

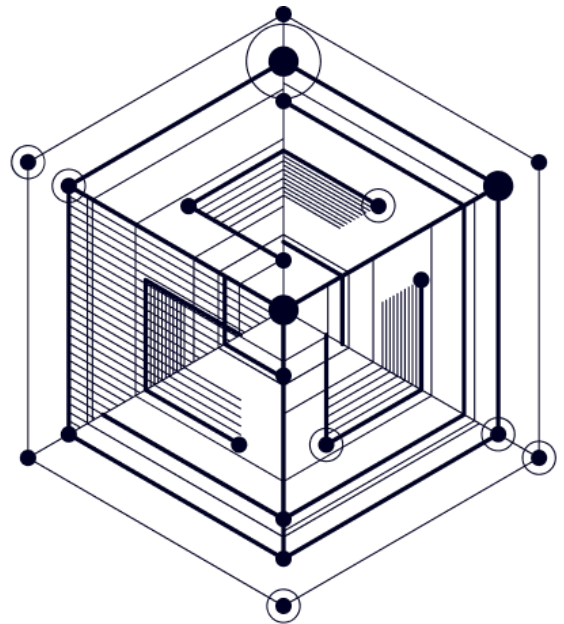
Tegment SDK

The official Relysia Unity SDK to use the Bitcoin SV blockchain

Relysia SDK

Quickly build your own applications with the Relysia Wallet SDK!

Developer Kit



Scroll to Explore

Infrastructure

Keep full control and make it your own.

Relysia is a wallet infrastructure, that means we can not only be accessed by logging into Vaionex platforms, but moreover enable 3rd party developers to create their very own dedicated versions of it. Relysia is a **Blockchain As A Service** platform, to create your very own sophisticated blockchain platforms while keeping your very own user database and control over settings and funds.

If you intend to build your own blockchain platform, go to the wallet interface of Relysia.com to sign up to your very own Infrastructure Plan. or Contact adam@vaionex.com

During the Infrastructure Plan setup, you can choose between a variety of functions, enable and disable user functionalities, tailored to your very own Application.

Miner Fee Settings

To create the optimal user experience for your user, you need to decide what options the user will prefer for your application. At the moment, Relysia Infrastructure supports 3 fee types.

- Classical: Let users pay for transaction fees with BSV
- Miner: Use A minerAPI cover your costs (e.g. TAAL mAPI Key)
- Dynamic: Cover the costs of your users to a certain limit (FeeManager).

Infrastructure Costs

In our initial beta, we will keep plans free of costs to certain limits. Within the next month, we will announce a low cost volume based plan that will keep average applications with 50k users well below 30\$ per month.

Wallets

How to create your first Blockchain Wallet.

Wallets

Once you create a new account (with signUP), you first need to create a wallet. Since Relysia allows you to pick between multiple wallet types, we rather leave the wallet choice to the user on their setup. Let's explore how to set up your first wallet.

Create new Wallet

As the name already indicates, the `createWallet.CreateWalletAccount()` adds a new wallet straight to your user account. There is no limit to the number of wallets you can have in a single account. Lets first explore the most simple **standard wallet** case, before exploring the

extended version. Using the `walletTitle: 'default'` tag, will create a wallet with a deterministic path id `0000-000000000-00000` .

Create Wallet

Wallet Title*

default

mnemonicPhrase

Enter mnemonicPhrase...

Paymail

Enter Paymail...

Type

Type...

Wallet Logo

Wallet Logo...

