AmberTime Blockchain Documentation

Overview **5**

Scope **5**

Authority Node Management 5
Record management 5
Service/Product Listing 6
Service/Product Availment 7
Certification 8
Blockchain Processing 9
Authorization 9
Payments and Fees 10
Other requirements 10

Detailed Steps **10**

Testing Pages 11
Using the WebAPI 11
Using Postman to test the WebAPI 11
Creating test addresses 12
Registering addresses 12
Handling raw transactions 12
Getting AMTC 14
Checking your AMTC balance 14
Sending AMTC 14
Fetching Transactions 14
Writing a record 14
Fetching Records 14
Sharing a record 14
Sharing a time limited access record 15
Viewing a shared record 15
Creating a service / product 15
Updating a service / product 15
Searching for a service / product 15
Restocking a service / product 16
Reducing the quantity of a service / product 16
Purchasing a non-consumable service / product 16
Purchasing a consumable service / product 16
Completing a purchase 17
Refunding a purchase 17

Running your own blockchain node 17
Installation 17
Invoking API functions via the CLI 17
Register as authority 18
Define new badge / certificate 18
Allow others to issue the badge / certificate 18

Appendix A: Blockchain API Documentation 18
approveauthority 18
requestauthority 19
writerecord 19
annotaterecord 20
revokerecord 20
createbadge 21
updatebadge 21
issuebadge 22
revokebadge 22
requestissuebadge 23
grantbadgeissuerpermission 23
revokebadgeissuerpermission 24
annotatebadge 24
writecategory 25
writerecordtype 25
listservice 26
updateservice 26
delistservice 26

Appendix B: WebAPI Documentation 27
Get Public Key 27
Create Key Pairs 27
Create Service 27
Update Service 29
Add Service Quantity 31
Remove Service Quantity 32
Search Service 32
Purchase Non Consumable Service 33
Purchase Consumable Service (3-part) 33
Claim Purchase 36
Update Purchase Status 37
Purchase History 37

Purchases 38
Transactions 38
Write Record 38
Create Badge with Issuer 39
Issue Badge 40
Revoke Badge 40
My Records 41
Issuable Badges 41
My Badges 41

I. Overview

AmberTime Blockchain is the blockchain platform of AmberTime with the goal of providing a platform for the exchange of education, travel and other services.

There are two ways to participate in the AmberTime blockchain:

- You can run your own blockchain node by installing the amberchain source from https://github.com/ambertime/amberchain/
- You can interact with a blockchain node via the WebAPI running on the node. The WebAPI endpoints are accessed via HTTP and return response in JSON format.

The following blockchain nodes are made available by AmberTime:

- https://amber0.ddns.net (Hong Kong)
- https://amber1.ddns.net (Singapore)
- https://amber2.ddns.net (South Korea)

II. Scope

This section covers the use cases and scenarios applicable to the AmberTime Blockchain

Authority Node Management

Node Registration	Partners can download the blockchain node application from ambertime.org and install it on their own servers. But before the server can become a participating node in the blockchain, the administrator must send a request to ambertime.org through the server. The administrator fills in the identity information about the organization and sends a certificate signing request to ambertime.org. Upon approval, the digital certificate is automatically uploaded into the server and the blockchain, and that's the only time when a node can start to function as an authority node.

Note: AmberTime's approval of authority node requests will be held every 6:00 PM daily.

Record management

Write Record	Anyone can write a record and link it to an account id. The fields are: ● Title ● Record Type - the default list of record types are maintained by AmberTime Org. Example are: Experience, Sports, Evidence ● Category - list of category that is under the record type ● Text

	• Document - attach a file to the record. This record can be an image. This record is private and can only be viewed by the account that wrote the record and account linked.
View Record	Record will be viewable by both the account used to create it, and account linked. This means that a private / public key will be generated to encrypt the data. This key pair will be encrypted by the private keys of the account who created the record and account linked. When either of the accounts linked to the record wishes to view the record, they will use the encrypted private key to decrypt the data.
Revoke Record	The account that wrote the record will be able to revoke the record. When a record is revoked, the record will still be viewable but with a status indicating that it is revoked. An optional text can be filled to indicate the reason for revoking. Primarily used for filtering.
Annotate Record	The account that wrote the record will be able to add annotations to a record. Text field with entries time stamped.

Service/Product Listing

Service/Product Types	• Consumable Product/Services - Merchant can specify product name, description, photo/image, quantity available and expiration period, refund policy and price (in fiat or AMTC amount). Buyer can specify quantity to be purchased. The merchant can optionally specify whether separate transactions should be created for each instance of the item or not, E.g. for Iphone, merchant specifies to record as separate transactions. So when user buys 2 iPhones, the 2 iPhones will be recorded as 2 separate transactions each requiring their own confirmation, and each can be delivered and confirmed separately. On the other hand, if separate transactions is not specified, then confirming the transaction would confirm all quantities purchased (such as the case of ice cream vendor). The quantity available should be displayed.

	● Certification/Badge Service - Merchant can specify service name, maturity period (duration required), photo/image, information to be included the certification. When user makes a purchase, he/she must fill in the information that is to be included in the certification. The corresponding badge/certificate can only be granted after the maturity period has passed. E.g. user enrolls in a training session, merchant requires user to fill in name, selected training dates, upon completion of the training, merchant can grant the user a certificate/badge with these information embedded. ● Custom Services - Merchant can specify service name, initialization web service URL, product web service URL, refund web service URL, expiration, maturity period, refund policy, success code, error message id, photo/image, client information to fill in, price (in fiat or AMTC), initial status, completion status, Initialization web service and refund web services are only required/invoked when expiration is set. (Note: Sending the calls to the external APIs will be done in the wallet level and are not included in the WebAPI)
Search/Filter Listing	User must be able to search or filter the posted services/products by category, by price range, by product/service name, by merchant entity or by description directly in the wallet application.
Service update	Posted Services and products can be updated by the poster by creating a new version of the service, when buyers purchase a service or product, the version of the service/product is indicated in the transaction. The quantity available is tracked on the blockchain, and the poster can update the quantity anytime.

