



**ANASTASIA LABS**

## **Proof of Achievement - Milestone 1**

Payment Subscription Smart Contract

**Project Number** 1100025  
**Project Manager** Jonathan Rodriguez  
**Project Name:** Anastasia Labs X Maestro - Plug 'n Play 2.0  
**URL:** Catalyst Proposal

# Contents

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Contract Functionality .....</b>	<b>2</b>
2.1. Service Contract .....	2
2.2. Account Contract .....	2
2.3. Payment Contract .....	2
<b>3. Effortlessly Manage Recurring Payments .....</b>	<b>3</b>
3.1. Test Suite Details .....	3
3.2. Test Execution Results .....	3
<b>4. Detailed Test Case Scenarios .....</b>	<b>4</b>
4.1. Initiating a Subscription (succeed_initiate_subscription) .....	5
4.2. Terminate Subscription (succeed_terminate_subscription) .....	6
4.3. Extend Subscription (succeed_extend_subscription) .....	7
4.4. Unsubscribe (succeed_unsubscribe) .....	8
4.5. Merchant Withdrawing Fees (succeed_merchant_withdraw) .....	9
4.6. Subscriber Withdrawing Fees (succeed_subscriber_withdraw) .....	10
<b>5. User Workflow for Managing Recurring Payments .....</b>	<b>11</b>
<b>6. Conclusion .....</b>	<b>12</b>

# Payment Subscription Smart Contract

## 1. Introduction

The Payment Subscription Smart Contract project aims to develop a robust and user-friendly system for managing recurring payments on the Cardano blockchain. This contract enables users to effortlessly set up, manage, and cancel recurring payments directly from their wallets, providing a seamless experience for both subscribers and merchants.

Key features include:

- Initiating subscriptions with customizable terms
- Extending or terminating subscriptions
- Automated recurring payments
- Secure withdrawal of funds for both merchants and subscribers
- Seamless integration with popular wallet applications

This report demonstrates our progress in implementing the contract and meeting the acceptance criteria, focusing on effortless management of recurring payments and integration with wallets.

## **2. Contract Functionality**

The Payment Subscription Smart Contract consists of three main validators:

### **2.1. Service Contract**

A multi-validator responsible for:

- Creating an initial service by minting a single CIP-68 compliant Service NFT asset
- Sending the Service NFT to the user and the reference NFT to the spending endpoint
- Updating the metadata for the user
- Deleting the service by setting it to inactive.

### **2.2. Account Contract**

A multi-validator responsible for:

- Creating an account for the user by minting a CIP-68 compliant Account NFT asset
- Sending the Account NFT to the user and the reference NFT to the spending endpoint
- Updating the metadata for the account
- Deleting the user account by burning the Account NFT

### **2.3. Payment Contract**

This is the core validator and is responsible for:

- Holding prepaid subscription fees for a service
- Renewing a subscription
- Unsubscribing from a service
- Withdrawing subscription fees

The contract incorporates a linear vesting mechanism to gradually release subscription fees to the merchant over the subscription period.

### 3. Effortlessly Manage Recurring Payments

Our smart contract enables users to easily manage their recurring payments through a series of intuitive operations. We've implemented and tested various scenarios to ensure a smooth user experience.

#### 3.1. Test Suite Details

We've developed a comprehensive test suite consisting of thirteen critical test cases, to validate the contract's functionality. These tests cover all aspects of subscription management, from initiation to termination and fund withdrawal.

Here's an overview of the test execution results:

#### 3.2. Test Execution Results

```
Testing ...

payment_subscription/tests/account_multi_validator
PASS [mem: 347249, cpu: 137903361] succeed_create_account
PASS [mem: 206854, cpu: 79875639] succeed_delete_account
PASS [mem: 477264, cpu: 179987133] succeed_update_account
PASS [mem: 289679, cpu: 112928610] succeed_remove_account
4 tests | 4 passed | 0 failed

payment_subscription/tests/payment_multi_validator
PASS [mem: 719170, cpu: 277679281] succeed_initiate_subscription
PASS [mem: 358259, cpu: 134424664] succeed_terminate_subscription
PASS [mem: 886708, cpu: 338587008] succeed_extend_subscription
PASS [mem: 712513, cpu: 273599426] succeed_unsubscribe
PASS [mem: 765639, cpu: 289918106] succeed_merchant_withdraw
PASS [mem: 598455, cpu: 229728929] succeed_subscriber_withdraw
6 tests | 6 passed | 0 failed

payment_subscription/tests/service_multi_validator
PASS [mem: 416801, cpu: 163250864] success_create_service
PASS [mem: 560773, cpu: 210655896] success_update_service
PASS [mem: 596384, cpu: 230611796] success_remove_service
3 tests | 3 passed | 0 failed

Summary 13 checks, 0 errors, 0 warnings
```

