Actions and Blinks

Solana Actions are specification-compliant APIs that return transactions on the Solana blockchain to be previewed, signed, and sent across a number of various contexts, including QR codes, buttons + widgets, and websites across the internet. Actions make it simple for developers to integrate the things you can do throughout the Solana ecosystem right into your environment, allowing you to perform blockchain transactions without needing to navigate away to a different app or webpage.

Blockchain links – or blinks – turn any Solana Action into a shareable, metadata-rich link. Blinks allow Action-aware clients (browser extension wallets, bots) to display additional capabilities for the user. On a website, a blink might immediately trigger a transaction preview in a wallet without going to a decentralized app; in Discord, a bot might expand the blink into an interactive set of buttons. This pushes the ability to interact on-chain to any web surface capable of displaying a URL.

Actions#

The Solana Actions specification uses a set of standard APIs to deliver signable transactions (and eventually signable messages) from an application directly to a user. They are hosted at publicly accessible URLs and are therefore accessible by their URL for any client to interact with.

Info You can think of Actions as a API endpoint that will return metadata and something for a user to sign (either a transaction or a authentication message) with their blockchain wallet. The Actions API consists of making simpleGET andPOST requests to an Action's URL endpoint and handling the responses that conform to the Actions interface.

1. theGET request
2. returns metadata that provides human-readable
3. information to the client about what actions are available at this URL, and
4. an optional list of related actions.
5. thePOST request
6. returns a signable transaction or message
7. that the client then prompts the user's wallet to sign and execute on the
8. blockchain or in another off-chain service.

Action Execution and Lifecycle#

In practice, interacting with Actions closely resembles interacting with a typical REST API:

- a client makes the initialGET
- request to an Action URL in order to fetch
- metadata about the Actions available
- the endpoint returns a response that include metadata about the endpoint (like
- the application's title and icon) and a listing of the available actions for
- this endpoint
- the client application (like a mobile wallet, chat bot, or website) displays a
- UI for the user to perform one of the actions
- after the user selects an action (by clicking a button), the client makes aPOST
- request to the endpoint in order to get the transaction for the user to
- sign
- the wallet facilitates the user signing the transaction and ultimately sends
- the transaction to the blockchain for confirmation

Solana Actions Execution and Lifecycle

When receiving transactions from an Actions URL, clients should handle submission of these transactions to the blockchain and manage their state lifecycle.

Actions also support some level of invalidation before execution. TheGET andPOST request may return some metadata that states whether the action is capable of be taken (like with thedisabled field).

For example, if there was an Action endpoint that facilitates voting on a DAO governance proposal whose voting window has closed, the initialGET request may return the error message "This proposal is no longer up for a vote" and the "Vote Yes" and "Vote No" buttons as "disabled".

Blinks#

Blinks (blockchain links) are client applications that introspect Action APIs and construct user interfaces around interacting with and executing Actions.

Client applications that support blinks simply detect Action-compatible URLs, parse them, and allow users to interact with

them in standardized user interfaces.

Info Any client application that fully introspects an Actions API to build a complete interface for it is ablink . Therefore, not all clients that consume Actions APIs are blinks.

Blink URL Specification#

A blink URL describes a client application that enables a user to complete the fullifecycle of executing an Action , including signing with their wallet.

https://example.domain/?action= For any client application to become a blink:

- The blink URL must contain a query parameter ofaction
- whose value is a
- URL-encodedAction URL
- . This value must beURL-encoded
- to not conflict with any other protocol parameters.
- The client application mustURL-decode
- theaction
- query parameter and introspect the Action API link provided (seeAction URL scheme
-).
- The client must render a rich user interface that enables a user to complete
- the fulllifecycle of executing an Action
- ,
- including signing with their wallet.

