Notify (older version)

Client library for a full notification UI experience. Powered by our Mempool API.

Notify is now part of web3-onboard

We've upgraded Notify with a modern look & feel and added it to web3-onboard. You can access the latest version onhttps://onboard.blocknative.com/ as part of thecore package . ?

To see the notify.js library in action, checkout thecode sandboxes at the end of this documentation or see the latest on the open source demo site athttps://reactdemo.blocknative.com/ .

Notify.js uses Notify API, which has rate limits. Please seeRate Limits for more details.

Quickstart

It should take less than 5 minutes to get going with Blocknative Notify.

Create a Blocknative Account

Setup your Blocknative account athttps://explorer.blocknative.com/account?signup=true using an email address. You will receive an email to confirm your account. You can then go to your account page athttps://explorer.blocknative.com/account .

Create an API Key

On your Account page, create a new API key with your choice of name using the Add new API Key button on the top right. Blocknative will create a default API key for you but it is recommended you create your own as API keys segment analytics data. Consider using different API keys for development, staging, and production releases.

?

Install the widget using npm

```

Copy npm install bnc-notify

```

Initialize the Library

Use the minimum required config. There are more options available in oudocumentation .

```

Copy importNotifyfrom"bnc-notify"

varnotify=Notify({ dappId:apiKey,// [String] The API key created by step one above networkId:networkId// [Integer] The Ethereum network ID your Dapp uses. });

```

Track transactions

Send a transaction using web3.js and get the transaction hash while its processing. Let notify know the hash and it will track its progress through the mempool and into a block.

```

Copy // get current account constaccounts=awaitweb3.eth.getAccounts(); constaddress=accounts[0];

// send the transaction using web3.js and get the hash web3.eth.sendTransaction({ from:address, to:"0x792ec62e6840bFcCEa00c669521F678CE1236705", value:"1000000"}) .on('transactionHash',function(hash){ // pass the hash to notify to track it notify.hash(hash) };

```

And you are live!

?

Example Code

For an example of a basic React demo integration of both Notify.js and Onboard.js , checkout the GitHub repo:https://github.com/blocknative/onboard-notify-react

and the live demo:

https://reactdemo.blocknative.com/

Screencasts

See the above Quickstart guide in a screencast:

?

See how to add and customize advanced notifications to your Dapp:

?

Initialization

```

Copy importNotifyfrom"bnc-notify"

constoptions={ dappId:String, system:String networkId: Number, onerror:Function darkMode: Boolean,// (default: false) mobilePosition:String,// 'top', 'bottom' (default: 'top') desktopPosition:String,// 'bottomLeft', 'bottomRight', 'topLeft', 'topRight' (default: 'bottomRight') txApproveReminderTimeout:Number,// (default: 20000) txStallPendingTimeout:Number,// (default: 20000) txStallConfirmedTimeout:Number// (default: 90000) transactionHandler: Function, clientLocale:String,// (default: 'en') notifyMessages:Object }

constnotify=Notify(options)

```

dappId [required] (optional if in UI only mode)

Your unique api key that identifies your application. You can generate a dappId by visiting the Blocknative account page and create a free account.

system [default = 'ethereum']

The blockchain system that you would like to track transactions on. Currently supported systems are'bitcoin' and'ethereum' .

networkId [required] (optional if in UI only mode)

The network id that your application runs on. The following values are valid:

Ethereum

- 1
- Main Network
- 3
- Ropsten Test Network
- 4
- Rinkeby Test Network
- 5
- __ Goerli Test Network
- 42
- Kovan Test Network
- 100
- xDai POA Network
- 137
- Polygon Matic Main Network
-

onerror

A function to catch any errors that Notify does not handle internally such as event rate limit errors.

darkMode

Boolean value sets notification rendering to use light text on dark background. Default isfalse .

mobilePosition

String value sets where notifications appear on mobile devices. Value must be eithertop orbottom . Default istop .

desktopPosition

String value sets where notifications appear on desktop devices. Value must bebottomLeft ,bottomRight ,topLeft , ortopRight . Default isbottomRight .

