](#) command is that for each supplied line, this command can call a specified function using that line as argument. This allows you to make custom processing on each generated line.

You can embed file sections using the following syntax:

```
{{regular_section}}
{u{unique_section}}
{c{common_section}}
```

Some escape sequences are translated:

`\n` : (Will split current line in two)
`\t` : Translated to a TAB character
`{\{` : Translated to `{{` to avoid inserting a file section
`\\` : Translated to `\`

common_section

common_section(name)

name : Name of section to insert

Command category : File

Allowed in sections : Exec, Post, Functions

This command will insert a [file section](#) at the [Generation Cursor](#)'s current position. This section will have a special behavior: All lines inserted in this section are automatically duplicated in all other common sections with the same name, in the same file.

Existence of other sections with the same name, but different behavior, is not verified. If there is a section with the same name, but with a different behavior, no text is duplicated into it.

This command can be embedded in source code through the

```
{c{section_name}}
```

construct, which should be the only text in that line.

counter_get

value = counter_get(name)

value : A positive integer returned by the specified counter

name : String that contains the name of the counter

Command category : General

Allowed in sections : Form, Create, Pre, Exec, Post

This function will get the current value of the specified counter. The counter is automatically

incremented.

Example:

```
local next_control_id = counter_get("control_ids")
```

This code will request a new unique ID to assign to a GUI control.

counter_new

```
counter_new(name [,initial_value [,limit [,step]]])
```

name : String that contains the name of the counter to create

initial_value : Starting value for this counter. Default is 1.

limit : Counter limit. AtomWeaver will stop with an error when this limit is crossed. The specified limit is still a valid value. If zero, no limit is set. Default is zero.

step : Incremental step. Default is 1.

Command category : General

Allowed in sections : Create, Pre, Exec, Post

This command will create a branch-accessible counter that can then be used to obtain unique, sequential or stepped positive integer numbers. Numbers are unique in the context of a single counter, as any counter can use any positive integer. If you want to obtain unique numbers among all counters, you'll have to set different limits on each one.

You can use the [counter_get\(\)](#) command to obtain a new, unique integer from the counter. The counter is automatically incremented or decremented. Like variables, the counter is only accessible to those Atoms that are on the branch headed by the counter's creator Atom.

Examples:

```
counter_new("control_ids")
```

Create a new counter named *control_ids* that will start at 1, with no upper limit, and supplying all integers above 1.

Repeated calls to *counter_get("control_ids")* will return 1, 2, 3, 4, 5, ...

```
counter_new("control_ids",10000)
```

Create a new counter named *control_ids* that will start at 10000, with no upper limit, and supplying all integers above 10000.

Repeated calls to *counter_get("control_ids")* will return 10000, 10001, 10002, 10003, ...

```
counter_new("control_ids",10000,10999)
```

Create a new counter named *control_ids* that will start at 10000, will stop at 10999, and supplying all integers in this interval.

Repeated calls to `counter_get("control_ids")` will return 10000, 10001, 10002, 10003, ...

```
counter_new("control_ids",10000,10999,10)
```

Create a new counter named `control_ids` that will start at 10000, will stop at 10999, and supplying integers in increments of 10.

Repeated calls to `counter_get("control_ids")` will return 10000, 10010, 10020, 10030, ...

CURSOR

```
cursor(section [,filename [,position]])
```

section : Name of the section where the generation cursor should move to.

filename : Path and name of the file where the generation cursor should move to. Default is the current file. Path separator is a slash ('/'). If empty, the current file is left unchanged.

position : Exact position in the section where the generation cursor should move to. Valid option strings are *begin*, *end* and *after*. Default is *end*.

Command category : File

Allowed in sections : Exec, Post, Functions

This command will place or move the [Generation Cursor](#).

If the specified section does not exist, an error is raised. Two special sections are pre-defined on each file: *begin* and *end*. If you specify *begin* as the section, subsequent generated lines will be placed starting at the beginning of the file. If you specify *end* as the section, subsequent generated lines will be placed at the end of the file. If the file is new, both *begin* and *end* will have the same effect: Lines will be appended.

If the specified filename does not exist, a new file is automatically created. Likewise, if the specified path does not exist, it will be created. Paths are always relative to the project's Output Folder.

The three possible positions on a section (*begin*, *end* and *after*) in a text file can be illustrated like this:

```
source line...
{{----section_begin_marker----}}
**begin**
source line...
source line...
**end**
{{----section_end_marker----}}
**after**
source line...
```

In the above fragment, you notice the two markers that make-up the section, surrounded by double curly brackets. These markers are internal and will not be visible in the generated file. As shown above through the corresponding keywords, you can place the Generation Cursor at the beginning or at the end of the section. In addition, you can place the Generation Cursor outside the section, immediately after the end marker. You cannot use the *after* keyword to place the Generation Cursor after a [common-type section](#).

Examples:

```
cursor("class_members")
```

Moves the generation cursor to the *class_members* section in the current file.

```
cursor("begin", "source/@headerfile")
```

Moves the cursor to the beginning of the file named through the *headerfile* variable, under the *source* folder. If the folder and/or file don't exist, they are automatically created.

```
cursor("message_ids", "definitions.h", "begin")
```

Moves the cursor to the beginning of the *message_ids* section on the "definitions.h" file.

cvar

```
var_contents = cvar(name)
```

var_contents : Contents of the requested variable. Can be a string or a pointer to an Atom Instance

name : Name of the variable to get

Command category : Variables

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This function will return the contents of an Atom Instance variable. But unlike the [var\(\)](#) function, this function will not search the current Atom Instance. Instead, it will try to get the variable from one of its direct Atom Instance child. If no variable is found at the Atom Instance's direct children, an error is raised. This command will **not** recurse the Tree in search of the variable.

Please note that if more than one child contains the requested variable, only the first of those children will be considered.

cvar stands for "child variable".

dict

```
dict(formula)
```

formula : String that contains a [formula](#) that can be translated into a unique string that can identify Instances of this Template.

Command category : Organization

Allowed in sections : Admin

Inherited from base class(es) : yes

Define a dictionary entry for every instance of this Atom Template.

Each Atom Instance that is created will automatically add an entry to the Atom Dictionary. All entries are unique, therefore a formula must be used to differentiate Instances. Otherwise a warning will be issued.

Example:

```
param_text("def", "class_name", "Class Name", "Name of this class")
dict("@class_name")
```

For each instance of *java_class* created, an entry to the Atom dictionary will be added, being the entry label the class' name.

doc

doc()

Command category : Classification

Allowed in sections : Admin

Inherited from base class(es) : yes

Specifies that instances of this type are for documentation purposes only, so they won't generate code.