Info Not all blink client applications (e.g. websites or dApps) will support all Actions. Application developers may chose which Actions they want to support within their blink interfaces. The following example demonstrates a valid blink URL with anaction value ofsolana-action:https://actions.alice.com/donate that is URL encoded:

https://example.domain/?action=solana-action%3Ahttps%3A%2F%2Factions.alice.com%2Fdonate

Detecting Actions via Blinks#

Blinks may be linked to Actions in at least 3 ways:

1. Sharing an explicit Action URL:solana-action:https://actions.alice.com/donate
2. In this case, only supported clients may render the blink. There will be no
3. fallback link preview, or site that may be visited outside of the
4. non-supporting client.
5. Sharing a link to a website that is linked to an Actions API via aactions.json file
6. on the website's domain root.
7. For example,https://alice.com/actions.json
8. mapshttps://alice.com/donate
9. , a website URL at which users can donate to Alice,
10. to API URLhttps://actions.alice.com/donate
11. , at which Actions for donating
12. to Alice are hosted.
13. Embedding an Action URL in an “interstitial” site URL that understands how to
14. parse Actions.
15. https://example.domain/?action=

Clients that support blinks should be able to take any of the above formats and correctly render an interface to facilitate executing the action directly in the client.

For clients that do not support blinks, there should be an underlying website (making the browser become the universal fallback).

If a user taps anywhere on a client that is not an action button or text input field, they should be taken to the underlying site.

Blink Testing and Verification#

While Solana Actions and blinks are a permissionless protocol/specification, client applications and wallets are still required to ultimately facilitate users to sign the transaction.

Each of these client applications or wallets may have different requirements on which Action endpoints their clients will automatically unfurl and immediately display to their users on social media platforms.

For example, some clients may operate on an "allow list" approach that may require verification prior to their client unfurling an Action for users such as Dialect's Actions Registry (detailed below).

All blinks will still render and allow for signing on Dialect's dial.to blinks Interstitial site, with their registry status displayed in the blink.

Dialect's Actions Registry#

As a public good for the Solana ecosystem, Dialect maintains a public registry — together with the help of Solana Foundation and other community members — of blockchain links that have are from pre-verified from known sources. As of launch, only Actions that have been registered in the Dialect registry will unfurl in the Twitter feed when posted.

Client applications and wallets can freely choose to use this public registry or another solution to help ensure user security and safety. If not verified through the Dialect registry, the blockchain link will not be touched by the blink client, and will be rendered as a typical URL.

Developers can apply to be verified by Dialect here dial.to/register

Specification#

The Solana Actions specification consists of key sections that are part of a request/response interaction flow:

- Solana Action URL scheme
- providing an Action URL
- OPTIONS response
- to an Action URL to pass CORS
- requirements
- GET request
- to an Action URL
- GET response
- from the server
- POST request
- to an Action URL
- POST response
- from the server

Each of these requests are made by the Action client (e.g. wallet app, browser extension, dApp, website, etc) to gather specific metadata for rich user interfaces and to facilitate user input to the Actions API.

Each of the responses are crafted by an application (e.g. website, server backend, etc) and returned to the Action client . Ultimately, providing a signable transaction or message for a wallet to prompt the user to approve, sign, and send to the blockchain.

URL Scheme#

A Solana Action URL describes an interactive request for a signable Solana transaction or message using the solana-action protocol.

The request is interactive because the parameters in the URL are used by a client to make a series of standardized HTTP requests to compose a signable transaction or message for the user to sign with their wallet.

solana-action: * A single link * field is required as the pathname. The value must be a * conditionally URL-encoded * absolute HTTPS URL. * If the URL contains query parameters, it must be URL-encoded. URL-encoding the * value prevents conflicting with any Actions protocol parameters, which may be * added via the protocol specification. * If the URL does not contain query parameters, it should not be URL-encoded. * This produces a shorter URL and a less dense QR code.

In either case, clients must URL-decode the value. This has no effect if the value isn't URL-encoded. If the decoded value is not an absolute HTTPS URL, the wallet must reject it as malformed .

OPTIONS response#

In order to allow Cross-Origin Resource Sharing CORS) within Actions clients (including blinks), all Action endpoints should respond to HTTP requests for the OPTIONS method with valid headers that will allow clients to pass CORS checks for all subsequent requests from their same origin domain.

An Actions client may perform "preflight " requests to the Action URL endpoint in order check if the subsequent GET request to the Action URL will pass all CORS checks. These CORS preflight checks are made using the OPTIONS HTTP method and should respond with all required HTTP headers that will allow Action clients (like blinks) to properly make all subsequent requests from their origin domain.

