

# DRAFT: Clarity roll backs

Part 5

Phillip G. Bradford\*

January 3, 2024

## Abstract

If a Clarity function returns (`err ...`), then it rolls back the current computation. This is an important feature for smart contracts. Particularly, if an error occurs in a contract evaluation, then the contract evaluation is rolled back.

## 1 A basic example

Clarity contracts have their own scope. That is, the variables defined in a *.clar* file have scope limited to the contract. However, while a function is run in Clarity the updates of the variables it modifies can be reverted back to their previous values. This is done when the function returns (`err ...`).

Listing 1: A contract with roll-back

---

```
(define-data-var my-var int -1)

(define-public (undo-my-var)
  (begin
    (var-set my-var 123)
    (err "reverse_ var-set") ;; Comment this out!
  )
)

(define-public (get-my-var)
  (ok (var-get my-var)))

(define-public (set-my-var (x int))
  (ok (var-set my-var x)))
```

---

\*phillip.bradford@uconn.edu, phillip.g.bradford@gmail.com, UNIVERSITY OF CONNECTICUT, SCHOOL OF COMPUTING, STAMFORD, CT USA

---

Listing 2: A roll-back

---

```
>> (contract-call? .c1 get-my-var)
(ok -1)
>> (contract-call? .c1 set-my-var 99)
(ok true)
>> (contract-call? .c1 get-my-var)
(ok 99)
>> (contract-call? .c1 undo-my-var)
(err "reverse var-set")
>> (contract-call? .c1 get-my-var)
(ok 99)
```

---

Listing 1 can also be done using a private-function.

---

Listing 3: A contract with roll-back

---

```
(define-private (roll-back (x int)) ;; PRIVATE
  (begin
    (var-set my-var x)
    (err "rollback")))
)

(define-data-var my-var int -1)

(define-public (undo-my-var)
  (begin
    (var-set my-var 123)
    (var-set save-my-var (var-get my-var))
    (roll-back 10) ;; !!
  )
)

;;
;; get-my-var and set-my-var are the same as before!
```

---

---

Listing 4: A roll-back

---

```
>> (contract-call? .c1 get-my-var)
(ok -1)
>> (contract-call? .c1 set-my-var 501)
(ok true)
>> (contract-call? .c1 get-my-var)
(ok 501)
>> (contract-call? .c1 undo-my-var)
(err "rollback")
>> (contract-call? .c1 get-my-var)
(ok 501)
```

---

The example in Listing 4 shows that when the `define-private` roll-back function returns (`err ...`), then the containing public function `undo-my-var` rolls back.

---

Listing 5: An outer-most contract containing a roll-back

---

```
(define-public (outer-most)
  (begin
    (var-set my-var 1729)
    (ok (undo-my-var))
  )
)
```

---

Adding the public function `outer-most` from Listing 5 to the contract in Listing 3 shows that the most nested defined contract is the only roll-back.

---

Listing 6: A nested roll-back

---

```
>> (contract-call? .c1 outer-most)
(ok (err "rollback"))
>> (contract-call? .c1 get-my-var)
(ok 1729)
```

---

The next text-book example is from the Clarity Cookbook [2, 1].

---

Listing 7: A contract with roll-back from [2]

---

```
(define-data-var even-values uint u0)

(define-public (count-even (number uint))
  (begin
    ;; increment the "event-values" variable by one.
    (var-set even-values (+ (var-get even-values) u1))

    ;; check if the input number is even (number mod 2
    equals 0).
    (if (is-eq (mod number u2) u0)
        (ok "the_number_is_even")
        (err "the_number_is_uneven"))
  )
)

(define-read-only (get-even)
  (ok (var-get even-values)))
```

---

The textbook example in Listing 7 is another example of a roll-back.

## 2 Exercises

1. How can we implement roll-backs in a stack-based language?
2. Even recursive programming languages do not *need* a stack. Though, conceptually stacks are very good for implementing recursion.

Describe how to implement functions calling themselves, but only a fixed number of times. How might we implement roll-backs in this case?

## References

- [1] <https://docs.stacks.co/docs/cookbook/creating-an-ft>, 2023-04-11.
- [2] <https://book.clarity-lang.org/> Clarity of Mind book, 2023-06-11.
- [3] Harold Abelson, Gerald Jay Sussman, with Julie Sussman: *Structure and Interpretation of Computer Programs*, Second Edition, MIT Press, 1996.
- [4] Daniel P. Friedman and Matthias Felleisen: *The Little Schemer*, Fourth edition, MIT Press, 1996.
- [5] Kenny Rogers: *Building an NFT with Stacks and Clarity*, <https://blog.developerdao.com/building-an-nft-with-stacks-and-clarity>, 2022-09-01.
- [6] Kenny Roger, Joe Bender: Stacks developer workshop: *Web3 for Bitcoin: The What, Why, and How of Building on Stacks*. Web3 for Bitcoin. Wed, Jun 29, 2022.