Service/Product Availment

Service Availment through search result	Buyers can browse/search through the listing and click on the service to view the details, then avail it by clicking purchase and enter the information required. If purchase is successful, user will be informed, and the successful transaction will be viewable in the list of transactions of the user

Certification

Authorizing Wallets for Badge processing	Authority Nodes can grant wallet addresses with permission to perform badge and certification operations. However, the final issuance of badge/certification is still done by the authority node through the digital signature of the server. I.e. define which wallets can invoke the signing of certifications on the authority node. This is specific to each authority node.
Create Badge	Only an authority can create badge. In creating a badge, an authority will enter, but not limited to, the following details: ● Badge title - examples are: Degree in Computer Science, Certificate of Participation in Math Learning ● Category - a default list of categories will be available for them to choose from. Examples are: certificate, experience, degree. This list is maintained by AmberTime Org. ● Photo - upload a badge photo ● A badge can also be linked to a Service / Product. ○ If linked, before the badge can be granted, the maturity specified in the service/product must be validated. Authority will also be able to request information to be included in the certification. If the badge is linked to a service / product, this information will be requested from the user upon purchase. When issuing the badge to an account who did not purchase the linked product/service or if the badge is not linked to any service / product, the Authority will be the one to enter the information. E.g. user enrolls in a training session, merchant requires user to fill in name, selected training dates, upon completion of the training, merchant can grant the user a certificate/badge with these information embedded. Authority can also customize the display of the badge in the user's account using html / vue.js / handlebars.
Modify Badge	The authority that created the badge has the permission to modify the badge information. When an authority modify the badge, the records of the students that have earned the badge is also updated to the latest information.
Issue Badge	There are two ways of issuing a badge: 1. An authority issues a badge to an Account ID.

	2. An authority issues badge to all linked services/product that satisfy the time requirement. This badge is private and can only be viewed by the Authority that issue the badge and the account issued to.
View Badge	Record will be viewable by both the Authority and account issued to. This means that a private / public key will be generated to encrypt the data. This key pair will be encrypted by the private keys of the Authority and account issued to. When the Authority or account linked to the record wishes to view the badge, they will use the encrypted private key to decrypt the data.
Revoke badge	The badge issuer can revoke a badge that was issued to a user account. An optional text field can be filled by the Authority indicating the reason why he is revoking this badge. In the user account, the badge will still show but there will be an indicator saying that this badge was revoked and the reason for it.
Annotate Badge	The Authority that wrote the record will be able to add annotations to a badge. Text field with entries time stamped.

Blockchain Processing

Expiring Transactions Processing	The blockchain nodes must be able to process expiring transactions and perform the refund logic without triggers from any of the wallets.

Authorization

Authorize View Access to a Record / BadgePost	The owner of the record / badge can set the record to be accessible by another account. A time expiry can be given such that after the allotted time, the record will not anymore be accessible. If the time expiry is enabled, the owner of the record will choose a date until which the record will be accessible. For instance, if the owner chose April 21, 2018, the data will not anymore be viewable after this date.

	Owner can also specify a set of records to be included each time authorization is granted for others to view his records. Such as the identity certification records.
Set Record / Badge to Public	A record / badge can be set to public. Once a record / badge is set to public, anyone can view the record. This record cannot be set back to private after it is set to public.

Payments and Fees

Direct Payment (fund transfer)	Straight credit of funds, no dependency. Allow to input comments. Allow to set an alias to the address.
Based on Fiat exchange rate	Ability to define service/product prices in terms of fiat currency and show prices to the user in terms of fiat currency. The exchange rate shall be provided by ambertime.org
Refund Processing	Ability to trigger refund before receipt of goods is done by a user
Escrow	_Ability to hold funds in the blockchain until the fund is either sent to the merchant or refunded to the buyer, or distributed to both.

Other requirements

Transaction fees	Transaction fee can be set per byte of data to be written to the blockchain. The rate is to be obtained from a web service from ambertime.org (outside of scope), all fees collected are to be forwarded directly to ambertime's wallet based on the % defined by ambertime. Ambertime's wallet address is to be obtained from the blockchain. All transactions generated/initiated by the authority nodes shall pay no transaction fees.
User records	Allow the user to view all records related to him/her

III. Detailed Steps

A. Testing Pages

The following web pages are made available to facilitate testing. These pages are available on all the 3 nodes, but the example urls below use amber0.

https://amber0.ddns.net/test/signing - used for signing raw transactions. You will need to input the decoded raw transaction returned by the WebAPI and the private key to be used for signing. The page will generate the raw transaction in hex, and it can be sent to the blockchain via the /webapi/sendsignedtx endpoint. See “Handling raw transactions” below for more instructions.

https://amber0.ddns.net/test/encrypt - used for encrypting/decrypting data to be sent to the blockchain. The encryption is asymmetric; the data will be encrypted using a public key, and the decryption will be done using the recipient’s private key. The public key for a registered address can be retrieved using the /webapi/getpubkey endpoint (refer to Appendix)

https://amber0.ddns.net/test/confirmation - used to generate a confirmation code that a buyer can give to a seller. This confirmation code will allow the seller to complete a purchase without the buyer having to claim the purchase. See Claim Purchase Web API.

https://amber0.ddns.net/test/keys - provides a function to get the public key for a given private key, and also to convert a private key to PEM format for use with OpenSSL