Activate

☒

Submit

Response

response text

```
Tegment.Wallet.CreateWallet.CreateWalletAccount(walletTitle.text,
walletActivate.isOn, TegmentSessionHandler.Instance._authToken,
CreateWalletCallBack, mnemonicPhrase.text, paymail.text, _type.text,
walletLogo.text,true);
```

Above is the example of wallet creation, Here is the details of params in a following table

Sr. No.	Param Name	Type	IsRequired	details
1.	_walletTitle	string	yes	If default is already created then yes
2.	_paymailActivate	bool	yes	
3.	_mnemonicPhrase	string	no	
4.	_paymail	string	no	
5.	_type	string	no	
6.	_walletLogo	string	no	
7.	_authToken	string	yes	You will get auth token on successful login or signup

8.	Callback	function	yes	To handle the data back from SDK
9.	enableLog	bool	no	If you want to add for log

*In most functions you can see a **walletId** input, if you create a default wallet, you can leave the field open and the package will automatically access the default wallet for the operation.*

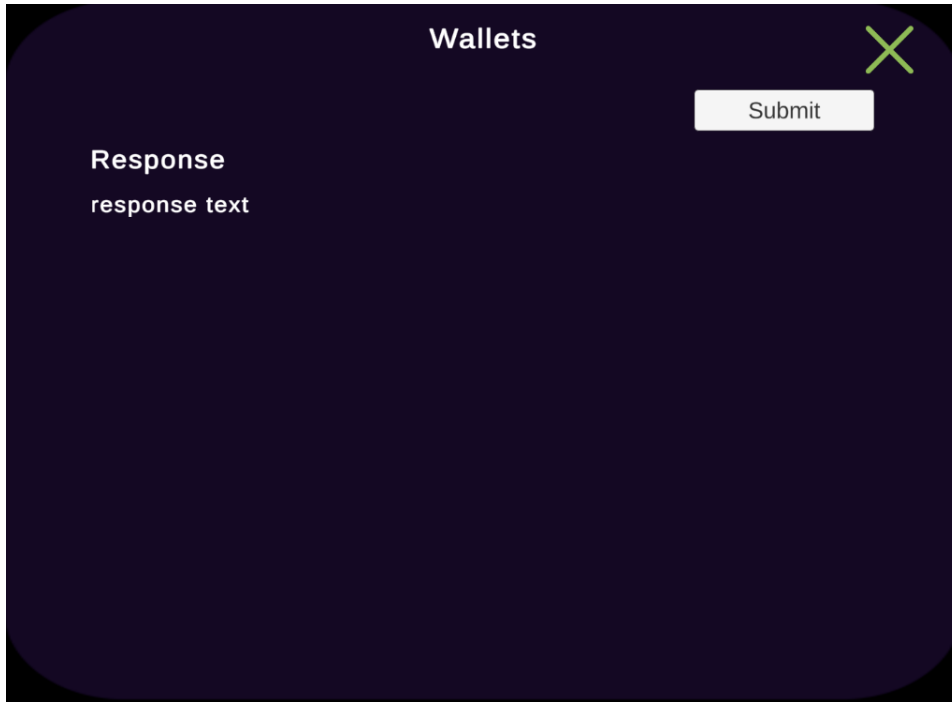
The standard wallet will do the trick for most wallet applications in terms of encryption and usability. For extended functionality, such as password protection or escrow wallets, Please fill all the input as shown in below

Create a HD wallet of choice in your user account. You can select “Type” among standard, secure, escrow and shared wallets (see docs.relysia.com).

*There are 4 wallet types: **Standard Wallets** have a base encryption, **Secure Wallets** are additionally password protected, unencrypted **Escrow Wallets** and **Multi Party Wallets** (that make use of multi-sig schemes). Each type has very specific use cases that depend on your applications needs*