At a minimum, the required HTTP headers include:

- Access-Control-Allow-Origin
- with a value of*
-
    - this ensures all Action clients can safely pass CORS checks in order to make
-
    - all required requests
- Access-Control-Allow-Methods
- with a value ofGET,POST,PUT,OPTIONS
-
    - ensures all required HTTP request methods are supported for Actions
- Access-Control-Allow-Headers
- with a minimum value ofContent-Type, Authorization, Content-Encoding, Accept-Encoding

For simplicity, developers should consider returning the same response and headers toOPTIONS requests as theirGET response .

GET Request#

The Action client (e.g. wallet, browser extension, etc) should make an HTTPGET JSON request to the Action's URL endpoint.

- The request should not identify the wallet or the user.
- The client should make the request with anAccept-Encoding header
- .
- The client should display the domain of the URL as the request is being made.

GET Response#

The Action's URL endpoint (e.g. application or server backend) should respond with an HTTPOK JSON response (with a valid payload in the body) or an appropriate HTTP error.

- The client must handle HTTPclient errors
- ,server errors
- ,
- andredirect responses
- .
- The endpoint should respond with aContent-Encoding header
- for HTTP compression.
- The endpoint should respond with aContent-Type header
- ofapplication/json
- .
- The client should not cache the response except as instructed byHTTP caching
- response headers.
- The client should display thetitle
- and render theicon
- image to user.

GET Response Body#

AGET response with an HTTPOK JSON response should include a body payload that follows the interface specification:

ActionGetResponse export interface

ActionGetResponse { /**image url that represents the source of the action request */ **icon : string ;** /describes the source of the action request */ title : string ; /brief summary of the action to be performed/* description : string ; /**button text rendered to the user */ **label : string ;** / UI state for the button being rendered to the user*/ disabled ?: boolean ; links ?: { / list of related Actions a user could perform* / actions : LinkedAction []; }; */* non-fatal error message to be displayed to the user* / error ?: ActionError ; } * icon * - The value must be an absolute HTTP or HTTPS URL of an icon image. The * file must be an SVG, PNG, or WebP image, or the client/wallet must reject it * asmalformed * . * title * - The value must be a UTF-8 string that represents the source of the * action request. For example, this might be the name of a brand, store, * application, or person making the request. * description * - The value must be a UTF-8 string that provides information on * the action. The description should be displayed to the user. * label * - The value must be a UTF-8 string that will be rendered on a button * for the user to click. All labels should not exceed 5 word phrases and should * start with a verb to solidify the action you want the user to take. For * example, "Mint NFT", "Vote Yes", or "Stake 1 SOL". * disabled * - The value must be boolean to represent the disabled state of the * rendered button (which displays thelabel * string). If no value is provided,disabled * should default tofalse * (i.e. enabled by default). For example, * if the action endpoint is for a governance vote that has closed, setdisabled=true * and thelabel * could be "Vote Closed". * error * - An optional error indication for non-fatal errors. If present, the * client should display it to the user. If set, it should not prevent the client * from interpreting the

action or displaying it to the user. For example, the * error can be used together withdisabled * to display a reason like business * constraints, authorization, the state, or an error of external resource.

ActionError export interface

ActionError { /* *non-fatal error message to be displayed to the user*/ message : string ; } * links.actions * - An optional array of related actions for the endpoint. Users * should be displayed UI for each of the listed actions and expected to only * perform one. For example, a governance vote action endpoint may return three * options for the user: "Vote Yes", "Vote No", and "Abstain from Vote". * * If nolinks.actions * * is provided, the client should render a single button * * using the rootlabel * * string and make the POST request to the same action * * URL endpoint as the initial GET request. * * If anylinks.actions * * are provided, the client should only render buttons * * and input fields based on the items listed in thelinks.actions * * field. The * * client should not render a button for the contents of the rootlabel * * .