AtomWeaver's generator will not process these Atoms, saving memory and processing time.

dvar

var_contents = dvar(atom, name)

var_contents : Contents of the requested variable. Can be a string or a pointer to an Atom Instance

atom : Atom to get the variable from

name : Name of the variable to get

Command category : Variables

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This function will return the contents of an Atom Instance variable. But unlike the [var\(\)](#) function, this function will not search the current Atom Instance. Instead, it will try to directly get the variable from the specified Atom Instance. If no variable is found at the specified Atom Instance, an error is raised. This command will **not** climb the Tree in search of the variable.

dvar stands for "direct variable".

eval

final_text = *eval(initial_text [,final_stage [,ignore_missing]])*

final_text : The final form of the *initial_text* after evaluation

initial_text : Text, with embedded variables, to be evaluated into its final form

final_stage : A boolean that tells if this text is to be used on a generated artifact (*true*) or to be further processed (*false*). Default is *true*

ignore_missing : A boolean that tells if missing variables are to be ignored (*true*) or if an error is raised (*false*). Default is *false*.

Command category : Variables

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This function will translate a string into another by evaluating embedded variables and replacing them by their current values. If *ignore_missing* is *true*, then missing variables will be silently ignored and will be kept on the final string.

exec

output = *exec(atom, func_name, args)*

output : Text as result of function execution

atom : Atom Instance that is to be associated with the called function. The Atom is made *current* before function is called.

func_name : Name of the Lua function to call.

args : A Lua table that contains useful input arguments for the invoked Atom Behavior, and optionally hold results.

Command category : Atoms

Allowed in sections : Create, Pre, Exec, Post, Functions

Calls a user-defined function, associating it with a given Atom. The function is executed in the [context](#) of the specified Atom and will have direct access to the Atom's variables.

find_by_var

child_atom = *find_by_var(atom, child_type, var_name, var_contents [,recursive [,silent]])*

child_atom : Requested Atom Instance, or *nil* if no child with the given variable contents was found.

atom : Parent Atom Instance where the search starts. If *nil*, starts at the current Atom Instance.

child_type : Type of child to find (Atom Template name).

var_name : String with the name of the variable to match.

var_contents : String with the contents to match.

recursive : If *true*, traverses the branch headed by the given Atom. Default is *false*.

silent : If *true* will simply return *nil* if a matching child was not found. Otherwise an error is raised.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return the first found child from the specified type under the given parent, that also has a variable whose contents match those supplied. If no parent was specified, the returned Atom Instance is a child of the current Atom. If the *recursive* option is used, the entire branch headed by *atom* is searched.

find_child

child_atom = *find_child(atom, child_type [,recursive])*

child_atom : Requested Atom Instance, or *nil* if no child of the given type was found.

atom : Atom Instance where the search starts. If *nil*, starts at the current Atom Instance.

child_type : Type of child to find (Atom Template name).

recursive : If *true*, traverses the branch headed by the given Atom. Default is *false*.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return the first found child from the specified type under the given parent. If no parent was specified, the returned Atom Instance is a direct or indirect child of the current Atom.

If the *recursive* flag is omitted, this command is equivalent to the [child\(\)](#) command.

find_var

atom = *find_var(name [,start_at_parent])*

atom : Atom Instance that defines the given variable.

name : Name of the variable to find.

start_at_parent : If *true*, will only start searching for the variable at the current Atom's parent. Default is *false*.

Command category : Variables

Allowed in sections : Create, Pre, Exec, Post, Functions

Search for a given variable in the variable search path, starting at the current Atom Instance. If the *start_at_parent* argument is used, then the search will be normally performed but local variables (at the current Atom Instance) are ignored.

If the variable is found, a pointer to the defining Atom Instance is returned. Otherwise, *nil* is returned.

forall

```
output = forall(child_type, format, func_name [,recursive [,parent_to_use]])
```

output : Text as result of child processing

child_type : String that specifies the type of Atom Instance to process.

format : String with one of these two options: "csv" or "list".

func_name : Name of the function to call on each processed child Atom.

recursive : If *true*, this function will recurse the current tree branch. Default is *false* (process only direct children).

parent_to_use : Instead of processing those child Atoms of the current Atom Instance, the function will process children of the specified Atom Instance.

Command category : Atoms

Allowed in sections : Create, Pre, Exec, Post, Functions

Iterates on child Atom Instances and returns the resulting text, according to a specified format. The specified function must return a string that will then be appended to the resulting string.

get_atom

```
atom = get_atom(atom_id)
```

atom : Pointer to the requested Atom if the given ID is valid; *nil* otherwise.

atom_id : ID of the Atom to get

Command category : Atoms

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return a pointer to the Atom with the given ID.

get_atom_id

```
atom_id = get_atom_id(atom)
```

atom_id : ID of the supplied Atom Instance

atom : Atom to query

Command category : General

Allowed in sections : Admin, Create, Pre, Exec, Post, Functions

This function returns a number that is the ID of the given Atom Instance. This is a unique string among the project's Atoms.

get_current_file

```
filename = get_current_file()
```

filename : String with the name of the file where the generation cursor is currently placed.

Command category : File

Allowed in sections : Exec, Post, Functions

This function will return the name of the file where the [Generation Cursor](#) is currently placed.

get_id

```
atom_id = get_id()
```

atom_id : ID of the Atom Instance being executed or interpreted

Command category : General

Allowed in sections : Admin, Create, Pre, Exec, Post, Functions

This function returns a number that is the ID of the Atom Instance being executed. This is a unique integer among the project's Atoms. However, if you are using this function on an Atom from a Library (located under the Auto-Include or Optional Include branch), the returned ID may collide with the ID of an Atom on the project model.

Keep in mind that this command is never run on the Atom Template, but only on its Atom Instances. Therefore, you cannot get the Atom ID of an Atom Template through this function.

get_type

```
result = get_type(atom)
```

result : Returns a string with the name of the Atom Template the supplied Atom Instance is an instance of.

atom : Atom Instance to query

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function returns the name of the Atom Template from which the supplied Atom is an instance.

Example:

```
if get_type(instance1) == get_type(instance2) then
  ...
end
```

The above code checks if two Atom Instances have the same type.

help

help(text)

text : String that contains a text that explains the use of this Atom Template

Command category : Presentation

Allowed in sections : Admin

Inherited from base class(es) : yes

Define help text to describe the use of this Atom Template.

In places where an Atom Template selection has to be made, the text defined by this command may be shown to help identify this particular Atom Template, or its purpose.

