Search documentation

Appian

Evaluation Functions

Appian 7.6

The following functions can be used in expression to perform complex evaluations.

Toggle Us

- bind()
- load()
- save()
- with()

NOTE: These functions are not supported in expressions for Portal reports or events.

For the full list of Appian Functions, see also: Appian Functions

bind()

When used in conjunction with load(), the `bind()` function lets you bind a getter and setter rule or function to a variable such that when that variable is read, the getter method is called and when it is saved into, the writer returned by the setter method is called.

Syntax

bind(*get*, *set*)

get: (Any Type) A rule or function that returns any type. The rule or function given as the `get` parameter will not be evaluated until the bound variable referenced in the expression.

set: (Writer) A rule or function that returns a writer. The writer will execute when the bound variable is written to in a `saveInto` in a SAIL UI.

Returns

Any Type

Notes

The `bind()` function can only be used when defining variables using the `load()` function on SAIL UIs.

The rule or function given as the `get` parameter will not be evaluated until the bound variable is evaluated in the expression. If the bound variable is never referenced in the expression, but only used in a `saveInto`, the `get` will be called prior to saving into the variable. If the bound variable is referenced many times in the expression and the `get` function is always passed the same parameters, it will only evaluate the `get` on the first reference and retrieve the result from cache for subsequent references.

Bound variables cannot be used as the `get` parameter of a subsequent `bind()` function.

The rule or function given as the `set` parameter must be a partial function with underscores indicating arguments that will be supplied when saving into the variable. The first blank argument will be filled with the value being saved into the variable. The second blank argument will be filled with one or more indices that are being saved into. Examples: * When storing into a variable such as `local!variable.field1`, the index argument supplied will be `field1` * When storing into a variable such as `local!variable[18]`, the index argument supplied will be `18` * When storing into a variable such as `local!employee.address.city`, the index argument supplied will be the array of indices `{address, city}`

When saving into the variable, the rule or function given as the *set* parameter must return a `writer`. If a rule is given as the `set` parameter, that rule must not use the `load()` function in its definition.

During the save evaluation when a `saveInto` is triggered in SAIL, the `writer` associated with the bound variable being saved into will execute its write method.

Any errors that occur during evaluation of the `get` will be handled as an expression evaluation error and cause the remainder of the expression to fail evaluation. Any errors that occur during evaluation of the `set` will occur after any non-bind saves are evaluated and will be displayed as an error to the user, but will not cause the evaluation of the expression to fail.

See also: Writer Functions

Examples

In the following example, the `bind()` function is used to Edit Data in an External System.

load()

Lets you define local variables within an expression for a SAIL interface and evaluate the expression with the new variables, then re-evaluate the function with the local variables' values from the previous evaluation.

Syntax

load(*localVar1*, ..., [*localVarN*], *expression*)

localVar1: (Any Type) The local variable to use when evaluating the given expression and defined using `load(local!a,..., expression)` or `load(local!a:10, ..., expression)`.

localVarN: (Any Type) Any additional local variables, as needed.

expression: (Any Type) The expression to evaluate using the local variables' values.

Returns

Any Type

Notes

This function is used in expressions for SAIL interfaces to allow for user interaction on the SAIL, such as sorting or paging through a grid.

You cannot use this function within the Web Content Channel or a looping function or else an error occurs.

A local variable's value is only calculated the first time the expression is evaluated and is then loaded back into the expression each time the expression is evaluated again within the same context. For SAIL interfaces, the context ends once the user navigates away from the SAIL.

A local variable may be defined with or without a value, and the value may be simple or complex. When a value is not defined, it's assigned a null value.

When you don't specify the `local!` domain, the system first matches your variables with rules or constants with the same name, then looks for local variables with the name. Appian recommends that you always use the `local!` domain when referring to local variables.

Local variables are not assigned a type. At runtime, the type of the variable will be based on the assigned value. For example, in `load(local!myvar:2, ...)`, `myvar` is of type `Integer`.

The type returned by the `load()` function will be that of the given expression.

A local variable may reference a previously defined local variable. They are evaluated in the given order.

The local variables are only available in the evaluation of the *expression*.

The *expression* can include other variables available in its context, such as process variables and rule inputs. For example, `load(local!myvar: 1, local!myvar + ri!myruleinput)`.

When a local variable uses the dot notation or brackets, a runtime error message will display. If the field name must contain special characters, enclose the name in single quotes.

Examples

To see the `load()` function in action, refer to the example walkthrough for creating a Tempo report with a grid SAIL component. It uses the `load()` function to save a `PagingInfo` value back into the expression when a user selects a column to sort by or another grid page to view.

See also: Grid Tutorial

a!save()

In SAIL saveInto parameters, updates the target with the given value. Use a!save for each item that you want to modify or alter in a `saveInto` parameter. This function has no effect when called outside of a component's saveInto parameter.

Syntax

save(*target*, *value*)

target: (List of Save) A `load()` variable, rule input, process variable, or node input in which to save the value. `with()` variables are reset each evaluation and cannot be used as save targets.

value: (Any Type) The value to save. The SAIL component's updated value can be accessed using the special variable `save!value`.

Returns

Save

Notes

`a!save()` can be called multiple times for a given component by passing them in a list to the component's *saveInto* parameter.

The *target* and *value* parameters are not evaluated until the user interacts with the component.

The variable `save!value` is only available in `a!save()`'s *value* parameter. It cannot be used in the *target* parameter or outside `a!save`.

If the component's updated value should be saved directly into a variable without modification, the a!save() function is not necessary (see first example below).