List All Wallets

This wallet function returns you a list of all the wallets associated with your Relysia account. The result details include an array of: **walletID**, **walletTitle** and a link to your **walletLogo**. It takes only authToken as header param and returns all the wallets for the relysia account.



```
Tegment.Wallet.Wallets.GetWallets(TegmentSessionHandler.Instance._authToken, WalletsCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_authToken	string	yes	
2	Callback	function	yes	
3	enableLog	bool	no	

Delete specific Wallets

This wallet function will delete the specific wallet. It takes walletId and authToken as header param. There are two additional params here, callback which will return the data and another is for logging.

Delete Wallet

WalletId

Response

response text

WalletId...

Submit

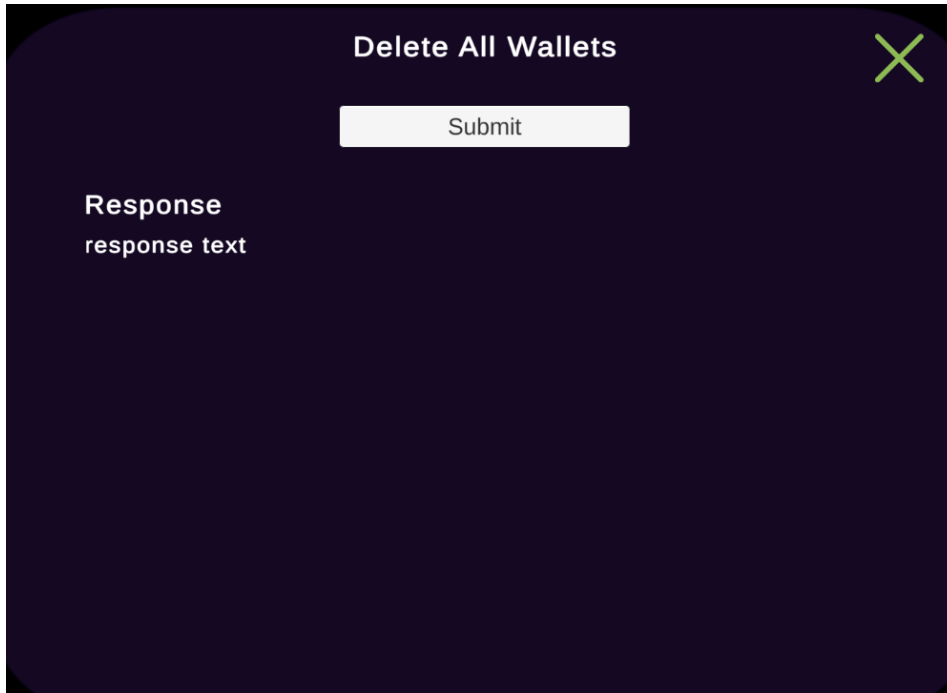
```
Tegment.Delete.DeleteWallet.Wallet(walletId.text,  
TegmentSessionHandler.Instance._authToken, DeleteWalletCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_walletID	string	yes	
2	_authToken	string	yes	
3	Callback	function	yes	
4	enableLog	bool	no	

Delete all Wallets

This wallet function deletes all the wallet associated with the current user account,It takes authToken as header param. There are two additional params here, callback which will return the data and another is for logging.



```
Tegment.Delete.DeleteAllWallets.AllWallets(TegmentSessionHandler.Instance._authToken, DeleteAllWalletsCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_authToken	string	yes	
2	Callback	function	yes	
3	enableLog	bool	no	

Mnemonic

We are non-custodial - make sure to keep your wallet backup

An essential aspect of regulatory compliance of our Blockchain as a service infrastructure is to keep wallets non-custodial. Users are interacting with their very own keys and we only store those fragments that enable them to unlock them for transaction processing purposes.

Mnemonic Phrase

As a non-custodial wallet, users are responsible for backing up their keys. The mnemonic phrase function returns the 12 word backup seed-phrase of the wallet selected.

Mnemonic

WalletID

WalletID...

Submit

Response

response text