LinkedAction export interface

LinkedAction { / **URL endpoint for an action */ href : string ; /**button text rendered to the user*/ *label : string ; /Parameter to accept user input within an action* / parameters ?: [ActionParameter]; }

/ **Parameter to accept user input within an action */ export interface ActionParameter {** /parameter name in url*/ name : string ; / placeholder text for the user input field*/ label ?: string ; /* declare if this field is required (defaults to*false*)* / required ?: boolean ; }

Example GET Response#

The following example response provides a single "root" action that is expected to be presented to the user a single button with a label of "Claim Access Token":

{ "title" : "HackerHouse Events" , "icon" : "" , "description" : "Claim your Hackerhouse access token." , "label" : "Claim Access Token" // button text } The following example response provides 3 related action links that allow the user to click one of 3 buttons to cast their vote for a DAO proposal:

{ "title" : "Realms DAO Platform" , "icon" : "" , "description" : "Vote on DAO governance proposals #1234." , "label" : "Vote" , "links" : { "actions" : [{ "label" : "Vote Yes" , // button text "href" : "/api/proposal/1234/vote?choice=yes" }, { "label" : "Vote No" , // button text "href" : "/api/proposal/1234/vote?choice=no" }, { "label" : "Abstain from Vote" , // button text "href" : "/api/proposal/1234/vote?choice=abstain" }] } }

Example GET Response with Parameters#

The following examples response demonstrate how to accept text input from the user (viaparameters) and include that input in the finalPOST request endpoint (via thehref field within aLinkedAction):

The following example response provides the user with 3 linked actions to stake SOL: a button labeled "Stake 1 SOL", another button labeled "Stake 5 SOL", and a text input field that allows the user to enter a specific "amount" value that will be sent to the Action API:

{ "title" : "Stake-o-matic" , "icon" : "" , "description" : "Stake SOL to help secure the Solana network." , "label" : "Stake SOL" , // not displayed since links.actions are provided "links" : { "actions" : [{ "label" : "Stake 1 SOL" , // button text "href" : "/api/stake?amount=1" // no parameters therefore not a text input field }, { "label" : "Stake 5 SOL" , // button text "href" : "/api/stake?amount=5" // no parameters therefore not a text input field }, { "label" : "Stake" , // button text "href" : "/api/stake?amount={amount}" , "parameters" : [{ "name" : "amount" , // field name "label" : "SOL amount" // text input placeholder }] }] } } The following example response provides a single input field for the user to enter anamount which is sent with the POST request (either as a query parameter or a subpath can be used):

{ "icon" : "" , "label" : "Donate SOL" , "title" : "Donate to GoodCause Charity" , "description" : "Help support this charity by donating SOL." , "links" : { "actions" : [{ "label" : "Donate" , // button text "href" : "/api/donate/{amount}" , // or /api/donate?amount={amount} "parameters" : [// {amount} input field { "name" : "amount" , // input field name "label" : "SOL amount" // text input placeholder }] }] } }

POST Request#

The client must make an HTTPPOST JSON request to the action URL with a body payload of:

{ "account" : "" } * account * - The value must be the base58-encoded public key of an account that * may sign the transaction.

The client should make the request with anAccept-Encoding header and the application may respond with aContent-Encoding header for HTTP compression.

The client should display the domain of the action URL as the request is being made. If aGET request was made, the client

should also display thetitle and render theicon image from that GET response.

POST Response#

The Action'sPOST endpoint should respond with an HTTPOK JSON response (with a valid payload in the body) or an appropriate HTTP error.

- The client must handle HTTPclient errors
- ,server errors
- ,
- andredirect responses
- .
- The endpoint should respond with aContent-Type header
- ofapplication/json
- .

POST Response Body#

APOST response with an HTTPOK JSON response should include a body payload of:

ActionPostResponse export interface

ActionPostResponse { /**base64 encoded serialized transaction */ transaction : string ;** /describes the nature of the transaction */ message ?: string ; } * transaction * - The value must be a base64-encodedserialized transaction * . * The client must base64-decode the transaction anddeserialize it * . * message * - The value must be a UTF-8 string that describes the nature of the * transaction included in the response. The client should display this value to * the user. For example, this might be the name of an item being purchased, a * discount applied to a purchase, or a thank you note. * The client and application should allow additional fields in the request body * and response body, which may be added by future specification updates.