For example, the Atom Template's help text is shown when a selection is made on the [New Instance dialog](#).

hide_param

hide_param(name)

name : Name of the parameter to hide.

Command category : Structure

Allowed in sections : Form

Inherited from base class(es) : yes

Hides a parameter, effectively removing it from the Instance's form. Use this command when a specific Atom Template parameter is not necessary for a given scenario, or when it is automatically defined on a Derived Template.

icon

icon(name)

name : String that contains the name of an existing icon in AtomWeaver's icon library.

Command category : Presentation

Allowed in sections : Admin

Inherited from base class(es) : yes

Set the icon to use for all instances of this Template. The icon is displayed next to the Atom Template and all instances of it. Derived Atom Templates will also display this icon unless they override it with another icon() command.

include_all_libs

include_all_libs()

Command category : Library Include

Allowed in sections : Create

Makes the current Atom Instance the insertion point of the Auto-Include and Optional-Include Atoms from all Atom Libraries. The Atom Libraries that are explicitly included in other Atoms, through the *include_lib()* command, are not included at this point.

The library's Include Atoms become virtual branches of the current Atom Instance.

See [this page](#) for an explanation of Include Atoms.

include_lib

include_lib(lib_prefix)

lib_prefix : Prefix of the Atom Library to include

Command category : Library Include

Allowed in sections : Create

Makes the current Atom Instance the insertion point of the specified Library's Auto-Include and Optional-Include Atoms.

The library's Include Atoms become a virtual branch of the current Atom Instance.

See [this page](#) for an explanation of Include Atoms.

is_child_of

result = is_child_of(atom, parent)

result : Returns *true* if the first supplied Atom Instance is under the second Atom Instance. Returns *false* otherwise.

atom : First Atom Instance to query. If *nil*, the current Atom Instance is used.

parent : Second Atom Instance to query, considered to be a parent of the first.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function checks if the first supplied Atom Instance is under the second Atom Instance. For the condition to be considered true, the first given Atom does not need to be a direct child of the second. Any tree depth distance is valid as long as the first Atom is on a branch headed by the second Atom.

is_empty

result = is_empty(name)

result : Returns *true* if the variable is empty, otherwise returns *false*

name : Name of the variable to inspect

Command category : Variables

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This function checks if the specified variable is empty. Internally this function calls the [var\(\)](#) function, therefore all the rules of variable search applies to this function too.

If the specified variable is of the Atom type, this function will return *true* (empty) if there is no Atom defined, and *false* (not empty) if there is one.

is_neutral

result = is_neutral(atom)

result : Returns *true* if the supplied Atom Instance is [neutral](#). Returns *false* if not.

atom : Atom Instance to query. If *nil*, the current Atom Instance is used.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function checks if the supplied Atom Instance is neutral.

is_relocated

result = is_relocated(atom)

result : Returns *true* if the supplied Atom Instance is [relocated](#). Returns *//false//* if not.

atom : Atom Instance to query. If *nil*, the current Atom Instance is used.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function checks if the supplied Atom Instance is relocated. Relocated Atoms are Auto-Generated Atoms that are moved under another parent through the evaluation of an associated expression.

is_type

```
result = is_type(atom, template_name)
```

result : Returns *true* if the supplied Atom Instance is an instance of the given Atom Template.

atom : Atom Instance to query. If *nil*, uses the current Atom Instance

template_name : Atom Template type to check

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function checks if the supplied Atom Instance is an Instance of the specified Atom Template. Base Templates are also taken into consideration:



Therefore, using the above example, the statements:

```
is_type(MyCustomerAtom, "root_default")
is_type(MyCustomerAtom, "java_class")
is_type(MyCustomerAtom, "bank_customer")
```

would all return *true*.

is_under

```
result = is_under(atom, parent_type)
```

result : Returns *true* if the supplied Atom Instance is under another of the specified type. Returns *false* otherwise.

atom : Atom Instance to query. If *nil*, the current Atom Instance is used.

parent_type : Atom Template type to check

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function checks if the supplied Atom Instance is on a branch headed by an Instance of the given type. This functions keeps searching for a suitable parent until it reaches the model tree's root.

label

label(label_formula [,description_formula])

label_formula : String that may contain a [formula](#) to define the Atom Instance's label to be presented on the ABSE Tree.

description_formula : String that may contain a formula, and which identifies a particular Atom Instance. This parameter is optional.

Command category : Presentation

Allowed in sections : Admin

Inherited from base class(es) : yes

Sets the formulas to use when computing the Atom Instance's label and description. The label formula is used to display a string next to the Atom Instance's icon on a project tree. The description formula is used to identify a particular Atom Instance on a list or other selection operation.

line

line(contents [,no_translation])

contents : Line(s) to insert at the current generation cursor position.

no_translation : If *true*, the command will not evaluate variables on the given contents. Default is *false* (evaluate variables).

Command category : File

Allowed in sections : Exec, Post, Functions

This command will insert the given contents at the [Generation Cursor](#)'s current position. If the supplied string is multi-line, then it will be split and each line will be used independently (as multiple calls to *line()*).

Some escape sequences are translated:

\n : (Will split current line in two)

\t : Translated to a TAB character

{} : Translated to {{ to avoid inserting a file section

**** : Translated to \

list

list(list_path, category)

list_path : String that represents a path to the list in the format <main-list>/<sub-list>/.../<list>. Use the double-slash ("//")escape sequence to use a slash in a path element.

category : A category under the specified list.

Command category : Organization

Allowed in sections : Create

Classifies the Atom Instance by adding it to an Atom List. Atom Lists form a tree whose branches you can freely define. If the path you specify does not exist, it is automatically created. Theoretically there is no limit to the depth of the Atom Lists tree.

The resulting Atom List structure will be used by interactive commands to help you classify, find and access Atom Instances in a simple, intuitive, and quick way.

Example:

```

java_class
└─ Admin
  └─ param_text("def", "class_domain", "Class Domain", "Domain this class belongs to")
  └─ param_text("def", "class_category", "Class Category", "Classification of this class th

```

```

java_class
└─ Create
  └─ list("class/@class_domain", "@class_category")

```

In this example, two Atom Parameters are used to classify the corresponding Atom Instances into the top-level list named *class*, while another list is created under it for each domain you specify. Then on each list, a category is specifically defined by you.