txApproveReminderTimeout

Number value sets time to wait until approval reminder notification is presented. This notifications reminds user to confirm/approve a transaction for processing by the wallet. Default is20000 milliseconds (20 seconds).

tsStallPendingTimeout

Number value sets time to wait until stall notification is presented if transaction is not detected in thetxPool . Default is20000 milliseconds (20 seconds).

tsStallConfirmedTimeout

Number value sets time to wait until stall notification is presented if transaction is not confirmed in a block. Default is90000 milliseconds (90 seconds).

transactionHandler

The function defined for thetransactionHandler parameter will be called once for every status update forevery transaction that is associated with a watched addressor a watched transaction. This is useful as a global handler for all transactions and status updates. The callback is called with the following object:

```

Copy { emitterResult:Any,// The value that was returned from the emitter for this transaction transaction:TransactionObject// See below for details }

```

See theTransaction Object section for more info on what is included in thetransaction parameter.

clientLocale

Sets the locale for internationalization within notify. Defaults to English ('en'). See thei18n section for more details.

notifyMessages

An object that defines all of the transaction messages. Can be used to customize the default messages, and can also be used to add multiple language support for transaction notifications. See thei18n section for more details.

Notifications

hash

To get notifications for every status that occurs after sending a transaction ("post-flight"), use thehash function:

```

Copy // Ethereum // send the transaction using web3.js and get the hash web3.eth.sendTransaction(txOptions).on("transactionHash",hash=>{ // pash the hash in to notify.hash const{emitter}=notify.hash(hash) })

// Bitcoin const{emitter}=notify.hash(txid)

```

Check out theEmitter Section for details on theemitter object

account

You can register an address with Notify to display notifications for all incoming and outgoing transactions that occur on that

account by using theaccount function:

```

Copy // Etherum constaccounts=awaitprovider.enable() // register address for all transactions that occur on the user's account const{emitter}=notify.account(accounts[0])

// Bitcoin const{emitter}=notify.account(address)

```

Check out theEmitter Section for details on theemitter object

transaction

To get notifications for every status that occurs for the full transaction lifecycle ("pre-flight" and "post-flight"), use thetransaction function:

The transaction function only applies to Ethereum transactions ```

Copy const{emitter,result}=notify.transaction({ estimateGas:Function, gasPrice:Function, balance:String, txDetails:{ to:String, value:Number||String }, sendTransaction:Function, contractCall:{ methodName:String, params:Array } })

```

result is a promise that resolves with theid of the transaction or rejects with an error if there was an error initiating the transaction. Theid is useful for when you would like to initiate the transaction yourself and didn't pass asendTransaction function in totransaction . You can then send the transaction yourself, receive the hash, and then callnotify.hash(hash, id) . By passing in theid tohash you enable notify to link the preflight notifications to the post send transaction notifications, ensuring a consistent UX.

Thetransaction function takes an object as the argument which allows you to progressively opt in to which checks and corresponding notifications that you would like during the "pre-flight" phase of the transaction lifecycle. Below is a list of the parameters, what is expected for each of those parameters, and the corresponding notifications you will get by including those parameters:

estimateGas - [balance check notification]

TheestimateGas function must return aPromise that resolves with aString that is the gas estimate inwei . This function can vary depending on whether it is a contract transaction or just a regular send transaction call.

gasPrice - [balance check notification]

ThegasPrice function must return aPromise that resolves with aString that is the gas price inwei .

balance - [balance check notification]

The users' balance inwei

txDetails - [repeat transaction check notification]

ThetxDetails object includes ato parameter which is the address the transaction is being sent to and thevalue parameter, which is the value of the transaction.

contractCall - [repeat transaction check notification]

ThecontractCall object includes amethodName parameter which is the name of the method on the contract being called and theparams parameter is the parameters that the contract method was called with.

sendTransaction - [send transaction error, approval reminder, and stall notifications]

ThesendTransaction function must return aPromise that resolves with aString that is the transaction hash.

notification

You may want to trigger a notification for a custom event that may not be related to a transaction. You can use thenotification function to do that:

```

Copy constnotificationObject={ eventCode:"dbUpdate" type:"pending", message:"Updating the database with your information" }

const{update,dismiss}=notify.notification(notificationObject)

//.... somewhere else in your code

if(dbUpdated) { update({ eventCode:"dbUpdateComplete" type:"success", message:"Your info is up to date!" }) }else{ update({ eventCode:"dbUpdateSlow" type:"pending", message:"Database update is taking longer than usual, hang in there!" }) }

```

Thenotification function is called with a notification object with the following parameters:

- eventCode
- : a string which is used to keep track of that event for your analytics dashboard
- type
- : a string that defines the style - ['hint' (gray), 'pending' (yellow), 'success' (green), 'error' (red)]
- message
- : a message string that is displayed on the notification
- autoDismiss
- : a number in milliseconds before the notification auto dismisses or0
- for no auto dismissal.success
- andhint
- types default to4000
- onclick
- : a function to call when the notification is clicked by the user
-

Returned from the notification function is an object that has two functions defined on it:

- update
- : a function that can be called with a new notification object to replace the original notification with
- dismiss
- : a function that can be called to dismiss the notification manually
-

unsubscribe

You may want to unsubscribe from an account address or transaction hash that you are watching and stop receiving notifications:

```

Copy // unsubscribe from watched address notify.unsubscribe(address)

// unsubscribe from watched transaction notify.unsubscribe(hash|txid)

```

config

Some configuration options can be set or updated after the library is initialized:

```

Copy notify.config({ networkId:String, system:String, darkMode:Boolean,// (default: false) mobilePosition:String,// 'top', 'bottom' (default: 'top') desktopPosition:String,// 'bottomLeft', 'bottomRight', 'topLeft', 'topRight' (default: 'bottomRight') txApproveReminderTimeout:Number,// (default: 20000) txStallPendingTimeout:Number,// (default: 20000) txStallConfirmedTimeout:Number,// (default: 90000) clientLocale:String,// (default: 'en') notifyMessages:Object })

```

If a newnetworkId orsystem is passed to theconfig function, then the current WebSocket connection will automatically be closed and a new connection will be established.

Emitter

Theemitter object returned is used to listen for transaction events:

```

Copy emitter.on("txRepeat",transaction=>{ // return false for no notification // no return (or return undefined) to show default notification // return custom notification object to show custom notifications, can return all or one of the following parameters:

return{ type:String,// ['hint' (gray), 'pending' (yellow), 'success' (green), 'error' (red)] message:String,// The message you would like to display autoDismiss:Number// The number of milliseconds before the notification automatically hides or 0 for no autodismiss onclick: Function// A function to call when the notification is clicked link: String// a url string that will make the notification a link when clicked on } })

// You can also use the all event to register a listener for all events for that transaction. The all listener will only be called if there isn't a listener defined for the particular event:

emitter.on("all",transaction=>{ // called on every event that doesn't have a listener defined on this transaction })

```

Linking to a Block Explorer

You may want to link to a block explorer (such as Etherscan ) on every transaction notification. To do this, a link parameter can be added to the notification object returned from the emitter callback specifying the url:

```

Copy functionaddEtherscan(transaction) { return{ link https://etherscan.io/tx{transaction.hash} } }

// then on each transaction... const{emitter}=notify.hash(hash) emitter.on('all',addEtherscan)

```

Event Codes

The following event codes are valid events to listen to on the transaction emitter:

Client Side Generated Events (when using transaction function):

- txRequest
- : Transaction has been initiated and is waiting approval from the user
- txRepeat
- : Transaction has the same value and to address, so may be a repeat transaction
- txAwaitingApproval
- : A previous transaction is awaiting approval
- txConfirmReminder
- : Transaction has not been confirmed after timeout
- txStallPending
- : Transaction has not been received in the mempool after timeout
- txStallConfirmed
- : Transaction has been in the mempool for longer than the timeout
- txError
- : An error has occurred initiating the transaction
- txSendFail
- : The user rejected the transaction
- txUnderpriced
- : The gas was set too low for the transaction to complete
- txSent
- : Transaction has been sent to the network
-

Server Side (Notify SDK) Generated Events

- txPool
- : Transaction was detected in the "pending" area of the mempool and is eligible for inclusion in a block
- txStuck
- : Transaction was detected in the "queued" area of the mempool and is not eligible for inclusion in a block
- txConfirmed
- : Transaction has been confirmed
- txFailed
- : Transaction has failed
- txSpeedUp
- : A new transaction has been submitted with the same nonce and a higher gas price, replacing the original transaction
- txCancel
- : A new transaction has been submitted with the same nonce, a higher gas price, a value of zero and sent to an external address (not a contract)
- txDropped
- : Transaction was dropped from the mempool without being added to a block.NOTE: Transaction could reappear in
```