Figure 1: Payment Subscription Tests Overview

## 4. Detailed Test Case Scenarios

This process comprises of six checks all from the Payments Contract which are:

- succeed\_initiate\_subscription
- succeed\_terminate\_subscription
- succeed\_extend\_subscription
- succeed\_unsubscribe
- succeed\_merchant\_withdraw
- succeed\_subscriber\_withdraw

## 4.1. Initiating a Subscription (succeed\_initiate\_subscription)

### Test Case Code: `succeed_initiate_subscription`

This test includes setting up a subscription with payment, creating the required datum and redeemer, and ensuring that the subscription is correctly recorded in the contract. It involves preparing inputs and outputs for the transaction and confirming that the subscription is successfully initiated.

```

Testing ...
- payment_subscription/tests/payment_multi_validator
- PASS [mem: 9148352, cpu: 480306766] succeed_initiate_subscription
- with traces
- Tests: Initiating a New Subscription
- -----
- Step 1: Setting up the subscription
- -----
- Service Currency Symbol:
- | h'bf726c3c1491658188e6ff550c81a1c4f8f0c2e9f26a9a16f488abe'
- Account Currency Symbol:
- | h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218'
- Subscription Fee: (lowelace)
- | 1000000000
- Subscription Period: (days)
- | 30
- Penalty Fee: (lowelace)
- | 1000000
- Step 2: Creating Payment Datum
- -----
- Service NFT:
- | h'000643800136752529784c15e638da2a27f1c08c9c8b92277913a8dc40a80d4'
- Account NFT:
- | h'0000e14001a028049afdc47fb302e059459582cb00545b7c6ac22504e98492'
- Subscription Start:
- | 1000000
- Subscription End:
- | 2593000000
- Step 3: Preparing Inputs
- -----
- Account Input:
- | 121([ 121([ 121([ h'ee155ace9c4029207acba9f8c9cc0d273c81648ff1149ef368cease' ), 1 ]), 121([ 121([ 121([ h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' ), 122([ ] ) ], _ h'': 4000000 ), h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' : [ h'0000e14001a028049afdc47fb302e059459582cb00545b7c6ac22504e98492' : 1 ] ), 123([ 121([ ] ), 122([ ] ) ] )
- Service Input:
- | 121([ 121([ 121([ h'ee155ace9c4029207acba9f8c9cc0d273c81648ff1149ef368cease' ), 1 ]), 121([ 121([ 121([ h'bf726c3c1491658188e6ff550c81a1c4f8f0c2e9f26a9a16f488abe' ), 122([ ] ) ], _ h'': 4000000 ), h'bf726c3c1491658188e6ff550c81a1c4f8f0c2e9f26a9a16f488abe' : [ h'000643800136752529784c15e638da2a27f1c08c9c8b92277913a8dc40a80d4' : 1 ] ), 123([ 121([ 121([ h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' ), 100000000, 121([ h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' : [ h'0000e14001a028049afdc47fb302e059459582cb00545b7c6ac22504e98492' : 1 ] ), 123([ 121([ ] ), 122([ ] ) ] )
- Step 4: Preparing Outputs
- -----
- User Output:
- | 121([ 121([ 121([ h'ee155ace9c4029207acba9f8c9cc0d273c81648ff1149ef368cease' ), 1 ]), 121([ 121([ 121([ h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' ), 122([ ] ) ], _ h'': 1000000000 ), h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' : [ h'0000e14001a028049afdc47fb302e059459582cb00545b7c6ac22504e98492' : 1 ] ), 123([ 121([ ] ), 122([ ] ) ] )
- Payment Output:
- | 121([ 121([ 121([ h'873e4f9e41e9249118ba3ec3ff4782efc80f244f675c879f8a432' ), 122([ ] ) ], _ h'': 1000000000 ), h'873e4f9e41e9249118ba3ec3ff4782efc80f244f675c879f8a432' : [ h'015f478f1a300f82b183e9858e3ab7772578567898150708920a190aa720ea' : 1 ] ), 123([ 121([ h'000643800136752529784c15e638da2a27f1c08c9c8b92277913a8dc40a80d4' : 1 ), 123([ 121([ h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' ), 100000000, 10000000, 25921000000, 2592000000, 100000000, 10, 500000, 121([ h'f830635c7c0b573d189e9bffa464ab4f25190d296ff08d094c665a218' : 1 ] ), 123([ 121([ ] ), 122([ ] ) ] )
- Step 5: Execution Result
- -----
- Subscription Successfully Initiated!
- -----
- Test Completed!
- -----
- token_name
- | h'0000e14001a028049afdc47fb302e059459582cb00545b7c6ac22504e98492'
- token_quantity
- | 1
- token_name
- | h'015f478f1a300f82b183e9858e3ab7772578567898150708920a190aa720ea'
- token_quantity
- | 1
  