With the above commands, you can obtain an Atom List structure like this (an hypothetical example):

```

└─ Class
  └─ Networking
  └─ User Interface
  └─ Database

```

list_children

```
children = list_children(atom [,child_type [,recursive]])
```

children : List of child Atom Instances from the requested type (or of any type if none was specified).

atom : Atom to use as parent. If no Atom is specified (nil is supplied), the current Atom Instance is used.

child_type : Type of child to get (Atom Template name). Default is empty (any type).

recursive : If *true*, will recurse the branch headed by the parent Atom. Otherwise only the direct children are listed. Default is *false*.

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return a list of Atom Instances that conform to the specified criteria. All Atoms are considered into inclusion including neutral Atoms.

Examples:

```
local children = list_children()
```

Lists all direct children of the current Atom Instance.

```
local children = list_children(nil, "java_function_args")
```

Lists all arguments of this function (for instance, if the current Atom is of type *java_function*).

```
local children = list_children(project_root_atom, "java_class", true)
```

Lists all Java classes in the project.

log

```
log(log_path, message [,icon])
```

log_path : Path to the log in the *log/sub_log/...* format. The last log in the path is where the message will be appended. If empty, the *Global* log is used.

message : Message to add. Embedded variables will be evaluated.

icon : ID of the icon to include. Default is "info".

Command category : General

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This command will append a message to the specified log.

neutral

```
neutral()
```

Command category : Constraint; Classification

Allowed in sections : Admin

Inherited from base class(es) : yes

Specifies that the atoms derived from this Atom Template are invisible when searching for a parent.

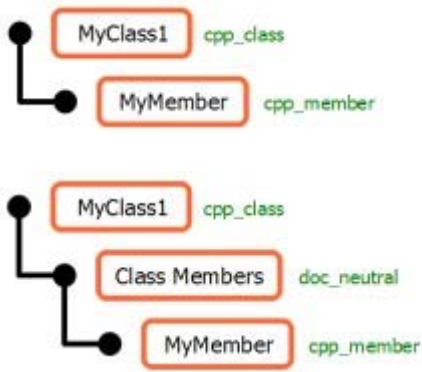
Example:

```

cpp_member
  Admin
  under("cpp_class")

```

The following constructions are allowed:



In the presence of a neutral atom, constraint checking moves to the next non-neutral parent.

oparent

```
atom = oparent([relocated_atom [,accept_neutral]])
```

atom : Original parent of the supplied relocated Atom

relocated_atom : Atom whose original parent we want to find. If no Atom is supplied, the current Atom Instance is used.

accept_neutral : If *true*, neutral Atoms are taken into consideration. Default is *false* (neutral Atoms are skipped).

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return the original parent of the supplied Atom.

If the supplied Atom was not directly relocated, but is on a relocated branch, the branch's head Atom will be considered instead.

If the supplied Atom is inactive this function will behave like the `parent()` command, returning its direct parent.

param

```
var_contents = param(name [,start_atom])
```

var_contents : Contents of the requested variable. Can be a string or a pointer to an Atom Instance

name : Name of the parameter-bound variable to get

start_atom : Atom where to start the search. If *nil*, start searching from the current Atom Instance

Command category : Variables

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This function will return the contents of a Parameter-bound variable. If the current Atom

Instance does not define such variable, it's parent Atom will be searched. This search continues until the Model's Tree root is reached. Then global variables are searched and if no variable is found, an error is raised.

Unlike the [var\(\)](#) function, this function will not search for temporary variables. Only variables that directly correspond to Atom Template Parameters, either set by the Instance's form, or on its Create Section, are considered.

param_area

param_area(area_name, area_type, caption, help_text)

area_name : Identifier of this specific area

area_type : Type of this area. Defining a type will allow adequate presentation and processing, like syntax highlighting.

caption : Area caption text. To be presented next to the area editor.

help_text : Area help text. To be presented under the caption text, next to the area editor.

Command category : Definition

Allowed in sections : Admin

Inherited from base class(es) : yes

Define a code area for this Atom Template. The specification of a type will allow you to apply special presentation or interactive helpers like syntax highlighting.

param_block

param_block(group, var_name, caption, help)

group : Name of the group this parameter belongs to.

var_name : Name of this parameter and also the name of the variable to be defined on Atom Instances.

caption : Title of this parameter, to be shown on the Atom Instance form.

help : Help text for this parameter.

Command category : Definition

Allowed in sections : Admin

Inherited from base class(es) : yes

Define a text block (multi-line text) parameter for this Atom Template.

Note: This parameter will only appear on an Atom Instance form if its group is defined in such form through the [param_group\(\)](#) command.

param_bool

param_bool(group, var_name, caption, help)

group : Name of the group this parameter belongs to.

var_name : Name of this parameter and also the name of the variable to be defined on Atom Instances.

caption : Title of this parameter, to be shown on the Atom Instance form.

help : Help text for this parameter.

Command category : Definition

Allowed in sections : Admin

Inherited from base class(es) : yes

Define a boolean parameter for this Atom Template.

Note: This parameter will only appear on an Atom Instance form if its group is defined in such form through the [param_group\(\)](#) command.

param_group

param_group(id, order, caption, help)

id : String that identifies this group.

order : Order of this group on the form. Can be any positive number. Groups are shown in ascending order.

caption : Group title, to be shown on the form.

help : Help text to be shown under the group's caption.

Command category : Structure

Allowed in sections : Form

Inherited from base class(es) : yes

Groups together several Atom Template parameters that have been tagged with the same group ID. Check the *group* argument on the [param_block\(\)](#), [param_bool\(\)](#), [param_link\(\)](#), [param_list\(\)](#) and [param_text\(\)](#) Admin commands.

param_link

param_link(group, var_name, caption, lists, help)

group : Name of the group this parameter belongs to.

var_name : Name of this parameter and also the name of the variable to be defined on Atom Instances.

caption : Title of this parameter, to be shown on the Atom Instance form.

lists : Atom Lists where the linked Atom Instance must belong to. If empty, any Atom is accepted.

help : Help text for this parameter.

Command category : Definition

Allowed in sections : Admin

Inherited from base class(es) : yes

Define a link-type parameter for this Atom Template.

This type of parameter should be used when referencing other Atoms. The corresponding variable on an Atom Instance can be used to obtain additional information about the referenced Atom Instance or other related Atoms.

Note: This parameter will only appear on an Atom Instance form if its group is defined in such form through the [param_group\(\)](#) command.

param_list

param_list(group, var_name, caption, options, help)

group : Name of the group this parameter belongs to.

var_name : Name of this parameter and also the name of the variable to be defined on Atom Instances.

caption : Title of this parameter, to be shown on the Atom Instance form.

options : Comma-separated list of selectable strings.

help : Help text for this parameter.

