Programmatic Subscription

You are viewing the VRF v2 guide - Subscription method

To learn how to request random numbers without a subscription, see the direct funding method guide.

Security Considerations

Be sure to review your contracts with the security considerations in mind.

How you manage the subscription depends on your randomness needs. You can configure your subscriptions using the Subscription Manager , but these examples demonstrate how to create your subscription and add your consumer contracts programmatically. For these examples, the contract owns and manages the subscription. Any wallet can provide funding to those subscriptions.

You can view and monitor your subscriptions in the Subscription Manager even if you create them programmatically. Go to vrf.chain.link to open the Subscription Manager.

Modifying subscriptions and configurations

Subscription configurations do not have to be static. You can change your subscription configuration dynamically by calling the following functions using the VRFCoordinatorV2Interface :

- Change the list of approved subscription consumers with:* addConsumer(uint64 subId, address consumer).
- removeConsumer(uint64 subId, address consumer).
- Transfer the subscription ownership with:* requestSubscriptionOwnerTransfer(uint64 subId, address newOwner).
- acceptSubscriptionOwnerTransfer(uint64 subId).
- View the subscription withgetSubscription(uint64 subId).
- Cancel the subscription withcancelSubscription(uint64 subId).

To send LINK to the subscription balance, use the LINK token interface withLINKTOKEN.transferAndCall(address(COORDINATOR), amount, abi.encode(subId)). Any wallet can fund a subscription.

See the example in the Subscription manager contract section to learn how to create a contract that can change your subscription configuration.

Subscription manager contract

In this example, the contract operates as a subscription owner and can run functions to add consuming contracts to the subscription. The consuming contracts must include therequestRandomWords()function with the correct coordinator parameters and the correct subscription ID to request random values and use the subscription balance. The consuming contracts must also include thefulfillRandomWords()function to receive the random values.

Subscription owners and consumers do not have to be separate. This contract not only allows adding consumers withaddConsumer(address consumerAddress)but can also act as a consumer by running its ownrequestRandomWords()function. This example contract includes acreateNewSubscription()function in theconstructor()that creates the subscription and adds itself as a consumer automatically when you deploy it.

// SPDX-License-Identifier: MIT// An example of a consumer contract that also owns and manages the subscriptionpragmasolidity^0.8.7;import{LinkTokenInterface}from"@chainlink/contracts/src/v0.8/shared/interfaces/LinkTokenInterface.sol";import{VRFCoordinatorV2Interface}from"@chainlink/contracts/s
*** Request testnet LINK and ETH here: https://faucets.chain.link/ * Find information on LINK Token Contracts and get the latest ETH and LINK faucets here: https://docs.chain.link/docs/link-token-contracts/** *// * THIS IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION. */contractVRFv2SubscriptionManagerisVRFConsumerBaseV2{VRFCoordinatorV2Interface COORDINATOR;LinkTokenInterface LINKTOKEN;// Sepolia coordinator. For other networks,// see https://docs.chain.link/docs/vrf-contracts/#configurationsaddressvrfCoordinator=0x8103B0A8A00be2DDC778e6e7eaa21791Cd364625;// Sepolia LINK token contract. For other networks, see// https://docs.chain.link/docs/vrf-contracts/#configurationsaddresslink_token_contract=0x779877A7B0D9E8603169DdbD7836e478b4624789;// The gas lane to use, which specifies the maximum gas price to bump to.// For a list of available gas lanes on each network,// see https://docs.chain.link/docs/vrf-contracts/#configurationsbytes32keyHash=0x474e34a077df58807dbe9c96d3c009b23b3c6d0cce433e59bbf5b34f823bc56c;// A reasonable default is 100000, but this value could be different// on other networks.uint32callbackGasLimit=100000;// The default is 3, but you can set this higher.uint16requestConfirmations=3;// For this example, retrieve 2 random values in one request.// Cannot exceed VRFCoordinatorV2.MAX_NUM_WORDS.uint32numWords=2;// Storage parametersuint256[]publics_randomWords;uint256publics_requestId;uint64publics_subscriptionId;addresss_owner;constructor()VRFConsumerBaseV2(vrfCoordinator){COORDINATOR=VRFCoordinatorV2Interface(vrfCoordinator);LINKTOKEN=LinkTokenInterface(link_token_contract);s_owner=msg.sender;//Create a new subscription when you deploy the contract.createNewSubscription();}// Assumes the subscription is funded sufficiently.functionrequestRandomWords()externalonlyOwner{// Will revert if subscription is not set and funded.s_requestId=COORDINATOR.requestRandomWords(keyHash,s_subscriptionId,requestConfirmations,callbackGasLimit,numWords);}functionfulfillRandomWords(uint256/* requestId */,uint256[]memoryrandomWords)internaloverride{s_randomWords=randomWords;}// Create a new subscription when the contract is initially deployed.functioncreateNewSubscription()privateonlyOwner{s_subscriptionId=COORDINATOR.createSubscription();// Add this contract as a consumer of its own subscription.COORDINATOR.addConsumer(s_subscriptionId,address(this));}// Assumes this contract owns link.// 1000000000000000000 = 1 LINKfunctiontopUpSubscription(uint256amount)externalonlyOwner{LINKTOKEN.transferAndCall(address(COORDINATOR),amount,abi.encode(s_subscriptionId));}functionaddConsumer(addressconsum
Add a consumer contract to the subscription.COORDINATOR.addConsumer(s_subscriptionId,consumerAddress);}functionremoveConsumer(addressconsumerAddress)externalonlyOwner{// Remove a consumer contract from the subscription.COORDINATOR.removeConsumer(s_subscriptionId,consumerAddress);}functioncancelSubscription(addressreceivingWallet)externalonlyOwner{// Cancel the subscription and send the remaining LINK to a wallet address.COORDINATOR.cancelSubscription(s_subscriptionId,receivingWallet);s_subscriptionId=0;}// Transfer this contract's funds to an address.// 1000000000000000000 = 1 LINKfunctionwithdraw(uint256amount,addressto)externalonlyOwner{LINKTOKEN.transfer(to,amount);}modifieronlyOwner(){require(msg.sender==s_owner);_;}} Open in Remix What is Remix? To use this contract, compile and deploy it in Remix.