Examples

Copy and paste an example into the Interface Designer to see how this works.

Saving a variable without modification (doesn't need a!save function)

```
=load(
  local!text,
  a!textField(
    value: local!text,
    saveInto: local!text
  )
)
```

Upper-casing the typed text

```
=load(
 local!text,
 a!textField(
  value: local!text,
  saveInto: a!save(local!text, upper(save!value))
 )
)
```

Upper-casing and appending to the typed text

```
=load(
 local!text,
 a!textField(
  value: local!text,
  saveInto: a!save(local!text, "You just typed: " & upper(save!value))
 )
)
```

Upper-casing and appending to the typed text in one variable, saving unchanged to another

```
=load(
 local!modifiedText,
 local!unmodifiedText,
 a!textField(
  instructions: local!modifiedtext,
  value: local!unmodifiedText,
  saveInto: {
   a!save(local!modifiedtext, "You just typed: " & upper(save!value)),
   local!unmodifiedText
  }
 )
)
```

Modifying two variables based on the same typed text

```
=load(
 local!upperCaseText,
 local!appendedText,
 a!textField(
  value: local!upperCaseText,
  instructions: local!appendedText,
  saveInto: {
   a!save(local!upperCaseText, upper(save!value)),
   a!save(local!appendedText, "You just typed: " & save!value)
  }
 )
)
```

Modifying two variables, one based on the typed text and one with an arbitrary value=

```
=load(
 local!upperCaseText,
 local!isModified: false,
 a!textField(
  value: local!upperCaseText,
  instructions: if(local!isModified, "Modified", ""),
  saveInto: {
   a!save(local!upperCaseText, upper(save!value)),
   a!save(local!isModified, true)
  }
 )
)
```

Modifying two variables, one based on the typed text and one conditionally

```
=load(
 local!text,
 local!longText: "Short Text",
 local!shortText: "Long Text",
 a!textField(
  label: local!shortText,
  instructions: local!longText,
  value: local!text,
```

```
    saveInto: {
      local!text,
      a!save(
        if(
          len(local!text) > 5,
          local!longText,
          local!shortText
        ),
        save!value
      )
    }
  )
)
```

with()

Lets you define local variables within a function and evaluate the expression with the new variables.

Syntax

with(*localVar1*, ..., [*localVarN*], *expression*)

localVar1: (Any Type) The local variable to use when evaluating the given expression and defined using `with(local!a,..., expression)` or `with(local!a:10, ..., expression)`.

localVarN: (Any Type) Any additional local variables, as needed.

expression: (Any Type) The expression to evaluate using the local variables' values.

Returns

Any Type

Notes

This function differs from the `load()` function because it recalculates the local variable values when the expression is re-evaluated.

You cannot use this function within the Web Content Channel or a looping function or else an error occurs.

A local variable may be defined with or without a value, and the value may be simple or complex. When a value is not defined, it's assigned a null value.

When you don't specify the `local!` domain, the system first matches your variables with rules or constants with the same name, then looks for local variables with the name. Appian recommends that you always use the `local!` domain when referring to local variables.

Local variables are not assigned a type. At runtime, the type of the variable will be based on the assigned value. For example, in `with(local!myvar:2, ...)`, `myvar` is of type `Integer`.

The type returned by the `with()` function will be that of the given expression.

A local variable may reference a previously defined local variable. They are evaluated in the given order.

The local variables are only available in the evaluation of the *expression*.

The *expression* can include other variables available in its context, such as process variables and rule inputs. For example, `with(local!myvar: 1, local!myvar + ri!myruleinput)`.

If an expression requires multiple evaluations of a complex value, you can use the `with()` function to define the value as a local variable, so it's only evaluated once.

See also: Grid Tutorial

When a local variable uses the dot notation or brackets, a runtime error message will display. If the field name must contain special characters, enclose the name in single quotes.

Examples

Copy and paste an example into the Test Rules interface to see how this works.

Using Local Variables

`with(local!a:10, local!b:20, local!a+local!b)` returns `30`

`with(local!a:10, local!b:20, local!c:30, local!a+local!b+local!c)` returns `60`

`with(local!a, isnull(local!a))` returns `true`

`with('local!a-name':10, local!b:20, 'local!a-name'+local!b)` returns `30`

Using Local Variables with a Complex Value

`with(local!a:getPersonForId(ri!id), local!a.firstName & " " & local!a.lastName)` returns `John Smith`

Evaluating an Expression in the Context of a Process or Rule

When an expression is evaluated in the context of a process, the expression has access to all PV values for that process. Similarly, if the expression is used in a rule, the expression has access to all rule inputs.

`with(local!a:pp!initiator, getDisplayName(local!a) & pv!b)` returns `John Smith123` where (pv!b=123)

`with(local!a:10, ri!a)` returns `100` where (ri!a=100)

Overriding Variables with the Same Identifier

If the local variable has the same domain and name (domain + name combination) as another variable in the evaluation context, the local variable is use

`with(local!a:100, with(local!a:20, local!z:10, local!a+local!z))` returns `30`

`with(local!a:100, local!b:1, with(local!a:20, local!z:10, local!a+local!b+local!z))` returns `31`

`with(ri!a:10, 2*ri!a)` returns `20`

Defining a Local Variable by Referencing another Local Variable

`with(local!a:10, local!b:local!a*2, local!a+local!b)` returns `30`

`with(local!a:10, with(local!b:local!a, local!c:20, local!a+local!b+local!c))` returns `40`

`with(local!a:local!b, local!b:10, local!a+local!b)` returns `error` because "a" references "b" which is not available to "a"