Command category : Definition

Allowed in sections : Admin

Inherited from base class(es) : yes

Define a list-selection parameter for this Atom Template. The corresponding variable will store the string that was interactively selected on the Atom Instance form.

Note: This parameter will only appear on an Atom Instance form if its group is defined in such form through the [param_group\(\)](#) command.

param_text

param_text(group, var_name, caption, help)

group : Name of the group this parameter belongs to.

var_name : Name of this parameter and also the name of the variable to be defined on Atom Instances.

caption : Title of this parameter, to be shown on the Atom Instance form.

help : Help text for this parameter.

Command category : Definition

Allowed in sections : Admin

Inherited from base class(es) : yes

Define a single-line text parameter for this Atom Template. When instantiating the Atom Template, the specialized editor that will be shown will include this parameter as input.

Note: This parameter will only appear on an Atom Instance form if its group is defined in such form through the [param_group\(\)](#) command.

parent

```
parent_atom = parent([atom [,parent_type [,silent [,accept_neutral]]]])
```

parent_atom : Requested parent Atom. *nil* if no suitable parent found.

atom : Atom Instance to query. If *nil*, the current Atom Instance is considered. Default is *nil*.

parent_type : Type of parent Atom to get. Default is empty string (any type).

silent : If *true*, will not raise an error if a suitable parent is not found. Default is *false* (raise an error).

accept_neutral : If *true*, neutral Atoms are taken into consideration. Default is *false* (neutral Atoms are skipped).

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function returns the parent Atom of the current Atom Instance or of another Atom if specified. The immediate parent is returned unless a specific Atom Template type is given. In that case, the function will climb the tree until a valid match is found. If neutral Atoms are found along the way, they are ignored unless the *accept_neutral* option is specified.

The function will raise an error if a suitable parent is not found, unless the *silent* option is specified.

Examples:

```
atom = parent()
```

Get the parent of the current Atom Instance.

```
atom = parent(nil,"java_class")
```

Get an ancestor of the current Atom Instance that is or derives from *java_class*.

```
atom = parent(nil,"java_class")
feature = parent(atom,"domain_feature")
```

Get the domain feature that contains the Java class the current Atom Instance belongs to.

pvar

```
var_contents = pvar(name)
```

var_contents : Contents of the requested variable. Can be a string or a pointer to an Atom Instance

name : Name of the variable to get

Command category : Variables

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This function will return the contents of an Atom Instance variable. But unlike the [var\(\)](#) function, this function will not search the current Atom Instance. Instead, it will start searching at its parent. If the parent Atom Instance does not define such variable, its own parent Atom will be searched. This search continues until the Model's Tree root is reached. Then global variables are searched and if no variable is found, an error is raised.

pvar stands for "parent's variable".

section

section(name)

name : Name of section to insert

Command category : File

Allowed in sections : Exec, Post, Functions

This command will insert a [file section](#) at the [Generation Cursor](#)'s current position. Existence of other sections with the same name is not verified. Therefore, a call to *cursor("duplicate_section")* will always make the cursor move to the first found section.

This command can be embedded in the source code through the

```
{{section_name}}
```

construct, which should be the only text in that line.

set

set(name, contents [,atom])

name : Name of the variable to set

contents : Contents to set. Can be a string or a pointer to an Atom Instance

atom : Atom where to set this variable

Command category : Variables

Allowed in sections : Create, Pre, Exec, Post, Functions

Create or change a variable on the current Atom Instance. If the *atom* argument is given, then the variable is set on the specified Atom Instance.

This command behaves differently according to the Atom Section it is being run:

If this command is invoked inside the Create Section a regular variable is created. When invoked on other sections, a temporary variable is created. A temporary variable overrides the contents of regular variables, but it will be removed in the next generation cycle.

You cannot set a variable directly on another Atom (using the *atom* argument) if the command is located on the Create Section.

set_global

set_global(name, contents)

name : Name of the variable to set

contents : Contents to set. Can be a string or a pointer to an Atom Instance

Command category : Variables

Allowed in sections : Create, Pre, Exec, Post, Functions

Create or change a variable on the Project Tree's own variable table. This variable will be accessible to all the Tree's Atom Instances.

setdef

setdef(param_name, contents)

param_name : Name of the parameter to define.

contents : Contents to set on the corresponding form control.

Command category : Contents

Allowed in sections : Form

Inherited from base class(es) : yes

Sets a default value to be presented on the control that corresponds to the given parameter on the Atom Instance form. The value is not automatically set on the Atom Instance.

tree_root

atom = tree_root()

atom : Atom that represents the tree's root

Command category : Atom

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will return the invisible Atom Instance that represents the root of the tree being executed. This Atom has limited use, but can be useful on functions or commands where a "parent" must be supplied. For instance, using *tree_root()* as the parent argument on a [child_count\(\)](#), [find_by_var\(\)](#) or [find_child\(\)](#) function, you can search on the entire tree.

under

under(template_name)

template_name : Type of the parent Atom Instance in whose branch the instances of this Atom Template are allowed to be part of.

Command category : Constraint

Allowed in sections : Admin

Inherited from base class(es) : no

Defines where the Instances of this Atom Template can be created as children. More than one instance can be added under the specified parent.

unique_section

unique_section(name)

name : Name of section to insert

Command category : File

Allowed in sections : Exec, Post, Functions

This command will insert a [file section](#) at the [Generation Cursor](#)'s current position. This section will have a special behavior: All lines that are inserted on this section are unique. If an attempt is made to insert a duplicate line, it is silently discarded. Empty lines are also discarded.

Existence of other sections with the same name is not verified. Therefore, a call to `cursor("duplicate_section")` will always make the cursor move to the first found section.

This command can be embedded in the source code through the

```
{u{section_name}}
```

construct, which should be the only text in that line.

unset

unset(name [,atom])

name : Name of the variable to remove

atom : Atom where to remove this variable

Command category : Variables

Allowed in sections : Create, Pre, Exec, Post, Functions

Remove a temporary variable from the current Atom Instance. If the *atom* argument is given, then the variable is removed on the specified Atom Instance. If no variable is found at the specified Atom, nothing will happen and no error is raised.

