Using a Smart Contract

Using a Smart contract

You can create a task that uses Web3 Function from your smart contract as well.

If your project involves complex interactions and you need the task creation to be a part of an on-chain transaction, you would directly interact with a smart contract.

Web3 Function secrets are not available for smart contract created tasks. To create a Web3 Function task with your smart contract, you can inheritAutomateTaskCreator which has helper functions to easily create your task.

- Pass theModule.PROXY
- &Module.WEB3_FUNCTION
- as modules inModuleData
-

```

Copy ModuleDatamemorymoduleData=ModuleData({ modules:newModule, args:newbytes });

moduleData.modules[0]=Module.PROXY; moduleData.modules[1]=Module.WEB3_FUNCTION;

```

- Use_web3FunctionModuleArg
- to encode arguments forWEB3_FUNCTION
- module.
-

```

Copy function_web3FunctionModuleArg( stringmemory_web3FunctionHash,// IPFS CID of deployed web3Function bytescalldata_web3FunctionArgsHex// Abi encoded web3 function arguments )

```

Here is how you can encode your Web3Function arguments to getweb3FunctionArgsHex.

In this example, the Web3Function has 2 arguments,counterW3fAddress &count .

```

schema.json Copy { "web3FunctionVersion":"2.0.0", "runtime":"js-1.0", "memory":128, "timeout":30, "userArgs":{ "counterW3fAddress":"string", "count":"number" } }

```

In your contract, you would encode the arguments according to the sequence defined inschema.json .

```

Copy function_getWeb3FunctionArgsHex( addresscounterW3fAddress, uint256count ) internal pure returns(bytesmemoryweb3FunctionArgsHex) { web3FunctionArgsHex=abi.encode(counterW3fAddress,count) }

```

The full code can be foundhere .

```

Copy functioncreateTask( stringmemory_web3FunctionHash, bytescalldata_web3FunctionArgsHex )external{ require(taskId==bytes32(""),"Already started task");

bytesmemoryexecData=abi.encodeCall(this.increaseCount,(1));

ModuleDatamemorymoduleData=ModuleData({ modules:newModule, args:newbytes }); moduleData.modules[0]=Module.PROXY; moduleData.modules[1]=Module.WEB3_FUNCTION;

moduleData.args[0]=_proxyModuleArg(); moduleData.args[1]=_web3FunctionModuleArg( _web3FunctionHash,
```

_web3FunctionArgsHex);

bytes32id=_createTask(address(this), execData, moduleData, address(0));

taskId=id; emitCounterTaskCreated(id); }

```

Additional Info

Tasks created via this route cannot be named Smart Contracts can also create and cancel tasks.

You can find a list of example smart contractshere .

Here are the functions exposed byAutomateTaskCreator which you can use when setting up your smart contract.

_createTask()

Interacts and creates a task on the Gelato Automate smart contract.

```

Copy function_createTask(addressexecAddress, bytesmemoryexecDataOrSelector, ModuleDatamemorymoduleData, addressfeeToken)internalreturns(bytes32taskId);

```

- execAddress
- 
  - Address of the contract which Gelato will call.
- execDataOrSelector
- 
  - Signature of function which Gelato will call / execution data (If Resolver Module is not used. More about modules below)
- moduleData
- 
  - Modules that are enabled for the task. (More about ModuleData below)
- feeToken
- 
  - Useaddress(0)
- if using Gelato 1balance. Use 0xeeeeee... for ETH or native tokens.
- 

ModuleData

```

Copy structModuleData{ Module[] modules; bytes[] args; }

```

Modules are conditions / specifications about your task. These are the current available Modules.

```

Copy enumModule{ RESOLVER, PROXY, SINGLE_EXEC, WEB3_FUNCTION, TRIGGER }

```

- RESOLVER
- 
  - Define dynamic conditions and execution data.
- TIME
- 
  - Repeated execution at a specific time and interval.
- PROXY
- 
  - Your function will be called by a dedicatedmsg.sender
- .Dedicated msg.sender
- SINGLE_EXEC
- 
  - Task is cancelled after one execution.
```

- WEB3_FUNCTION
  - Define a Typescript function to get off-chain execution data.
- TRIGGER
  - Define your execution trigger (Time interval, Event, every block, ...)

Each Module would require additional arguments which is an encoded data.

IncludingModule.Proxy inmoduleData is mandatory, otherwise task creation will fail. You can use these helper functions to get the arguments for each Module.

```

Copy function_resolverModuleArg(address_resolverAddress,bytesmemory_resolverData)

function_proxyModuleArg()

function_singleExecModuleArg()

function_timeTriggerModuleArg(uint128_start,uint128_interval)

function_cronTriggerModuleArg(stringmemory_expression)

function_blockTriggerModuleArg()

function_eventTriggerModuleArg( address_address, bytes32[][]memory_topics, uint256_blockConfirmations
)internalpurereturns(bytesmemory)

```

CraftingModuleData will look like this if we want to create a task which utiliseRESOLVER ,PROXY &SINGLE_EXEC Module.

```

Copy ModuleDatamemorymoduleData=ModuleData({ modules:newModule, args:newbytes });

moduleData.modules[0]=Module.RESOLVER; moduleData.modules[1]=Module.PROXY;
moduleData.modules[2]=Module.SINGLE_EXEC

moduleData.args[0]=_resolverModuleArg( address(this), abi.encodeCall(this.checker,()) );
moduleData.args[1]=_proxyModuleArg(); moduleData.args[2]=_singleExecModuleArg();

```

Module[] must follow the orderRESOLVER ,PROXY ,SINGLE_EXEC, WEB3_FUNCTION, TRIGGER

_cancelTask()

Cancels a task owned by the smart contract.

```

Copy function_cancelTask(bytes32_taskId)internal

```

onlyDedicatedMsgSender

Function modifier to restrictmsg.sender to only task executions created bytaskCreator (defined in constructor). Learn more about it atSecurity Considerations

```

Copy modifieronlyDedicatedMsgSender() { require(msg.sender==dedicatedMsgSender,"Only dedicated msg.sender"); _; }

```

_depositFunds1Balance()

```
```

Copy functionfunction_depositFunds1Balance(uint256_amount, address_token, address_sponsor)

```

Deposit funds into the Gelato 1balance contract.

The_depositFunds1Balance method is only available on Polygon

Single Execution Task

If you want to have Gelato call your function only once. If so, you can IncludeSingleExec module inModuleData.modules . Check out the full codehere .

```

Copy ModuleDatamemorymoduleData=ModuleData({ modules:newModule, args:newbytes });

moduleData.modules[0]=Module.PROXY; moduleData.modules[1]=Module.SINGLE_EXEC;

moduleData.args[0]=_proxyModuleArg(); moduleData.args[1]=_singleExecModuleArg();

bytes32id=_createTask(address(this), execData, moduleData, address(0));

```

Previous Using the UI Next Using the Automate SDK Last updated3 months ago On this page *Using a Smart contract * Additional Info * _createTask() * ModuleData * _cancelTask() * onlyDedicatedMsgSender * _depositFunds1Balance() * Single Execution Task
```