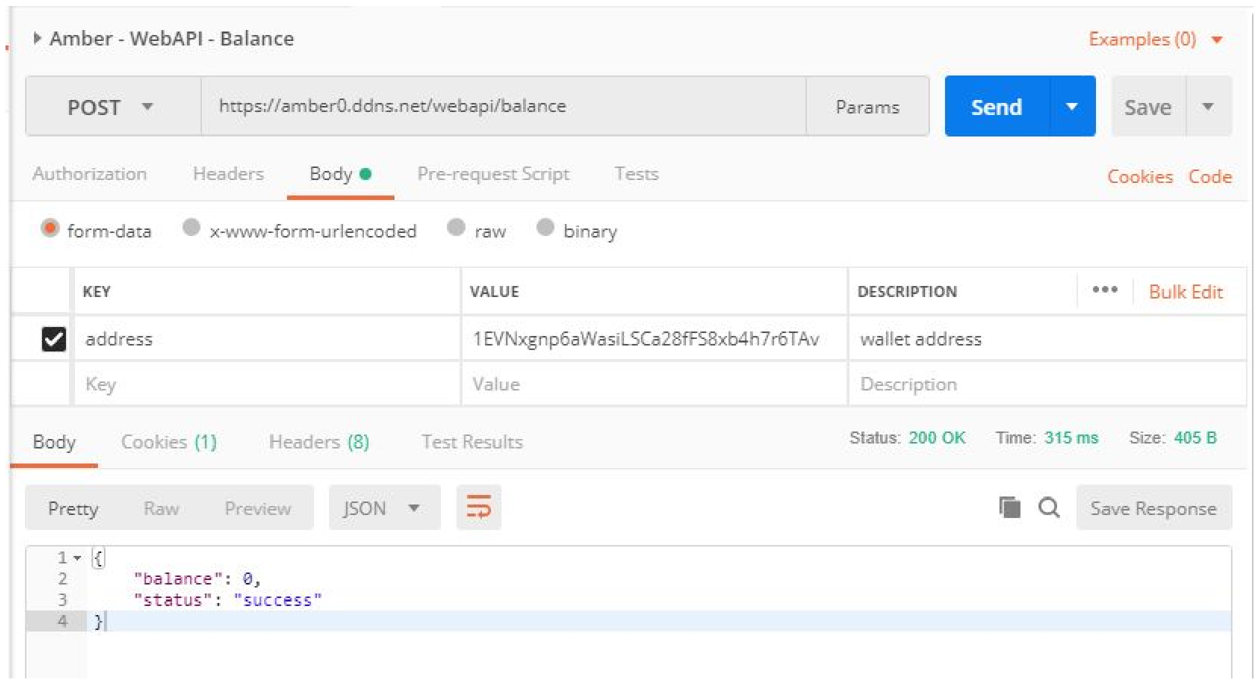
B. Using the WebAPI

The WebAPI consists of a series of endpoints accessed using HTTP and returning JSON response. This section describes how to use the WebAPI for various tasks. You may refer to the Appendix B of this document for details on individual WebAPI endpoints. The WebAPI may be accessed via any of the nodes listed in the overview.

Using Postman to test the WebAPI

We recommend using Postman (https://www.getpostman.com/) software for testing the WebAPI endpoints.

Here’s an example screenshot of using Postman to send a WebAPI request:



https://amber0.ddns.net/webapi/balance

KEY	VALUE	DESCRIPTION
address	1EVNxgnp6aWasiLSCa28fFS8xb4h7r6TAv	wallet address
Key	Value	Description

Status: 200 OK Time: 315 ms Size: 405 B

```
{
    "balance": 0,
    "status": "success"
}
```

Method - should always be post
URL - choose one of the host names provided (in this case https://amber0.ddns.net) and append the endpoint (in this example, /webapi/balance)
Parameters should be specified under the “Body” tab
The response will be shown in the lower part of the window, in JSON format.

Creating test addresses

An address, private key, and public key can be generated using the Create Key Pairs Web API. Store this information. The private key will be used to sign most of the transactions the address will perform on the blockchain.

Registering addresses

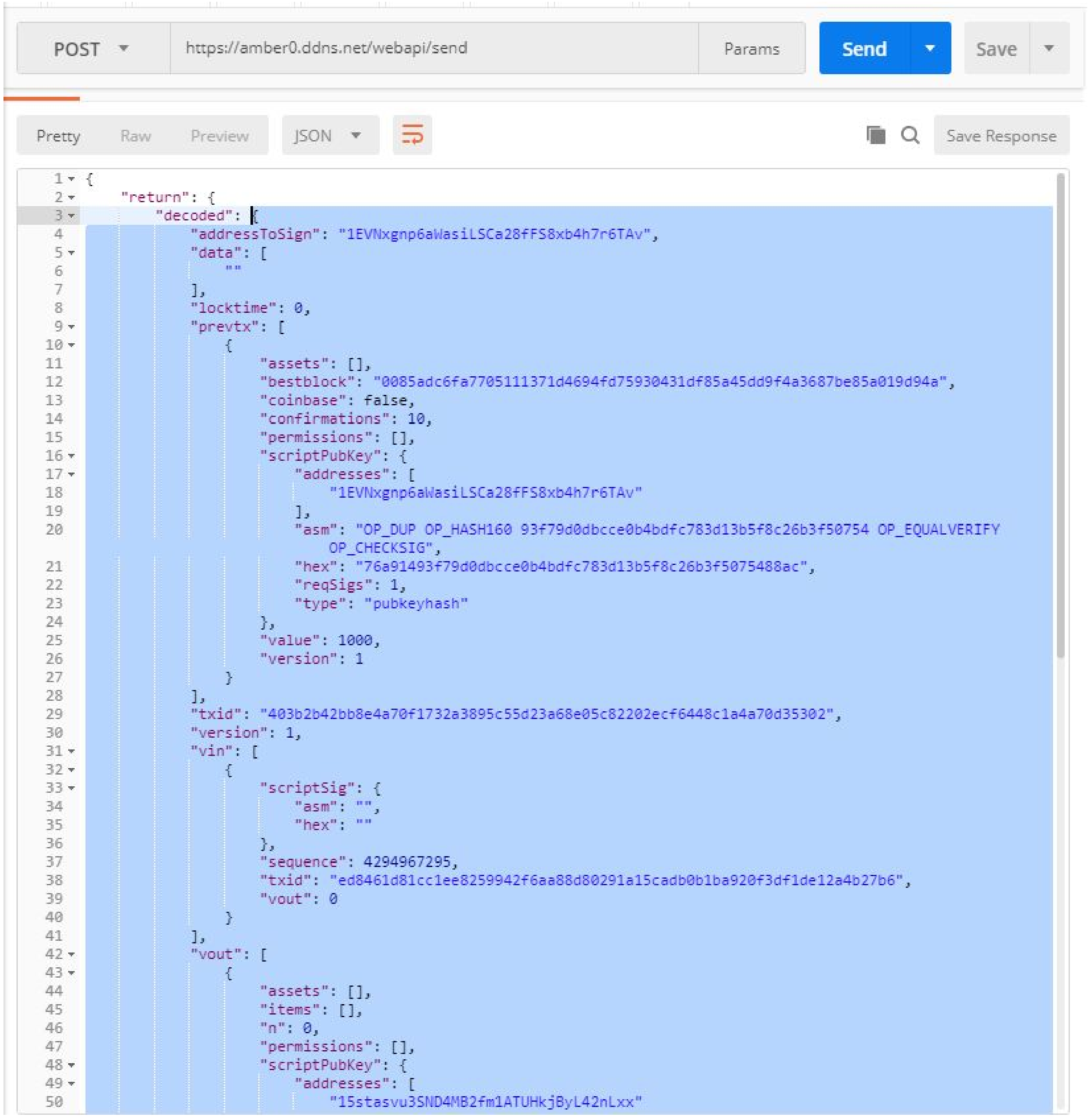
Once you have a set of address, private key and public key for testing, you should register the address and public key using /webapi/register endpoint. This needs to be done at least once whenever you use a new node for the WebAPI, as it will allow the node to track your AMTC balance.