```

Figure 2: Succeed Initialize Subscription Test

This test validates the contract's ability to initiate a new subscription. It demonstrates:

- Correct setup of subscription parameters
- Proper creation of the Payment Datum
- Accurate handling of inputs and outputs
- Successful minting of the Payment NFT





### 4.3. Extend Subscription (succeed\_extend\_subscription)

Test Case Code: succeed\_extend\_subscription

[illegible]

Figure 4: Succeed Extend Subscription Test

This test demonstrates the contract's ability to extend an existing subscription, showcasing the flexibility offered to subscribers. It shows:

- Accurate calculation of the new subscription end date
- Correct fee adjustment for the extension
- Proper updating of the Payment Datum
- Successful execution of the extension transaction



## 4.5. Merchant Withdrawing Fees (succeed\_merchant\_withdraw)

Test Case Code: succeed\_merchant\_withdraw

```

Testing ...
payment_subscription/tests/payment_multi_validator
PASS [mem: 497396, cpu: 2558971473] succeed_merchant_withdraw
- with traces
  Test: Withdrawing Subscription Fees
  -----
  Step 1: Current Contract State
  -----
  Service Currency Symbol:
  h'BA726C3C1A9165B108E6FF550CB1A1C4F0DC2E9F26A9A16F48B8E'
  Payment Currency Symbol:
  h'873E4FE9E41E924911BBA3EC53FF4782EFC80F244F875CB79FBA432'
  Subscription Start:
  1000000
  Subscription End:
  259200000
  Total Subscription Fee: (lovelace)
  1000000000
  Last Claimed:
  1000000
  Current Time:
  5185000000
  Step 2: Withdrawal Calculation
  -----
  Time Elapsed: (days)
  60
  Actual Withdrawal: (lovelace)
  200000000
  Step 3: Verifying Outputs
  -----
  Merchant Output:
  121([ 121([ 121([ h'BA726C3C1A9165B108E6FF550CB1A1C4F0DC2E9F26A9A16F48B8E' ], 122([ ])), [ h'': 202000000 ], h'BA726C3C1A9165B108E6FF550CB1A1C4F0DC2E9F26A9A16F48B8E': [ h'000DE1400136752529784C15E638A2A27F81C00C9C8B92277913A8DC40A80D4': 1 ]
  ), 122([ 121([ ])), 122([ ]))
  Remaining Payment Output:
  121([ 121([ 121([ h'873E4FE9E41E924911BBA3EC53FF4782EFC80F244F875CB79FBA432' ], 122([ ])), [ h'': 900000000 ], h'873E4FE9E41E924911BBA3EC53FF4782EFC80F244F875CB79FBA432': [ h'01A402B049A4DC47E3302ED594595BDCB6D054587C6AC2504E984922B30968A': 1 ]
  ), 122([ 121([ h'000DE1400136752529784C15E638A2A27F81C00C9C8B92277913A8DC40A80D4', h'000DE1400154778F1A300F82B1B3E9858E3AB777257856789150708920A19D', 121([ h'', h'']), 800000000, 1000000, 2592000000, 1000000000, 1, 5185000000, 121([ h'', h'']), 1000
  000, 200000000 ])), 122([ ]))
  Step 4: Updating Payment Datum
  -----
  Original Last Claimed:
  1000000
  Updated Last Claimed:
  5185000000
  Step 5: Execution Result
  -----
  Withdrawal Successfully Processed!
  -----
  Test Completed!
  