Please note that only temporary variables can be removed. If you remove a temporary variable

and a parameter-bound variable exists on the same Atom, the parameter-bound variable will take its place.

var

var_contents = *var*(*name* [,*start_atom*])

var_contents : Contents of the requested variable. Can be a string or a pointer to an Atom Instance

name : Name of the variable to get

start_atom : Atom where to start the search. If *nil*, start searching from the current Atom Instance

Command category : Variables

Allowed in sections : Admin, Form, Create, Pre, Exec, Post, Functions

This function will return the contents of an Atom Instance variable. If the current Atom Instance does not define such variable, it's parent Atom will be searched. This search continues until the Model's Tree root is reached. Then global variables are searched and if no variable is found, an error is raised.

var_exists

result = *var_exists*(*name* [,*start_at_parent* [,*local_only*]])

result : *true* : The specified variable exists; *false* : The specified variable does not exist or is not reachable.

name : Name of the variable to check.

start_at_parent : If *true*, will only start searching for the variable at the current Atom's parent. Default is *false*.

local_only : If *true*, will just check the current Atom Instance. Default is *false*.

Command category : Variables

Allowed in sections : Create, Pre, Exec, Post, Functions

Check if a given variable exists in the variable seek path, starting at the current Atom Instance. If the *start_at_parent* argument is used, then the search will be normally performed but local variables (at the current Atom Instance) are ignored. However, if the *local_only* argument is used, then the opposite occurs: only the local variables are taken into account, and no upward search is made.

The *start_at_parent* and *local_only* arguments are mutually exclusive in the sense that only one of them can be *true*.

vfunc

vfunc(behavior, function)

behavior : String that specifies a global action

function : Function that will be called when the global action is applied to an Instance of this Template


Command category : Atom Behaviors

Allowed in sections : Functions

This command will declare a function as the handler of a globally defined [Atom Behavior](#). Any Atom Instance will be able to run a function on Instances of this Atom Template even without knowing which function to call. For that matter, a common "behavior" will be invoked by issuing a request using a globally recognized string, through the `vfunc_exec()` command.

The declaration of an Atom Behavior on an Atom Template may override the declaration of one of its Base Templates. In that case, the newly declared Behavior will be called instead of the inherited one.

Example:



```
function cpp_class_get_header_file(args)
    return var("headerfile")
end

vfunc("get_header_file", cpp_class_get_header_file)
```

The above code show the declaration of an extra function of the *cpp_class* Atom Template. The *vfunc()* command turns that function into an Atom Behavior by associating it with the *get_header_file* behavior.

vfunc_exec

value = vfunc_exec(atom, behavior, args)

value : A value returned by the invoked Atom Behavior

atom : Atom Instance where the Atom Behavior will be invoked

behavior : String that identifies the Atom Behavior to invoke

args : A Lua table that contains useful input arguments for the invoked Atom Behavior, and optionally hold results.

Command category : Atom Behaviors

Allowed in sections : Create, Pre, Exec, Post, Functions

This function will invoke an Atom Behavior on a designated Atom Instance. If its Atom Template does not have an handler to that behavior, the equivalent Behavior on the Base Template will be called. This search will continue until *root_default* is reached.

Example:

```

local args = {}
local include_file = vfunc_exec(var("base_class"), "get_header_file", args)
cursor("include_files")
line("#include \" .. include_file .. \""")

```

With this code fragment we wanted to include the declaration of this class' base class. For that, we invoked the *get_header_file* behavior on the Atom Instance pointed by the local Atom's *base_class* variable. Then we moved the generation cursor to the "include_files" insertion point, and added the intended include statement.

ABSE Commands & Functions by Section

These pages list all the ABSE commands and functions that can be used on a given Atom Template Section

Available Admin Section Commands

[abstract](#) - Specifies that the Atom Template cannot be directly instantiated.

[dict](#) - Define a dictionary entry for every instance of an Atom Template.

[doc](#) - Specifies that the Template's Instances are for documentation purposes only.

[help](#) - Define help text to describe the use of an Atom Template.

[icon](#) - Set the icon to display on all Instances of a Template.

[label](#) - Sets the formulas to use when computing the Atom Instance's label and description.

[neutral](#) - Specifies that the atoms derived from this Atom Template are invisible to several ABSE rules.

[param_area](#) - Define a code area for an Atom Template.

[param_block](#) - Define a text block (multi-line text) parameter for an Atom Template.

[param_bool](#) - Define a boolean parameter for an Atom Template.

[param_link](#) - Define a link-type parameter for an Atom Template.

[param_list](#) - Define a list-selection parameter for an Atom Template.

[param_text](#) - Define a single-line text parameter for this Atom Template.

[under](#) - Defines where the Instances of an Atom Template can be created as children.

Available Form Section Commands

[hide_param](#) - Hides a parameter, removing it from the Instance's form (but not from the Atom Template).

[param_group](#) - Groups together several Atom Template parameters that have been tagged with the same group ID.

[setdef](#) - Sets a default value to be presented on the control that corresponds to the given

parameter on the Atom Instance form.

Available Create Section Commands

[child](#) - Return the first direct child Atom from a specified type.

[child_count](#) - Return the number of direct child Atoms from a specified type.

[counter_get](#) - Get the current value of a counter and increment/decrement automatically.

[counter_new](#) - Create a counter that can then be used to obtain unique integers.

[cvar](#) - Return the contents of a variable from a direct child Instance.

[dvar](#) - Return the contents of a variable directly from a specified Instance.

[eval](#) - Translate a string into another by evaluating embedded variables.

[exec](#) - Calls a user-defined function in the context of a given Atom.

[find_by_var](#) - Return the first found child from a specified type that also has a variable whose contents match those supplied.

[find_child](#) - Return the first found child from a specified type under the given parent.

[find_var](#) - Search for an Atom Instance given a variable that is defined on that Atom.

[forall](#) - Iterates on child Atom Instances and returns the resulting text, according to a specified format.

[get_atom](#) - Return a reference to an Atom, given its ID.

[get_atom_id](#) - Returns the ID of a given Atom Instance.

[get_id](#) - Return the ID of the Atom Instance being executed.

[get_type](#) - Return an Atom Instance's type (the name of its Atom Template).

[include_all_libs](#) - Makes an Atom Instance the insertion point of Auto-Include and Optional-Include Atoms from all Atom Libraries.

[include_lib](#) - Makes an Atom Instance the insertion point of Auto-Include and Optional-Include Atoms from a given Atom Library.

[is_child_of](#) - Check if an Atom Instance is under another, specified Instance.

[is_empty](#) - Check if the specified variable is empty.

[is_neutral](#) - Check if the supplied Atom Instance is neutral.

[is_relocated](#) - Check if the supplied Atom Instance is relocated.

[is_type](#) - Check the type of a given Atom Instance.

[is_under](#) - Check if the supplied Atom Instance is on a branch headed by an Instance of the given type.

[list](#) - Classifies Atom Instances by adding them to an Atom List.

[list_children](#) - Return a list of Atom Instances that conform to the specified criteria.