Handling raw transactions

The WebAPI server will not store your private keys, so whenever you need to perform a transaction that writes to the blockchain, you will need to sign the transaction on the client side. For testing purposes, the /test/signing page is provided to perform the transaction signing via JavaScript (Refer to the Testing Pages section)

In such a case, the WebAPI endpoint will return a “decoded” version of the transaction. The contents of the “decoded” section should be copied into the /test/signing page and signed using the private key of the transaction sender.

Example of the WebAPI returning a raw transaction with a “decoded” entry:



https://amber0.ddns.net/webapi/send

```
{
    "return": {
        "decoded": {
            "addressToSign": "1EVNxgnp6aWasiLSCa28fFS8xb4h7r6TAv",
            "data": [
                ""
            ],
            "locktime": 0,
            "prevtx": [
                {
                    "assets": [],
                    "bestblock": "0085adc6fa7705111371d4694fd75930431df85a45dd9f4a3687be85a019d94a",
                    "coinbase": false,
                    "confirmations": 10,
                    "permissions": [],
                    "scriptPubKey": {
                        "addresses": [
                            "1EVNxgnp6aWasiLSCa28fFS8xb4h7r6TAv"
                        ],
                        "asm": "OP_DUP OP_HASH160 93f79d0dbcce0b4bdfc783d13b5f8c26b3f50754 OP_EQUALVERIFY
                            OP_CHECKSIG",
                        "hex": "76a91493f79d0dbcce0b4bdfc783d13b5f8c26b3f5075488ac",
                        "reqSigs": 1,
                        "type": "pubkeyhash"
                    },
                    "value": 1000,
                    "version": 1
                }
            ],
            "txid": "403b2b42bb8e4a70f1732a3895c55d23a68e05c82202ecf6448c1a4a70d35302",
            "version": 1,
            "vin": [
                {
                    "scriptSig": {
                        "asm": "",
                        "hex": ""
                    },
                    "sequence": 4294967295,
                    "txid": "ed8461d81cc1ee8259942f6aa88d80291a15cadb0b1ba920f3df1de12a4b27b6",
                    "vout": 0
                }
            ],
            "vout": [
                {
                    "assets": [],
                    "items": [],
                    "n": 0,
                    "permissions": [],
                    "scriptPubKey": {
                        "addresses": [
                            "15stasvu3SND4MB2fm1ATUHkjByL42nLxx"
```

Once the transaction has been signed, a long hex string will be generated. Unless otherwise specified, this hex transaction should be passed to the /webapi/sendsignedtx endpoint which will write the signed transaction to the blockchain.

If you encounter an error regarding ConnectInputs, please verify that the correct private key has been used to sign the transaction.

Getting AMTC

For testing purposes, we have provided a /webapi/faucet endpoint that will give your wallet address 1000 AMTC. The use of this faucet is limited to once every 24 hours per address.

Note: /web/faucet must be done in the Admin Node (https://amber1.ddns.net/)

Checking your AMTC balance

You may check the AMTC balance for any address by using the /webapi/balance endpoint.

Sending AMTC

You may send AMTC to any other valid address by using the /webapi/send endpoint.

Fetching Transactions

A summary of the recent transactions for an address may be viewed by retrieving from the /webapi/profile/transactions endpoint.

Writing a record

The Write Record function is available via the /webapi/profile/writerecord. The data to be passed is expected to be encrypted on the client side. This can be done using the /test/encrypt page.

Fetching Records

You can view records written to this address via the /webapi/profile/myrecords endpoint. Note that the data returned here will be as it was passed to the /webapi/profile/writerecord endpoint, if the data was encrypted as expected, this endpoint will return the encrypted data.

Sharing a record

You may share records with another address using the /webapi/sharetxn. The “payload” passed should be the content you wish to share in JSON format, encrypted using the recipient’s public key.

Sharing a time limited access record

When using /webapi/sharetxn, you may also specify an access expiry date. When this is provided, an additional layer of encryption is applied when writing the payload to the blockchain, and authority nodes will have to verify that the access time is valid before decrypting the content.

Viewing a shared record

You can use /webapi/listsharedtxns to view a list of records shared via the two methods above. This will show you who shared the record and if there is an expiry date.

To view the details of the content shared, take the txid from the list above and pass it to /webapi/viewsharedtxn. If there is an expiry period, authority nodes will verify first that the access is valid. The returned payload here is as it was passed to /webapi/sharetxn, so if it was encrypted there, the client side will have to decrypt the payload to view the content.

Creating a service / product

To create a service a merchant should use the Create Service Web API. A service with a quantity equal to 0 will be a non consumable service. Attach necessary certificates and photos, and provide refund policies and expiration. A transaction id will be provided upon completion. This transaction ID will be used for purchasing, and updating this service.

Updating a service / product

An existing service can be updated using the Update Service Web API. Only updates to details are permitted. This mean that a quantity of an existing service cannot be modified using this endpoint.

Searching for a service / product

After a service is created, allow for some time to pass so that it can be indexed by the routine process. This process runs every 3 minutes. Once a service is indexed, you can search for it using the Search Service Web API. This should return the most recent information about the services that match the search criteria. This will include a TXID of the original service, which will be used to purchase a service.

Restocking a service / product

When the quantity of a consumable service is depleting or if there is any need to restock a service, the Add Service Quantity Web API can be used. Using the TXID of the original service, any amount can be added to a service.

Reducing the quantity of a service / product

Similar to restocking a service, this requires the TXID of the original service. Using the Remove Service Quantity WEB API, the quantity of a consumable service can be reduced. This send the current stock to the burn address and requires the confirmation of peers.

Purchasing a non-consumable service / product

Using the service TXID obtained through Search Service a product/ service can be purchased with the Purchase Non Consumable Web API. Purchased services can be viewed using the Purchases Web API.

Purchasing a consumable service / product

Step 1

Using the Purchase Consumable Step 1 Web API, pass the transaction ID of the service to be purchased, quantity to purchase, and wallet address of purchaser. This will return a decoded raw transaction that should be signed with the private key of the purchaser. The signed transaction will be used in Step 2.

Step 2:
Using the Purchase Consumable Step 2 Web API, still pass the service transaction ID, quantity, and wallet address. Make sure that these values don't change for all the steps of the purchase. In addition to these parameters pass the signed transaction from step one as "signedtx". This will return a decoded raw transaction for signing and, if needed, an escrow address. Sign the raw transaction and copy the escrow address.