```

tests | 1 passed | 0 failed  
Summary 1 check, 0 errors, 0 warnings

Figure 6: Succeed Merchant Withdraw Test

This test confirms the contract's ability to process withdrawals of subscription fees by a merchant. It shows:

- Correct calculation of withdrawable amounts based on elapsed time
- Proper distribution of funds to the merchant
- Accurate updating of the Payment Datum with new 'last claimed' time

## 4.6. Subscriber Withdrawing Fees (succeed\_subscriber\_withdraw)

Test Case Code: `succeed_subscriber_withdraw`

```

Testing ...

- payment_subscription/tests/payment_multi_validator -----
PASS [mm: 12258889, cpu: 6372695650] succeed_subscriber_withdraw
  • with traces
    Test: Withdrawing from Inactive Service
    -----
    Step 1: Current Contract State
    -----
    Service Active Status:
    121{[]
    Payment Amount:
    1000000000
    Step 2: Withdrawal Process
    -----
    Refund Amount:
    1000000000
    Step 3: Verifying Outputs
    -----
    User Output:
    121{[_ 121{[_ 122{[_ h'FB3D635C7CB573D1B9E98FF4A64AB4F25190D29B6FD8D894C605A218' ]}, 122{[]}], [_ h'': { h'': 1100000000 }, h'FB3D635C7CB573D1B9E98FF4A64AB4F25190D29B6FD8D894C605A218': { h'000DE140015F47F8F1A3D0
FB2B1B3E9858E3AB77725785678981507D8920A190': 1 } }, 123{[_ 121{[]}], 122{[]}]
    Payment Output:
    121{[_ 121{[_ 122{[_ h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432' ]}, 122{[]}], [_ h'': { h'': 1000000000 }, h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432': { h'01A4028049A9FDC47FB302ED
59459582CB0D054587C6AC22504E984922830968A': 1 } }, 123{[_ 121{[_ h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432' ]}, 122{[]}], [_ h'': { h'': 1000000000 }, h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A4
1000000000, 1000000, 2593000000, 2592000000, 100000000, 1, 1000000, 121{[_ h'', h'']}, 1000000, 200000000}], 122{[]}]
    Step 4: Validating Transaction
    -----
    Transaction Inputs:
    121{[_ 121{[_ 121{[_ h'E155ACE9C48D3D3874C86A4F8C9CDD0273C8168FF1140EF36CEAE' ]}, 122{[]}], 121{[_ 121{[_ h'FB3D635C7CB573D1B9E98FF4A64AB4F25190D29B6FD8D894C605A218' ]}, 122{[]}], [_ h'': { h'': 400000000 },
h'FB3D635C7CB573D1B9E98FF4A64AB4F25190D29B6FD8D894C605A218': { h'000DE140015F47F8F1A3D0F82B1B3E9858E3AB77725785678981507D8920A190': 1 } }, 123{[_ 121{[_ h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432' ]}, 122{[]}], 121{[_ 121{[_ h'8B30A42C1E63FAFA0A5F0A4E8A
562F7A13A2ACEA008E8E81917866B89' ]}, 122{[]}], 121{[_ h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432' ]}, 122{[]}], [_ h'': { h'': 1004000000 }, h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A4
32': { h'01A4028049A9FDC47FB302ED59459582CB0D054587C6AC22504E984922830968A': 1 } }, 123{[_ 121{[_ h'000DE1400136752529784C15E638DA2A27F81C00C9C8B92277913ABDC40A86D4', h'000DE140015F47F8F1A3D0F82B1B3E9858E3AB77725785678981
507D8920A190', 121{[_ h'', h'']}, 1000000000, 1000000, 2593000000, 2592000000, 100000000, 1, 1000000, 121{[_ h'', h'']}, 1000000, 200000000}], 122{[]}]
    Transaction Outputs:
    121{[_ 121{[_ 122{[_ h'FB3D635C7CB573D1B9E98FF4A64AB4F25190D29B6FD8D894C605A218' ]}, 122{[]}], [_ h'': { h'': 1100000000 }, h'FB3D635C7CB573D1B9E98FF4A64AB4F25190D29B6FD8D894C605A218': { h'000DE140015F47F8F1A
3D0F82B1B3E9858E3AB77725785678981507D8920A190': 1 } }, 123{[_ 121{[_ h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432' ]}, 122{[]}], 121{[_ 121{[_ h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432' ]}, 122{[]}], [_ h'': { h'': 1000000000 }, h'873E4FE9E41E
924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432': { h'01A4028049A9FDC47FB302ED59459582CB0D054587C6AC22504E984922830968A': 1 } }, 123{[_ 121{[_ h'000DE1400136752529784C15E638DA2A27F81C00C9C8B92277913ABDC40A86D4', h'000DE1400
15F47F8F1A3D0F82B1B3E9858E3AB77725785678981507D8920A190', 121{[_ h'', h'']}, 1000000000, 1000000, 2593000000, 2592000000, 100000000, 1, 1000000, 121{[_ h'', h'']}, 1000000, 200000000}], 122{[]}]
    Tokens:
    { h'': { h'': 0 }, h'873E4FE9E41E924911BBA3EC53FF4782EFC8C0F244FB75C879F8A432': { h'01A4028049A9FDC47FB302ED59459582CB0D054587C6AC22504E984922830968A': -1 } }
    Step 5: Execution Result
    -----
    Withdrawal from Inactive Service Successfully Processed!
    -----
    Test Completed!

Summary 1 check, 0 errors, 0 warnings
1 tests | 1 passed | 0 failed