[log](#) - Append a message to the specified log.

[param](#) - Returns the contents of a Parameter-bound variable.

[parent](#) - Return the parent Atom of the current Atom Instance or of another Atom if specified.

[pvar](#) - Get the contents of a variable, but ignore local variables (start searching at the parent Atom).

[set](#) - Create or change a variable on the current Atom Instance.

[set_global](#) - Create or change a variable on the Tree's own variable table.

[tree_root](#) - Return the invisible Atom Instance that represents the root of the tree being executed.

[unset](#) - Remove a temporary variable from the current Atom Instance.

[var](#) - Return the contents of a variable, following ABSE's variable seek rules.

[var_exists](#) - Check if a given variable exists in the variable seek path.

[vfunc_exec](#) - Invoke an Atom Behavior on a designated Atom Instance.

Available Pre Section Commands

[child](#) - Return the first direct child Atom from a specified type.

[child_count](#) - Return the number of direct child Atoms from a specified type.

[counter_get](#) - Get the current value of a counter and increment/decrement automatically.

[counter_new](#) - Create a counter that can then be used to obtain unique integers.

[cvar](#) - Return the contents of a variable from a direct child Instance.

[dvar](#) - Return the contents of a variable directly from a specified Instance.

[eval](#) - Translate a string into another by evaluating embedded variables.

[exec](#) - Calls a user-defined function in the context of a given Atom.

[find_by_var](#) - Return the first found child from a specified type that also has a variable whose contents match those supplied.

[find_child](#) - Return the first found child from a specified type under the given parent.

[find_var](#) - Search for an Atom Instance given a variable that is defined on that Atom.

[forall](#) - Iterates on child Atom Instances and returns the resulting text, according to a specified format.

[get_atom](#) - Return a reference to an Atom, given its ID.

[get_atom_id](#) - Returns the ID of a given Atom Instance.

[get_id](#) - Return the ID of the Atom Instance being executed.

[get_type](#) - Return an Atom Instance's type (the name of its Atom Template).

[is_child_of](#) - Check if an Atom Instance is under another, specified Instance.

[is_empty](#) - Check if the specified variable is empty.

[is_neutral](#) - Check if the supplied Atom Instance is neutral.

[is_relocated](#) - Check if the supplied Atom Instance is relocated.

[is_type](#) - Check the type of a given Atom Instance.

[is_under](#) - Check if the supplied Atom Instance is on a branch headed by an Instance of the given type.

[list_children](#) - Return a list of Atom Instances that conform to the specified criteria.

[log](#) - Append a message to the specified log.

[param](#) - Returns the contents of a Parameter-bound variable.

[parent](#) - Return the parent Atom of the current Atom Instance or of another Atom if specified.

[pvar](#) - Get the contents of a variable, but ignore local variables (start searching at the parent Atom).

[set](#) - Create or change a variable on the current Atom Instance.

[set_global](#) - Create or change a variable on the Tree's own variable table.

[tree_root](#) - Return the invisible Atom Instance that represents the root of the tree being executed.

[unset](#) - Remove a temporary variable from the current Atom Instance.

[var](#) - Return the contents of a variable, following ABSE's variable seek rules.

[var_exists](#) - Check if a given variable exists in the variable seek path.

[vfunc_exec](#) - Invoke an Atom Behavior on a designated Atom Instance.

Available Exec Section Commands

[block](#) - Insert the given contents at the Generation Cursor's current position, using a mask.

[child](#) - Return the first direct child Atom from a specified type.

[child_count](#) - Return the number of direct child Atoms from a specified type.

[code](#) - Insert a block of code at the Generation Cursor's current position.

[common_section](#) - Insert a common-type file section at the Generation Cursor's current position.

[counter_get](#) - Get the current value of a counter and increment/decrement automatically.

[counter_new](#) - Create a counter that can then be used to obtain unique integers.

[cursor](#) - Place or move the [Generation Cursor](#).

[cvar](#) - Return the contents of a variable from a direct child Instance.

[dvar](#) - Return the contents of a variable directly from a specified Instance.

[eval](#) - Translate a string into another by evaluating embedded variables.

[exec](#) - Calls a user-defined function in the context of a given Atom.

[find_by_var](#) - Return the first found child from a specified type that also has a variable whose contents match those supplied.

[find_child](#) - Return the first found child from a specified type under the given parent.

[find_var](#) - Search for an Atom Instance given a variable that is defined on that Atom.

[forall](#) - Iterates on child Atom Instances and returns the resulting text, according to a specified format.

[get_atom](#) - Return a reference to an Atom, given its ID.

[get_atom_id](#) - Returns the ID of a given Atom Instance.

[get_current_file](#) - Return the name of the file where the Generation Cursor is currently placed.

[get_id](#) - Return the ID of the Atom Instance being executed.

[get_type](#) - Return an Atom Instance's type (the name of its Atom Template).

[icon](#) - Set the icon to display on all Instances of a Template.

[is_child_of](#) - Check if an Atom Instance is under another, specified Instance.

[is_empty](#) - Check if the specified variable is empty.

[is_neutral](#) - Check if the supplied Atom Instance is neutral.

[is_relocated](#) - Check if the supplied Atom Instance is relocated.

[is_type](#) - Check the type of a given Atom Instance.

[is_under](#) - Check if the supplied Atom Instance is on a branch headed by an Instance of the given type.

[line](#) - Insert a single line of code at the Generation Cursor's current position.

[list_children](#) - Return a list of Atom Instances that conform to the specified criteria.

[log](#) - Append a message to the specified log.

[param](#) - Returns the contents of a Parameter-bound variable.

[parent](#) - Return the parent Atom of the current Atom Instance or of another Atom if specified.
[pvar](#) - Get the contents of a variable, but ignore local variables (start searching at the parent Atom).

[section](#) - Insert a file section at the [Generation Cursor](#)'s current position.
[set](#) - Create or change a variable on the current Atom Instance.
[set_global](#) - Create or change a variable on the Tree's own variable table.

[tree_root](#) - Return the invisible Atom Instance that represents the root of the tree being executed.

[unique_section](#) - Insert a unique-type file section at the Generation Cursor's current position.
[unset](#) - Remove a temporary variable from the current Atom Instance.

[var](#) - Return the contents of a variable, following ABSE's variable seek rules.
[var_exists](#) - Check if a given variable exists in the variable seek path.
[vfunc_exec](#) - Invoke an Atom Behavior on a designated Atom Instance.

Available Post Section Commands

[block](#) - Insert the given contents at the Generation Cursor's current position, using a mask.