Step 3:
Pass the signed raw transaction as "exchangeoffertx" and the escrow address as "escrow_address" to the Purchase Consumable Step 3 Web API. If there was no escrow address returned in step 2, leave the escrow_address parameter blank. Pass the same service transaction ID, quantity, and wallet address along with the exchange offer tx and escrow address.

Step 4:

Sign the raw transaction returned in step 3 and use the Send Signed TX Web API. This will complete the purchase and write it to the stream.

For a more detailed description of this process please see the Purchase Consumable Web API in the Appendix.

Completing a purchase

After purchasing a consumable or non-consumable service, if the service has no expiration period specified, the transaction is completed.

.

If the service has an expiration period, the amount is put into escrow. This purchase can be completed by using the Claim Service Web API. A claim can be done provided that a purchase is mature. A request to claim should be done by both the merchant and buyer in order for a purchase to be completed. Additionally, if there is a badge/certificate associated with the service that was purchased, the merchant should issue the certificate using the /webapi/requestissuebadge endpoint once the maturity period is reached.

The seller can complete a purchase, without the buyer making a claim, with the use of a confirmation code provided by the buyer. See the testing pages for more information of confirmation codes.

Refunding a purchase

Refunding a purchase will be done using the Claim Service Web API. Provided there are necessary refund option, the refund value is automatically computed and is sent back to the buyer. There is no need for the merchant to confirm this refund. You can view a full history of a purchase through the Purchase History Web API.

C. Running your own blockchain node

You may also choose to participate in the blockchain directly by running your own node and connecting to the testnet chain.

Installation

Refer to https://github.com/ambertime/amberchain/blob/amber-dev/README.md for instructions on downloading the source from github and installing the blockchain node. We recommend installing on Ubuntu Linux.

Invoking API functions via the CLI

Once you have a running node, you can run blockchain API commands directly via the amberchain-cli executable. You can invoke “amberchain-cli amber-testnet” to enter interactive mode.

The available blockchain API commands are listed in Appendix A below, please refer there for the parameters and other details. While most of the functionality can be tested directly using the WebAPI, certain functions may be available only to those running a node. Such operations are detailed below.

Register as authority

Authority nodes are nodes which are allowed to participate in mining activities on the chain. To request to become an authority node, generate a certificate signing request and submit it via the requestauthority API method.

The from-address passed to the API method should be an address belonging to the node. By default one such address is created when you connect to the chain. Use listaddresses API call on the amberchain CLI..

For creating the certificate signing request, this can be done manually using OpenSSL. You may check the procedure here under “Generating the CSR”: https://www.ssl.com/how-to/manually-generate-a-certificate-signing-request-csr-using-openssl/

For the above, you will need to provide a private key in PEM format used to sign the CSR. It is recommended to use the private key of the from-address as provided by amberchain. To get the private key, you may use dumpprivkey API call on the amberchain CLI. This will give you the private key in multichain format. We have provided a function on the /test/keys Testing Page to convert this to PEM format for use with OpenSSL.

The requests will need to be vetted and manually approved by Ambertime.

Note: AmberTime’s approval of authority node requests will be held every 6:00 PM daily.

Define new badge / certificate

Once approved as authority, you can create a new badge/certificate using the createbadge API call. You can update the definition using updatebadge API call.

A created badge can be issued to another address using the issuebadge API call.

Allow others to issue the badge / certificate

Normally, only the authority node that created the badge can issue that same badge. You may use grantbadgeissuerpermission API call to allow other addresses to issue the badge as well. Once others have been granted permission, they may use the /webapi/requestissuebadge to issue the badge even without a blockchain node.

Alternative to the above two processes, we have provided a WebAPI endpoint /webapi/badges/createbadgeissuer that will create a badge and automatically assign a certain address permission to grant it. This WebAPI can be invoked without needing to run your own blockchain node.

IV. Appendix A: Blockchain API Documentation

Amberchain is built on top of Multichain, hence all the Multichain API calls listed at https://www.multichain.com/developers/json-rpc-api/ are supported. In Addition, the following custom API endpoints are available.

approveauthority

Location: *rpcpermissions.cpp*
Permission: admin

Description	Approves request for authority and gives the necessary permissions
Parameters	from-address `address of an admin wallet that approves the request` to-address `address of the requestor` public-key `public key of the CSR` digital-certificate `signed certification of the CSR` certificate-details `JSON string of the CSR contents`
Returns	Stream transaction ID of the log in the stream that records authority nodes

requestauthority

Location: *rpcpermissions.cpp*
Permission: none

Description	Sends a request for authority permissions by writing to the stream that logs all the requests
Parameters	from-address: `address of the requestor` public-key: `public key of the CSR. This should be the same as the public key of your node's wallet address` csr: `the certificate signing request, in PEM format`
Returns	Stream transaction ID of the log in the stream that logs all requests to be authority nodes

writerecord

Location: *rpcstreams.cpp*
Permission: none

Description	Writes an entry to the record stream
Parameters	from-address: `address of the writer` wallet-address: `the address that the writer wishes to write to` encrypted-data: `JSON data encrypted using the recipient's public key` encrypted-key: `JSON data encrypted using the sender's public key`
Returns	Raw transaction hash to be signed.

annotaterecord

Location: *rpcstreams.cpp*

Permission: must be the same publisher with the record that will be annotated

Description	Annotates a current entry in the records stream
Parameters	from-address `address of the writer` stream-txid `stream transaction ID that is to be annotated` encrypted-data `encrypted JSON data`
Returns	Raw transaction hash to be signed.

revokerecord

Location: *rpcstreams.cpp*
Permission: must be the same publisher with the record that will be revoked

