

Ruby - Bug #21375

Set[] does not call #initialize

05/27/2025 05:30 AM - Ethan (Ethan -)

Status: Priority: Assignee: Target version: ruby -v:	Open Normal ruby 3.5.0dev (2025-05-26T17:42:35Z master 909a0daab6) +PRISM [x86_64-darwin22]	Backport:	3.2: UNKNOWN, 3.3: UNKNOWN, 3.4: UNKNOWN
Description I have a subclass of Set that overrides #initialize. Following #21216 , .new does call #initialize but .[] does not. <pre>class MySet < Set def initialize(enum = nil) compare_by_identity super end end</pre> <pre>MySet.new.compare_by_identity? # => true MySet[].compare_by_identity? # => false</pre>			
Related issues: Related to Ruby - Bug #21396: Set#initialize should call Set#add on items pas... <div>Open</div>			

History

#1 - 05/27/2025 06:09 AM - nobu (Nobuyoshi Nakada)

I'm not sure if it is intentional or not.
If unintentional, this patch may fix it.

```
diff --git a/set.c b/set.c
index 8676c62cd35..c41781c446f 100644
--- a/set.c
+++ b/set.c
@@ -409,13 +409,7 @@ static VALUE
set_s_create(int argc, VALUE *argv, VALUE klass)
{
    VALUE set = set_alloc_with_size(klass, argc);
-   set_table *table = RSET_TABLE(set);
-   int i;
-
-   for (i=0; i < argc; i++) {
-       set_table_insert_wb(table, set, argv[i], NULL);
-   }
+   rb_obj_call_init(set, argc, argv);
    return set;
}
```

#2 - 05/27/2025 02:43 PM - jeremyevans0 (Jeremy Evans)

nobu (Nobuyoshi Nakada) wrote in [#note-1](#):

I'm not sure if it is intentional or not.

It was intentional when I developed core Set. Set[] takes different arguments than Set#initialize.

Array[] does not call Array#initialize, and Hash[] does not call Hash#initialize, so there is precedent for the core collection classes to operate this way.

Set[] was not documented as calling Set#initialize previously, and there were no tests or specs for it. Relying on Set[] calling Set#initialize was relying on undefined behavior.

If unintentional, this patch may fix it.

```
diff --git a/set.c b/set.c
index 8676c62cd35..c41781c446f 100644
--- a/set.c
+++ b/set.c
@@ -409,13 +409,7 @@ static VALUE
set_s_create(int argc, VALUE *argv, VALUE klass)
{
    VALUE set = set_alloc_with_size(klass, argc);
-   set_table *table = RSET_TABLE(set);
-   int i;
-
-   for (i=0; i < argc; i++) {
-       set_table_insert_wb(table, set, argv[i], NULL);
-   }
+   rb_obj_call_init(set, argc, argv);
    return set;
}
```

I'm not sure whether this is correct. #initialize arguments are enumerables of elements, [] arguments are elements. I think if we wanted to call #initialize, we would have to allocate an array for the elements, and pass that, which would reduce performance.

#3 - 05/28/2025 03:56 AM - Ethan (Ethan -)

It seems like a regression to me. I mean, it broke my code - maybe I'm alone, I can't say whether other people override #initialize and expect Set[] to call it, but it seems like a reasonable assumption to make, particularly since it always has done that. I wasn't aware that Array[]/Hash[] don't call initialize and I find that counterintuitive, and would advocate against changing other classes/constructors to do that.

Set[] was not documented as calling Set#initialize previously

My feeling is that the base expectation is that #initialize is called on new objects. I know constructors can allocate and do things without #initialize, but I think that is quite rare. Maybe I'm wrong, I just learned Array[] and Hash[] do that, but I've almost never seen it.

I'll also note that the effects of not calling #initialize are easy to miss, since the set will still be a perfectly functional set, but will have lost whatever checks or changes are intended on initialization. At least in my case this was not straightforward to trace back to this change, it took me a while in the debugger to figure out that some of my sets were unexpectedly no longer compare_by_identity as my #initialize sets.

Having figured that out it is of course trivial for me to override MySet[], but I'd rather Set behave like it used to, especially if that would save other people having to debug subtle bugs and fix.

#4 - 06/01/2025 08:03 PM - Eregon (Benoit Daloze)

IMO this is unnecessarily breaking compatibility.

If we don't want to support subclasses of Set properly as it used to work (and calling initialize is I think a totally reasonable assumption, especially for any library which used to be written in Ruby), then I think we should forbid subclassing Set entirely to make it clear.

#5 - 06/04/2025 07:41 PM - jeremyevans0 (Jeremy Evans)

- Related to Bug #21396: Set#initialize should call Set#add on items passed in added