mempool.

-

Local Networks

When using a local network (such as Ganache), Notify has no way of seeingpending andconfirmed statuses for transactions on that network andwill not display any notifications . The emitter will also not emit events for those statuses. Notify will set the network tolocal if anetworkId is passed in during initialization that is not one of thesupported networks

Transaction Object

The callback that is registered for events on the emitter will be called with the following transaction object:

```

Copy { status:String,// current status of the transaction hash:String,// transaction hash (ethereum) txid:String,// transaction id (bitcoin) to:String,// the address the transaction is being sent to from:String,// the address the transaction is being sent from gas:Number,// the gas in wei gasPrice:String,// the gasPrice in wei gasPriceGwei:Number,// the gasPrice in gwei nonce:Number,// the nonce of the transaction value:String,// the value being sent eventCode:String,// the event code for this status blockHash:String,// the hash of the block that this transaction was included in blockNumber:Number,// the block number of the block that this transaction was included in input:String,// hex string of the input data transactionIndex:Number,// same as the nonce counterParty:String,// address of the counterparty of the transaction when watching an account direction:String,// the direction of the transaction in relation to the account that is being watched ("incoming" or "outgoing") watchedAddress:String,// the address of the account being watched originalHash:String,// if a speedup or cancel status, this will be the hash of the original transaction asset:String,// the asset that was transferred contractCall:{// if transaction was a contract call otherwise undefined contractAddress:String,// the address of the contract that has been called contractType:String,// the contract type eg: ERC20, ERC721 methodName:String,// the name of the method that was called params:{ // params that the contract method was called with } }, system:String// 'bitcoin' or 'ethereum' }

```

Customizing CSS

To gain the required specificity to override the default styles, use the class name.bn-notify-custom plus the class name that you would like to override in yourCSS file. For example, if you would like to customize thefont-family of the notifications, then you would target the top levelul element that has the classbn-notify-notifications in yourCSS file:

```

Copy .bn-notify-custom.bn-notify-notifications{ font-family:Roboto; }

```

Below is a list of the main classes that you would target to customize theCSS :

class name description .bn-notify-notifications container holding all of the notifications .bn-notify.notification top level element of each notification .bn-notify-notification-info notification content container .bn-notify-notification-info-message notification message .bn-notify-notification-close close notification icon container

i18n

Notify has internationalization built in, allowing you to enable transaction notifications that match your user's locale and to give the user an option to switch locale.

clientLocale

Notify will automatically detect the browser's locale and set an internalclientLocale . If that locale is defined within thenotifyMessages object, then it will use that locale for all transaction messages. If it isn't defined, then it will fallback to the default'en' locale. You can explicitly set theclientLocale for the user by calling theconfig function or passing it in viainitializationObject .

notifyMessages

This object contains the messages for all the transaction states supported by Notify. The library has default transaction messages for the'en' clientLocale and can be extended to include transaction messages for additionalclientLocale settings. To extend the defaultnotifyMessages object, you can pass in anotifyMessages object to either theconfig function or to theinitializationObject .