Description	Revokes a current entry in the records stream
Parameters	from-address `address of the writer` stream-txid `stream transaction ID that is to be annotated` encrypted-data `encrypted JSON data`
Returns	Raw transaction hash to be signed

createbadge

Location: *rpcstreams.cpp*
Permission: authority

Description	Creates a badge in the amberchain network
Parameters	badge-creator `address of the creator`

	badge-data	badge data in JSON format		
		category	root txid in categories stream	
		degree	string	
		photo	base 64 encoded photo	
		customhtml	string	
		dynamicfields	array of objects each containing the fields below	
			field_type	string
			field_required	string
			field_name	string
Returns	Stream transaction ID of the log in the stream that logs all existing badges in the network			

updatebadge

Location: *rpcstreams.cpp*
Permission: authority and must be badge creator

Description	Updates the detail of an existing badge in the network.	
Parameters	badge-creator	address of the creator
	badge-txid	stream transaction ID of the badge in the badges stream
	badge-annotations	additional data in JSON format
Returns	Stream transaction ID of the log in the stream that logs all existing badges in the network	

issuebadge

Location: *rpcstreams.cpp*
Permission: authority and must be badge creator

Description	Issues an existing badge to an address
Parameters	from-address `address of the issuer` badge-receiver `address that receives the badge` badge-txid `stream transaction ID of the  badge in the badges stream` badge-notes `badge notes in JSON format`
Returns	Raw transaction hash that needs to be signed

revokebadge

Location: *rpcstreams.cpp*
Permission: authority and must be badge creator

Description	Revokes an existing badge to an address
Parameters	from-address `address of the issuer` badge-receiver `address that the received the badge will be revoked from` badge-txid `stream transaction ID of the  badge in the badges stream` badge-notes `badge notes in JSON format`
Returns	Raw transaction hash that needs to be signed

requestissuebadge

Location: *rpcstreams.cpp*

Permission: none

Description	Sends a request to issue a badge
Parameters	from-address — `address of the badge creator` badge-receiver — `address of the badge receiver` badge-txid — `stream transaction ID of the badge in the badges stream` badge-notes — `badge notes in JSON format` request-status — `status of the request` badge-action — `badge action to grant to receiver` issue-badge-requestor — `address of the requestor for issuing/revoking a badge`
Returns	Raw transaction hash that needs to be signed

grantbadgeissuerpermission

Location: *rpcstreams.cpp*
Permission: authority and must be badge creator

Description	Grants a wallet address permission to issue a badge
Parameters	from-address — `address of the badge creator` badge-txid — `stream transaction ID of the badge in the badges stream` badge-issuer-address — `address to grant badge issue permission`
Returns	Stream transaction ID of the log in the stream that logs all addresses that can issue badges

revokebadgeissuerpermission

Location: *rpcstreams.cpp*
Permission: authority and must be badge creator

Description	Revokes a wallet address' permission to issue a badge
Parameters	from-address: `address of the badge creator` badge-txid: `stream transaction ID of the badge in the badges stream` badge-issuer-address: `address to revoke badge issue permission`
Returns	Stream transaction ID of the log in the stream that logs all addresses that can issue badges

annotatebadge

Location: *rpcstreams.cpp*
Permission: authority and must be badge creator

Description	Annotate an existing badge to update its details
Parameters	from-address: `address of the badge creator` badge-txid: `stream transaction ID of the badge in the badges stream` badge-annotations: `JSON data that contains changes to the badge details`
Returns	Stream transaction ID of the log in the stream that logs all the details of annotated badges

writecategory

Location: *rpcstreams.cpp*
Permission: admin

Description	Adds a new category to the network
Parameters	

	from-address — address of the category creator category-key — unique string that identifies the category category-data — details of the category in JSON format
Returns	Stream transaction ID of the log in the stream that logs all the categories in the chain

writerecordtype

Location: *rpcstreams.cpp*
Permission: admin

Description	Adds a new record type to the network
Parameters	from-address — address of the record type creator record-type-key — unique string that identifies the record type record-type-data — details of the record type in JSON format
Returns	Stream transaction ID of the log in the stream that logs all the record types in the chain

listservice

Location: *rpcstreams.cpp*
Permission: must be able to write to services stream

Description	Adds a new service to the network
Parameters	from-address — address of the service creator service-data — JSON details of the service
Returns	Raw transaction hash that needs to be signed

updateservice

Location: *rpcstreams.cpp*
Permission: must be able to write to services stream & must be the creator of the service

Description	Updates an existing service's details
Parameters	from-address `address of the service creator` stream-txid `transaction ID of the item to be updated in the services stream` service-data `updated details of the service in JSON format`
Returns	Raw transaction hash that needs to be signed

delistservice

Location: *rpcstreams.cpp*
Permission: must be able to write to services stream & must be the creator of the service

Description	Delists an existing service in the network
Parameters	from-address `address of the service creator` stream-txid `transaction ID of the item to be updated in the services stream` service-data `updated details of the service in JSON format`
Returns	Raw transaction hash that needs to be signed

V. Appendix B: WebAPI Documentation

Get Public Key

Endpoint	/webapi/getpubkey
Description	Fetches the public key of a given address
Parameters	address `: String`
Response	Public key

Create Key Pairs

Endpoint	/webapi/createkeypairs
Description	Generates a new address with public and private key
Parameters	No parameters
Response	address : "", pubkey : "", privkey : ""

Register Address

Endpoint	/webapi/register
Description	Register an address to the node
Parameters	address `: String : Address to register` pubkey `: String : Public key of address`
Response	`status : "success"`

Get AMTC

Endpoint	/webapi/faucet
Description	Get AMTC for your address
Parameters	address `: String : Address of receiver`
Response	`status : “success”`

Send AMTC

Endpoint	/webapi/send
Description	Send AMTC from an address to another address
Parameters	from_address `: String : Address of sender` to_address `: String : Address of receiver` amount `: Integer : Number of AMTC to send` remarks `: String : Any notes`

Response	Json object containing decoded raw transaction that needs to be signed by the sender.
	`status : “raw”`
	`return : {`
	`decoded : {}, // decoded raw transaction`
	`message : "Transfer successful",`
	`raw : “”, // raw transaction`
	`}`

Check AMTC Balance

Endpoint	/webapi/balance
Description	Check the balance of an address
Parameters	address `: String : Address to check`

Response	`status : "success",`
	`balance : "" // Balance of address`

Send Signed Transaction

Endpoint	/webapi/sendsignedtx
Description	Send a signed transaction to the blockchain
Parameters	signedtx `: String : Hex returned in /test/signing`

Response	`status : "success"`
	`return : {`
	`message : "",`
	`txid : "", // txid of sent transaction`
	`}`

Create Service

Endpoint	/webapi/services/create
Description	Creates a new service

Parameters	walletaddress : `String` : `Address of Vendor` productname : `String` producttype : `String` : `[ "Consumable Products/Services", "Certification Badge/Service", "Custom Services" ]` description : `String` amount : `Float` currency : `String` : `[ "AMTC" ]` category : `String` : `[ no choices yet ]` quantity : `Integer` separatetx : `Boolean` expirationperiod : `Integer` expirationrefund : `Integer` availabilityfrom : `Date` : `yyyy-mm-dd` availabilityto : `Date` : `yyyy-mm-dd` photo : `File` certificate : `String` : `root TXID of Badge` maturityperiod : `Integer` Refund Policies refundpolicies-0-refundpercentage : `Integer` refundpolicies-0-numberofdays : `Integer` refundpolicies-0-typeofdate : `String` : `[ "expirationperiod", "maturityperiod", "purchasedate" ]` refundpolicies-0-beforeorafter : `String` : `[ "before", "after" ]` Custom Transaction Statuses customstatuses-0-customstatusname : `String`

	Web Services
	`webservices-0-transactionstatus : String : [ “initialization”, “product”, “refund” ]`
	
	`webservices-0-url : String`

Response	Json object containing decoded raw transaction that needs to be signed by the vendor.
	