[child](#) - Return the first direct child Atom from a specified type.
[child_count](#) - Return the number of direct child Atoms from a specified type.
[code](#) - Insert a block of code at the Generation Cursor's current position.
[common_section](#) - Insert a common-type file section at the Generation Cursor's current position.
[counter_get](#) - Get the current value of a counter and increment/decrement automatically.
[counter_new](#) - Create a counter that can then be used to obtain unique integers.
[cursor](#) - Place or move the [Generation Cursor](#).
[cvar](#) - Return the contents of a variable from a direct child Instance.

[dvar](#) - Return the contents of a variable directly from a specified Instance.

[eval](#) - Translate a string into another by evaluating embedded variables.
[exec](#) - Calls a user-defined function in the context of a given Atom.

[find_by_var](#) - Return the first found child from a specified type that also has a variable whose contents match those supplied.
[find_child](#) - Return the first found child from a specified type under the given parent.
[find_var](#) - Search for an Atom Instance given a variable that is defined on that Atom.
[forall](#) - Iterates on child Atom Instances and returns the resulting text, according to a specified format.

[get_atom](#) - Return a reference to an Atom, given its ID.
[get_atom_id](#) - Returns the ID of a given Atom Instance.
[get_current_file](#) - Return the name of the file where the Generation Cursor is currently placed.
[get_id](#) - Return the ID of the Atom Instance being executed.
[get_type](#) - Return an Atom Instance's type (the name of its Atom Template).

[icon](#) - Set the icon to display on all Instances of a Template.
[is_child_of](#) - Check if an Atom Instance is under another, specified Instance.
[is_empty](#) - Check if the specified variable is empty.

[is_neutral](#) - Check if the supplied Atom Instance is neutral.

[is_relocated](#) - Check if the supplied Atom Instance is relocated.

[is_type](#) - Check the type of a given Atom Instance.

[is_under](#) - Check if the supplied Atom Instance is on a branch headed by an Instance of the given type.

[line](#) - Insert a single line of code at the Generation Cursor's current position.

[list_children](#) - Return a list of Atom Instances that conform to the specified criteria.

[log](#) - Append a message to the specified log.

[param](#) - Returns the contents of a Parameter-bound variable.

[parent](#) - Return the parent Atom of the current Atom Instance or of another Atom if specified.

[pvar](#) - Get the contents of a variable, but ignore local variables (start searching at the parent Atom).

[section](#) - Insert a file section at the [Generation Cursor](#)'s current position.

[set](#) - Create or change a variable on the current Atom Instance.

[set_global](#) - Create or change a variable on the Tree's own variable table.

[tree_root](#) - Return the invisible Atom Instance that represents the root of the tree being executed.

[unique_section](#) - Insert a unique-type file section at the Generation Cursor's current position.

[unset](#) - Remove a temporary variable from the current Atom Instance.

[var](#) - Return the contents of a variable, following ABSE's variable seek rules.

[var_exists](#) - Check if a given variable exists in the variable seek path.

[vfunc_exec](#) - Invoke an Atom Behavior on a designated Atom Instance.

Available Functions Section Commands

(Command availability on the Functions Section depends on the caller Section).

Specific commands for this section:

[vfunc](#) - Declare a function as the handler of a globally defined [Atom Behavior](#).

If the caller Section is Create, Pre, Exec or Post, the following commands are available:

[child](#) - Return the first direct child Atom from a specified type.

[child_count](#) - Return the number of direct child Atoms from a specified type.

[counter_get](#) - Get the current value of a counter and increment/decrement automatically.

[counter_new](#) - Create a counter that can then be used to obtain unique integers.

[cvar](#) - Return the contents of a variable from a direct child Instance.

[dvar](#) - Return the contents of a variable directly from a specified Instance.

[eval](#) - Translate a string into another by evaluating embedded variables.

[exec](#) - Calls a user-defined function in the context of a given Atom.

[find_by_var](#) - Return the first found child from a specified type that also has a variable whose contents match those supplied.

[find_child](#) - Return the first found child from a specified type under the given parent.

[find_var](#) - Search for an Atom Instance given a variable that is defined on that Atom.
[forall](#) - Iterates on child Atom Instances and returns the resulting text, according to a specified format.

[get_atom](#) - Return a reference to an Atom, given its ID.
[get_atom_id](#) - Returns the ID of a given Atom Instance.
[get_current_file](#) - Return the name of the file where the Generation Cursor is currently placed.
[get_id](#) - Return the ID of the Atom Instance being executed.
[get_type](#) - Return an Atom Instance's type (the name of its Atom Template).

[icon](#) - Set the icon to display on all Instances of a Template.
[is_child_of](#) - Check if an Atom Instance is under another, specified Instance.
[is_empty](#) - Check if the specified variable is empty.
[is_neutral](#) - Check if the supplied Atom Instance is neutral.
[is_relocated](#) - Check if the supplied Atom Instance is relocated.
[is_type](#) - Check the type of a given Atom Instance.
[is_under](#) - Check if the supplied Atom Instance is on a branch headed by an Instance of the given type.

[list_children](#) - Return a list of Atom Instances that conform to the specified criteria.
[log](#) - Append a message to the specified log.

[param](#) - Returns the contents of a Parameter-bound variable.
[parent](#) - Return the parent Atom of the current Atom Instance or of another Atom if specified.
[pvar](#) - Get the contents of a variable, but ignore local variables (start searching at the parent Atom).

[set](#) - Create or change a variable on the current Atom Instance.
[set_global](#) - Create or change a variable on the Tree's own variable table.

[tree_root](#) - Return the invisible Atom Instance that represents the root of the tree being executed.

[unset](#) - Remove a temporary variable from the current Atom Instance.

[var](#) - Return the contents of a variable, following ABSE's variable seek rules.
[var_exists](#) - Check if a given variable exists in the variable seek path.
[vfunc](#) - Declare a function as the handler of a globally defined [Atom Behavior](#).
[vfunc_exec](#) - Invoke an Atom Behavior on a designated Atom Instance.

If the caller Section is Exec or Post, the following commands are also available:

[block](#) - Insert the given contents at the Generation Cursor's current position, using a mask.

[code](#) - Insert a block of code at the Generation Cursor's current position.
[common_section](#) - Insert a common-type file section at the Generation Cursor's current position.
[cursor](#) - Place or move the [Generation Cursor](#).

[line](#) - Insert a single line of code at the Generation Cursor's current position.

[section](#) - Insert a file section at the [Generation Cursor](#)'s current position.

[unique_section](#) - Insert a unique-type file section at the Generation Cursor's current position.