Info The application may respond with a partially or fully signed transaction. The client and wallet must validate the transaction asuntrusted .

POST Response - Transaction#

If the transactionsignatures are empty or the transaction has NOT been partially signed:

- The client must ignore thefeePayer
- in the transaction and set thefeePayer
- to theaccount
- in the request.
- The client must ignore therecentBlockhash
- in the transaction and set therecentBlockhash
- to thelatest blockhash
- .
- The client must serialize and deserialize the transaction before signing it.
- This ensures consistent ordering of the account keys, as a workaround forthis issue
- .

If the transaction has been partially signed:

- The client must NOT alter thefeePayer
- orrecentBlockhash
- as this would invalidate any existing signatures.
- The client must verify existing signatures, and if any are invalid, the client
- must reject the transaction asmalformed
- .

The client must only sign the transaction with theaccount in the request, and must do so only if a signature for theaccount in the request is expected.

If any signature except a signature for theaccount in the request is expected, the client must reject the transaction asmalicious .

actions.json#

The purpose of theactions.json file allows an application to instruct clients on what website URLs support Solana Actions and provide a mapping that can be used to performGET requests to an Actions API server.

Theactions.json file should be stored and universally accessible at the root of the domain.

For example, if your web application is deployed tomy-site.com then theactions.json file should be accessible athttps://my-site.com/actions.json .

Rules#

Therules field allows the application to map a set of a website's relative route paths to a set of other paths.

Type: Array ofActionRuleObject .

ActionRuleObject interface

ActionRuleObject { / **relative (preferred) or absolute path to perform the rule mapping from */ pathPattern : string ;** / relative (preferred) or absolute path that supports Action requests */ apiPath : string ; } * pathPattern * - A pattern that matches each incoming * pathname. * apiPath * - A location destination defined as an absolute * pathname or external URL.

Rules - pathPattern#

A pattern that matches each incoming pathname. It can be an absolute or relative path and supports the following formats:

- Exact Match
- : Matches the exact URL path.
- - Example:/exact-path
- - Example:https://website.com/exact-path
- Wildcard Match
- : Uses wildcards to match any sequence of characters in the
- URL path. This can match single (using*
-) or multiple segments (using**
-).
- (seePath Matching
- below).
- - Example:/trade/*
- - will match/trade/123
- - and/trade/abc
- - , capturing only
- - the first segment after/trade/
- - .
- - Example:/category//*item/**
- - will match/category/123/item/456
- - and/category/abc/item/def
- - .
- - Example:/api/actions/trade/*/confirm
- - will match/api/actions/trade/123/confirm
- - .

Rules - apiPath#

The destination path for the action request. It can be defined as an absolute pathname or an external URL.

- Example:/api/exact-path
- Example:https://api.example.com/v1/donate/*
- Example:/api/category//*item/*
- Example:/api/swap/**

Rules - Query Parameters#

Query parameters from the original URL are always preserved and appended to the mapped URL.

Rules - Path Matching#

The following table outlines the syntax for path matching patterns:

Operator Matches * A single path segment, not including the surrounding path separator / characters. ** Matches zero or more characters, including any path separator / characters between multiple path segments. If other operators are included, the** operator must be the last operator. ? Unsupported pattern.

Rules Examples#

The following example demonstrates an exact match rule to map requests requests to/buy from your site's root to the exact path/api/buy relative to your site's root:

actions.json { "rules" : [{ "pathPattern" : "/buy" , "apiPath" : "/api/buy" }] } The following example uses wildcard path matching to map requests to any path (excluding subdirectories) under/actions/ from your site's root to a corresponding path under/api/actions/ relative to your site's root:

actions.json { "rules" : [{ "pathPattern" : "/actions/*" , *"apiPath" : "/api/actions/*"* }] } The following example uses wildcard path matching to map requests to any path (excluding subdirectories) under/donate/ from your site's root to a corresponding absolute pathhttps://api.dialect.com/api/v1/donate/ on an external site:

actions.json { "rules" : [{ "pathPattern" : "/donate/*" , *"apiPath" : "https://api.dialect.com/api/v1/donate/*"* }] } The following example uses wildcard path matching for an idempotent rule to map requests to any path (including subdirectories) under/api/actions/ from your site's root to itself:

Info Idempotent rules allow blink clients to more easily determine if a given path supports Action API requests without having to be prefixed with thesolana-action: URI or performing additional response testing. actions.json { "rules" : [{ "pathPattern" : "/api/actions/**" , "apiPath" : "/api/actions/**" }] }

Action Identity#

Action endpoints may include anAction Identity in the transactions that are returned in theirPOST response for the user to sign. This allows indexers and analytics platforms to easily and verifiably attribute on-chain activity to a specific Action Provider (i.e. service) in a verifiable way.

TheAction Identity is a keypair used to sign a specially formatted message that is included in transaction using a Memo instruction. ThisIdentifier Message can be verifiably attributed to a specific Action Identity, and therefore attribute transactions to a specific Action Provider.

The keypair is not required to sign the transaction itself. This allows wallets and applications to improve transaction deliverability when no other signatures are on the transaction returned to a user (seePOST response transaction).

If an Action Provider's use case requires their backend services to pre-sign the transaction before the user does, they should use this keypair as their Action Identity. This will allow one less account be included in the transaction, lowering the total transactions size by 32-bytes.

Action Identifier Message#

The Action Identifier Message is a colon separate UTF-8 string included in a transaction using a singleSPL Memo instruction.

protocol:identity:reference:signature * protocol * - The value of the protocol being used (set tosolana-action * per * theURL Scheme * above) * identity * - The value must be the base58-encoded public key address of the * Action Identity keypair * reference * - The value must be base58-encoded 32-byte array. This may or may * not be public keys, on or off the curve, and may or may not correspond with * accounts on Solana. * signature * - base58-encoded signature created from the Action Identity * keypair signing only thereference * value.

Thereference value must be used only once and in a single transaction. For the purpose of associating transactions with an Action Provider, only the first usage of thereference value is considered valid.

Transactions may have multiple Memo instructions. When performing agetSignaturesForAddress , the resultsmemo field will return each memo instruction's message as a single string with each separated by a semi-colon.

No other data should be included with Identifier Message's Memo instruction.

Theidentity and thereference should be included as read-only, non-signerkeys in the transaction on an instruction that is NOT the Identifier Message Memo instruction.

The Identifier Message Memo instruction must have zero accounts provided. If any accounts are provided, the Memo program requires theses accounts to be valid signers. For the purposes of identifying actions, this restricts flexibility and can degrade the user experience. Therefore it is considered an anti-pattern and must be avoided.

Action Identity Verification#

Any transaction that includes theidentity account can be verifiably associated with the Action Provider in a multi-step process:

1. Get all the transactions for a givenidentity
2. .
3. Parse and verify each transaction's memo string, ensuring thesignature
4. is
5. valid for thereference
6. stored.
7. Verify the specific transaction is the first on-chain occurrence of thereference
8. on-chain:* If this transaction is the first occurrence, the transaction is considered
9.
    - verified and can a safely attributed to the Action Provider.
10.
    - If this transaction is NOT the first occurrence, it is considered invalid
11.
    - and therefore not attributed to the Action Provider.

Because Solana validators index transactions by the account keys, thegetSignaturesForAddress RPC method can be used locate all transactions including theidentity account.

This RPC method's response includes all the Memo data in thememo field. If multiple Memo instructions were used in the transaction, each memo message will be included in thismemo field and must be parsed accordingly by the verifier to obtain theIdentity Verification Message .

These transactions should be initially consideredUNVERIFIED . This is due to theidentity not being required to sign the transaction which allows any transaction to include this account as a non-signer. Potentially artificially inflating attribution and usage counts.

The Identity Verification Message should be checked to ensure thesignature was created by theidentity signing thereference . If this signature verification fails, the transaction is invalid and should be attributed to the Action Provider.

If the signature verification is successful, the verifier should ensure this transaction is the first on-chain occurrence of thereference . If it is not, the transaction is considered invalid.

Previous «State Compression Next Rust»