	`decoded : {}, // Decoded Raw Transaction`
	`message : "Successfully created service.",`
	`raw     : "", // Hex String or Raw Transaction`

Testing	You can validate that the service was created using the *Search Services* endpoint.
Notes	● expirationperiod valid beforeorafter value is before only ● maturityperiod valid beforeorafter value is before or after ● purchasedate valid beforeorafter value is after only

Update Service

Endpoint	/webapi/services/update
Description	Updates an existing service

Parameters	`walletaddress : String : Address of Vendor`
	`txid : String : TXID of original service`
	`productname : String`
	`description : String`
	`amount : Float`
	`currency : String : [ "AMTC" ]`
	`category : String : [ no choices yet ]`
	`separatetx : Boolean`

	expirationperiod `: Integer` availabilityfrom `: Date : yyyy-mm-dd` availabilityto `: Date : yyyy-mm-dd` photo `: File` certificate `: String : root TXID of Badge` maturityperiod `: Integer` Refund Policies refundpolicies-0-refundpercentage `: Integer` refundpolicies-0-numberofdays `: Integer` refundpolicies-0-typeofdate `: String : [ "expirationperiod", "maturityperiod", "purchasedate" ]` refundpolicies-0-beforeorafter `: String : Select [ "before", "after" ]` Custom Transaction Statuses customstatuses-1-customstatusname `: String` Web Services webservices-0-transactionstatus `: String : [ "initialization", "product", "refund" ]` webservices-0-url `: String`

Response	Json object containing decoded raw transaction that needs to be signed by the vendor.
	
	`decoded : {}, // Decoded Raw Transaction`
	`message : "Successfully created service.",`
	`raw     : "", // Hex String or Raw Transaction`

Testing	You can validate that the service was updated using the *Search Services* endpoint. This should now return the service with updated information.

Add Service Quantity

Endpoint	/webapi/services/addservicequantity
Description	Allows a vendor to add quantity to a service
Parameters	walletaddress `: String  :  Address of service creator` txid `: String  :  Service txid` quantity `: Integer :  quantity to add to the service`

Response	Json object containing decoded raw transaction that needs to be signed by the vendor.
	`decoded :{}, // decoded raw transaction`
	`message :  "Successfully removed quantity to service.",`
	`raw     : "", // Raw transaction`

Testing	Use *Search Services* to view the latest updated services.

Remove Service Quantity

Endpoint	/webapi/services/removeservicequantity
Description	Removes quantity from an existing service
Parameters	walletaddress `: String    : Address of service creator` txid `: String    : Service txid` quantity `: Integer   : quantity to remove from the service`

Response	Json object containing decoded raw transaction that needs to be signed by the vendor.
	`decoded :{}, // decoded raw transaction`
	`message :  "Successfully removed quantity to service.",`
	`raw     : "", // Raw transaction`

Testing	Use *Search Services* to view the latest updated services.

Search Service

Endpoint	/webapi/services/search
Description	Searches the indexed services.
Parameters	field : String : Select ["productname", "txid", "producttype", ”description"] searchparam : String : String to match to the value of the field specified start : Integer : lowest price to match end : Integer : maximum price to match
Response	Array of Objects Representing Services that match search parameters

Purchase Non Consumable Service

Endpoint	/webapi/services/purchasenonconsumable
Description	Purchases a non-consumable service (no quantity defined on creation)
Parameters	walletaddress : String : Address of the buyer txid : String : Service txid (optional) badgenotes_creator : String : Badge notes in JSON format, that is encrypted for the badge creator (optional) badgenotes_seller : String : Badge notes in JSON format, that is encrypted for the badge service seller
Response	JSON object containing decoded raw transaction that needs to be signed by the buyer.

	`status : "raw"`
	`return : {`
	`decoded : {}, // decoded raw transaction`
	`message : "Successfully purchased a non-consumable service.",`
	`raw : "", // raw transaction`
	`}`

Purchase Consumable Service (3-part)

Purchase Consumable Service: Step 1

Endpoint	/webapi/services/purchaseconsumable OR /webapi/services/purchaseconsumable?step=1
Description	Executes step 1 in purchasing a consumable service
Parameters	walletaddress `: String : Address of the buyer` txid `: String : Service txid` quantity `: Integer : Number of goods to purchase`

Response	Json object containing decoded raw transaction that needs to be signed by the buyer with his/her private key, and passed as `signedtx` parameter in step 2.
	`status : "raw"`
	`return : {`
	`decoded : {}, // decoded raw transaction`
	`message : "Successfully created step 1 in purchase of consumable service.",`
	`raw : "", // raw transaction`

	}

Purchase Consumable Service: Step 2

Endpoint	/webapi/services/purchaseconsumable?step=2
Description	Executes step 2 in purchasing a consumable service
Parameters	walletaddress `: String :` `Address of the buyer` txid `: String :` `Service txid` quantity `: Integer :` `Number of goods to purchase` signedtx `: String :` `Signed transaction that was returned by step 1.`

Response	Json object containing decoded raw transaction that needs to be signed by the buyer with his/her private key, and passed as `exchangeoffertx` parameter in step 2.
	If applicable, Json object also contains the escrow_address that will be passed in step 3, as `escrow_address`.
	```
	status         :   "raw"
	return         :   {
	                   decoded :   {}, // decoded raw transaction
	                   message :   "Successfully created step 2 in
	                               purchase  of consumable service.",
	                   raw     :   "", // raw transaction
	                   }
	(optional response data)
	escrow_address :   "", // escrow address if applicable
	```

Purchase Consumable Service: Step 3

Endpoint	/webapi/services/purchaseconsumable?step=3
Description	Executes step 3 in purchasing a consumable service
Parameters	walletaddress : String : Address of the buyer txid : String : Service txid quantity : Integer : Number of goods to purchase exchangeoffertx : String : Signed transaction that was returned by step 2. (optional) escrow_address : String : Escrow address that was returned by step 2. (optional) badgenotes_creator : String : Badge notes in JSON format, that is encrypted for the badge creator (optional) badgenotes_seller : String : Badge notes in JSON format, that is encrypted for the badge service seller

Response	Json object containing decoded raw transaction that needs to be signed by the buyer with his/her private key.
	