1. Open the contract in Remix .
2. Compile and deploy the contract using the Injected Provider environment. The contract includes all of the configuration variables that you need, but you can edit them if necessary. For a full list of available configuration variables, see the Supported Networks page.

This contract automatically creates a new subscription when you deploy it. Read thes_subscriptionIdvariable to find your subscription ID. You can use this value to find the subscription at vrf.chain.link . 3. In this example, thetopUpSubscription()function sends LINK from your contract to the subscription. Fund your contract with at least three testnet LINK. Alternatively, you can send LINK directly to the subscription at vrf.chain.link . Any address can provide funding to a subscription balance. If you need testnet LINK, you can get it from faucets.chain.link . 4. Run thetopUpSubscription()function to send LINK from your contract to your subscription balance. For this example, specify a value of3000000000000000000, which is equivalent to three LINK. 5. Run therequestRandomWords()function. The request might take several minutes to process. Track the pending request status at vrf.chain.link . 6. You can also add and test consumer contracts using the same programmatic subscription process:

1. Create and deploy a consumer contract that includes the following components:

2. TherequestRandomWords()function and the required variables and your subscription ID.

3. ThefulfillRandomWords()callback function.

You can use the example from the Get a Random Number guide. 2. After you deploy the consumer contract, add it to the subscription as an approved consumer using theaddConsumer()function on your subscription manager contract. Specify the address of your consumer contract. 3. On the consumer contract, run therequestRandomWords()function to request and receive random values. The request might take several minutes to process. Track the pending request status at vrf.chain.link .

The consumer contract can continue to make requests until your subscription balance runs out. The subscription manager contract must maintain sufficient balance in the subscription so that the consumers can continue to operate. 4. If you need to remove consumer contracts from the subscription, use theremoveConsumer()function. Specify the address of the consumer contract to be removed. 7. When you are done with your contracts and the subscription, run thecancelSubscription()function to close the subscription and send the remaining LINK to your wallet address. Specify the address of the receiving wallet.

Funding and requesting simultaneously

You can fund a subscription and request randomness in a single transaction. You must estimate how much the transaction might cost and determine the amount of funding to send to the subscription yourself. See the Subscription billing page to learn how to estimate request costs.

// Assumes this contract owns link// You must estimate LINK cost yourself based on the gas lane and limits.// 1_000_000_000_000_000_000 = 1 LINKfunctionfundAndRequestRandomWords(uint256amount)externalonlyOwner{LINKTOKEN.transferAndCall(address(COORDINATOR),amount,abi.encode(s_subscriptionId));// Will revert if subscription is not set and funded.s_requestId=COORDINATOR.requestRandomWords(keyHash,s_subscriptionId,requestConfirmations,callbackGasLimit,numWords);} Add this function to your contracts if you need to provide funding simultaneously with your requests. ThetransferAndCall()function sends LINK from your contract to the subscription, and therequestRandomWords()function requests the random words. Your contract still needs thefulfillRandomWords()callback function to receive the random values.