Below is an example object that implements a'es' (Spanish)notifyMessages object:

```

Copy constnotifyMessages={ es:{ transaction:{ txRequest:'Su transacción está esperando que confirme', nsfFail:'No tiene fondos suficientes para completar esta transacción.', txUnderpriced: 'El precio del gas para su transacción es demasiado bajo, intente nuevamente con un precio del gas más alto', txRepeat:'Esto podría ser una transacción repetida', txAwaitingApproval: 'Tienes una transacción anterior esperando que confirmes', txConfirmReminder: 'Confirme su transacción para continuar, la ventana de transacción puede estar detrás de su navegador', txSendFail:'Rechazaste la transacción', txSent:'Su transacción ha sido enviada a la red.', txStallPending: 'Su transacción se ha estancado y no ha ingresado al grupo de transacciones', txPool:'Su transacción ha comenzado', txStallConfirmed: 'Su transacción se ha estancado y no ha sido confirmada.', txSpeedUp:'Su transacción ha sido acelerada', txCancel:'Tu transacción está siendo cancelada', txFailed:'Su transacción ha fallado', txConfirmed:'Su transacción ha tenido éxito.', txError:'Vaya, algo salió mal, por favor intente nuevamente' }, watched:{ txPool: 'su cuenta está {verb, select, receiving {recibiendo} sending {enviando}} {formattedValue} {asset} {preposition, select, from {desde} to {a}} {counterpartyShortened}', txSpeedUp: 'su cuenta está {verb, select, receiving {recibiendo} sending {enviando}} {formattedValue} {asset} {preposition, select, from {desde} to {a}} {counterpartyShortened}', txCancel: 'su cuenta está {verb, select, receiving {recibiendo} sending {enviando}} {formattedValue} {asset} {preposition, select, from {desde} to {a}} {counterpartyShortened}', txConfirmed: 'su cuenta {verb, select, received {recibió} sent {ha enviado}} con éxito {formattedValue} {asset} {preposition, select, from {de} to {a}} {counterpartyShortened}', txFailed: 'su cuenta fallado {verb, select, received {recibió} sent {ha enviado}} con éxito {formattedValue} {asset} {preposition, select, from {de} to {a}} {counterpartyShortened}' }, time:{ minutes:'min', seconds:'sec' } } }

notify.config({ notifyMessages })

```

ThenotifyMessages object above would get merged with Notify's internalnotifyMessages object, extending Notify to support both English and Spanish transaction notification messages.

You can include as many of the above parameters as you would like in each language object. If Notify gets anundefined value back from the language object, then it will fallback to the English default included in Notify.

Notify usesformat.js under the hood, which uses the ICU message syntax to format messages. Check out their guide to the ICU message syntaxhere .

For each language, you can include atransaction object, awatched object and atime object.

Thetransaction object will be referenced for notifications that have been triggered via thehash ortransaction function.

Thewatched object will be referenced for notifications that have been triggered by watching an address via theaccount function.

Thetime object will be referenced for expressing the amount of time that has passed during pending states on the notification.

Each message will be called with the following inputs that will be formatted in to a message string:

The formatted value of the transaction:formattedValue - (transaction ,watched)

The asset that is being sent or received:asset - (transaction ,watched)

The correct verb based on whether the transaction is being incoming or outgoing and whether it is confirmed or pending:verb - (watched) Possible values are:'received' ,'sent' ,'receiving' ,'sending'

The correct preposition based on the direction of the transaction:preposition - (watched) Possible values are:'from' ,'to'

A shortened address of the counter party of the transaction:counterpartyShortened - (watched)

UI Only Mode

You can optionally run Notify.js as a simple notification UI library by not including adappId in your initialization options. In this scenario, Notify.js will not initiate a WebSocket connection to the Blocknative servers and all functionality other than thenotification function will be disabled.

Rollup Builds

You may run in to an error:Uncaught TypeError: dist is not a constructor when building with Rollup.js. To fix this issue make sure that you are using@rollup/plugin-commonjs version16.0.0 and above

Get Started Today

Sign up for a free Blocknative Account athttps://explorer.blocknative.com/account?signup=true with your work email

address.

If you have any questions, connect with the team on ourdiscord

React Demo

To see a Notify integration in a simple React app you can check out ourreact-demo repo on GitHub:https://github.com/blocknative/react-demo We have thereactdemo codebase deployed so that you try Notify from an end user perspective:https://reactdemo.blocknative.com/

Previous Onboard (older version) Next Mempool Data Program Last updated11 months ago

Was this helpful?