```

Figure 7: Succeed Subscriber Withdraw Test

This test verifies the contract's ability to process withdrawals of subscription fees by a subscriber when the service becomes inactive. It demonstrates:

- Correct identification of an inactive service
- Full refund of the subscription amount to the subscriber
- Proper burning of the Payment NFT
- Accurate updating of the Payment UTxO

## 5. User Workflow for Managing Recurring Payments

The following outlines the user workflow for managing recurring payments:

### 1. **Initiate Subscription:**

- User selects a service and subscription period
- Smart contract mints a Payment NFT and locks the subscription fee
- User receives confirmation of successful subscription

### 2. **Extend Subscription:**

- User chooses to extend their subscription
- Smart contract calculates additional fee and new end date
- User approves the extension
- Contract updates the Payment Datum with new details

### 3. **Unsubscribe:**

- User requests to end their subscription
- Contract calculates refund and penalty amounts
- User receives refund, minus any applicable penalties
- Payment NFT is burned, ending the subscription

### 4. **Merchant Withdrawal**

- Merchant can withdraw accrued fees at any time
- Contract calculates withdrawable amount based on elapsed time
- Remaining funds stay locked until the next withdrawal or end of subscription

### 5. **Subscriber Withdrawal**

- Subscriber can withdraw remaining funds if the service becomes inactive
- Contract verifies the inactive status of the service
- Full remaining subscription amount is refunded to the subscriber
- Payment NFT is burned, finalizing the withdrawal

This workflow demonstrates the ease with which users can manage their recurring payments, from initiation to termination, directly from their wallets.

## 6. Conclusion

The Payment Subscription Smart Contract demonstrates robust functionality and ease of use. Through comprehensive testing and thoughtful implementation, it effectively manages recurring payments, allowing users to initiate, extend, and terminate subscriptions efficiently.

Key achievements include:

- Successful implementation of subscription initiation, extension, and termination processes
- Accurate handling of fee calculations, including prorated refunds and penalties
- Secure management of funds through the payment contract
- Flexible service and account management through dedicated contracts

These features collectively ensure that the contract meets the needs of both service providers and subscribers, offering a secure and user-friendly solution for managing subscription-based services on the Cardano blockchain.

All documentation, including detailed test cases and explanations, is available in our [Payment Subscription Github Repo](#)