	`status : "raw"`
	`return : {`
	`decoded : {}, // decoded raw transaction`
	`message : "Successfully completed step 3 in purchase service. Just need to sign the included raw tx here that writes to the purchasestatus stream. ",`
	`exchangetxid : "", // txid of exchange`
	`raw : "", // raw transaction`
	`}`

Claim Purchase

Endpoint	/webapi/services/claimservice
Description	Claim or Refund an existing purchase
Parameters	address : `String` : `Wallet Address of buyer` roottxid : `String` : `Root TXID of Purchase to claim` newstatus : `String` : `Select [ "claim", "refund" ]` confirmationcode (optional) : `String` : `Hash of purchase TXID signed by buyer private key`

Response	Json object containing decoded raw transaction that needs to be signed by the buyer or vendor
	`decoded :{}, // decoded raw transaction`
	`message : "",`
	`raw     : "", // Raw transaction`

Testing	● Testing and verification can be done by reading the messages returned in the response. ● Use *Profile Purchases* to view the latest status of a profiles purchase. ● Use *Profile Purchase History* to view the full history of a purchase.
Notes	● *Refund* can be done by the buyer only. ● A *Claim* should be done by both the vendor and buyer to be completed.

Update Purchase Status

Endpoint	/webapi/services/updatepurchasestatus
Description	Updates a status of an existing purchase to a custom status.
Parameters	walletaddress : `String` : `Wallet Address of buyer` roottxid : `String` : `Root TXID of Purchase to claim`

	newstatus `: String : Custom Status`

Response	Json object containing decoded raw transaction that needs to be signed by the buyer or vendor
	`decoded :{}, // decoded raw transaction`
	`message : "Purchase Updated.",`
	`raw     : "", // Raw transaction`

Testing	● Use *Profile Purchases* to view the latest status of a profiles purchase. ● Use *Profile Purchase History* to view the full history of a purchase.
Notes	● Cannot update the status of a purchase to any of the following: [\`claimed\`,\`refunded\`,\`expired\`,\`completed\`,\`claim\`,\`refund\`]

Purchase History

Endpoint	/webapi/profile/purchase/history
Description	View a full history of a purchase
Parameters	walletaddress `: String : Address of buyer` txid `: String : Root TXID of the Purchase`
Response	Array of Objects representing all the updates made to a purchase. Sorted.

Purchases

Endpoint	/webapi/profile/purchases
Description	View latest status of all purchases of an address.
Parameters	walletaddress `: String : Address of buyer`
Response	Array of Objects representing the purchases of an address

Transactions

Endpoint	/webapi/profile/transactions
Description	Generates a new address with public and private key
Parameters	walletaddress : String : Address of buyer count (optional) : Integer : Number of latest transactions to fetch. If count is not passed default is 10
Response	Array of Objects representing the transactions that affected the address balance

Write Record

Endpoint	/webapi/profile/writerecord
Description	Writes a record from one address to another address
Parameters	from_address : String : Address of Sender to_address : String : Receiver encrypteddata : String : Record data encrypted encryptedkey : String : Encrypted key used to decrypt data
Response	Raw transaction the needs to be signed.

Annotate Record

Endpoint	/webapi/profile/annotaterecord
Description	Annotates a record previously written
Parameters	from_address : String : Address of Sender

	encrypteddata : String : Record data encrypted recordtxid : String : TXID of record to annotate
Response	Raw transaction the needs to be signed.

Revoke Record

Endpoint	/webapi/profile/writerecord
Description	Revokes a previously written record from a receiver
Parameters	from_address : String : TXID of record to annotate recordtxid : String : Receiver encripteddata : String : Record data encrypted
Response	Raw transaction the needs to be signed.

Create Badge with Issuer

Endpoint	/webapi/badges/createbadgeissuer
Description	[FOR TESTING PURPOSES ONLY] Creates a badge and grants permission to issue.
Parameters	issuer_address : String : Address to Grant issue permission to title : String : Title of badge degree : String category : String customhtml : String photo : File

	fields-0-fieldType : `String` fields-0-name : `String` fields-0-required : `String`
Response	Transaction ID of the created badge..

Issue Badge

Endpoint	/webapi/badges/requestissuebadge
Description	Requests to issue a badge to an address
Parameters	badgecreator : `String` : `Address of badge creator` badgereceiver : `String` : `Address to issue badge to` txid : `String` : `Transaction ID of Badge to issue` walletaddress : `String` : `Address of Badge Issuer` badgenotes : `JSON` : `Badge notes in JSON format`
Response	Raw transaction the needs to be signed by the badge issuer.

Revoke Badge

Endpoint	/webapi/badges/requestrevokebadge
Description	Requests to revoke a badge from an address
Parameters	badgecreator : `String` : `Address of badge creator` badgereceiver : `String` : `Address to revoke badge from` txid : `String` : `Transaction ID of Badge to issue`

	walletaddress `: String : Address of Badge Issuer`
Response	Raw transaction the needs to be signed by the badge issuer.

My Records

Endpoint	/webapi/profile/myrecords
Description	Retrieves the records written to given address
Parameters	walletaddress `: String : Address to retrieve records for`
Response	Array of objects representing the records written to this address.

My Records OutBox

Endpoint	/webapi/profile/fetchrecords
Description	Retrieves the records written by a given address
Parameters	walletaddress `: String : Address to retrieve records for`
Response	Array of objects representing the records written to this address.

Get Record Updates

Endpoint	/webapi/profile/recordupdates
Description	Retrieves the annotations and revokes of a record
Parameters	

	roottxid : String : Root TXID of record with updates to fetch
Response	Array of objects representing the record updates.

Issuable Badges

Endpoint	/webapi/profile/issuablebadges
Description	Retrieves a list of badges that a given address can issue.
Parameters	walletaddress : String : Address of Badge Issuer Count (optional) : Integer : Number of most recent badges to retrieve
Response	Array of objects representing the badges that an address can issue.

My Badges

Endpoint	/webapi/profile/mybadges
Description	Retrieves a list of badges that were issued to a given address.
Parameters	walletaddress : String : Address of Badge Recepient Count (optional) : Integer : Number of most recent badges to retrieve
Response	Array of objects representing the badges that an address was issued.