```
Tegment.Wallet.Mnemonic.GetMnemonic(walletID.text,  
TegmentSessionHandler.Instance._authToken, MnemonicCallBack,true);
```

Make sure you backup your mnemonic phrase and never share it with anyone.

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_walletID	string	yes	
2	_authToken	string	yes	
3	Callback	function	yes	
4	enableLog	bool	no	

Retrieve Funds

The mnemonic phrase is the seed for a HDPrivateKey with millions of potential addresses. You can extract your funds from the **M/44/0/0/{1,2,3...}** path. First transactions will always go to the first addresses and their corresponding private keys, while over time, to prevent address re-usage for privacy reasons we use a larger and larger address space. Nonetheless, knowing the order of writings, you can recover all funds without Relysia.

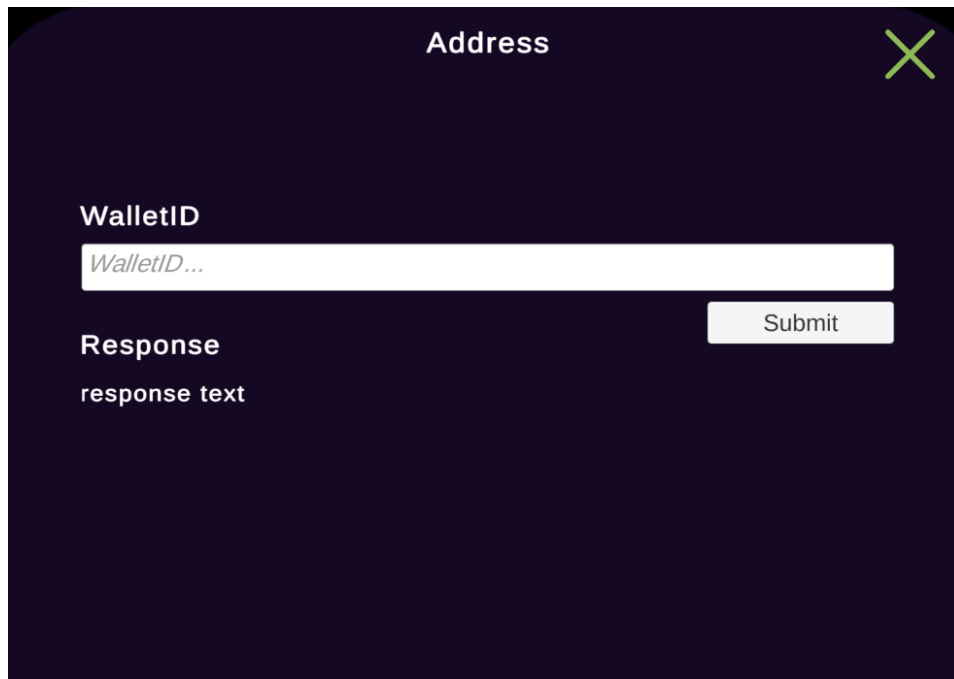
*If you want to learn more about manually extracting funds, we got you covered.
Visit the [Satolearn.com](https://satolearn.com) platform to learn more about bitcoin coding.*

Addresses

Get your address to receive your first BitcoinSV, token or NFT.

Get your Wallet Address

An essential feature of every wallet is the wallet address, which can be either a **classical bitcoin address** or the more modern **paymail alias** address. To ensure privacy for the user, each function call will return a different classical address.



```
Tegment.Wallet.Address.GetAddress(walletID.text, TegmentSessionHandler.Instance._authToken, AddressCallBack, true);
```

Addresses are public, you can share them with anyone.

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_walletID	string	yes	
2	_authToken	string	yes	
3	Callback	function	yes	
4	enableLog	bool	no	

All Wallet Addresses

In case your application requires to get all available addresses, use the code below. This might be useful if you want to manage your own address usage flow.

All Address

WalletID

WalletID...

Submit

Response

response text

```
Tegment.Wallet.AllAddresses.GetAllAddress(walletID.text,  
TegmentSessionHandler.Instance._authToken, AllAddressCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_walletID	string	yes	
2	_authToken	string	yes	
3	Callback	function	yes	
4	enableLog	bool	no	

Paymail vs Classical Addresses

Paymail aliases enable instant p2p transactions with close to no waiting time. They are operating at reduced resource cost and lead to a generally better user experience due to their event based triggering. Classical transactions on the other hand will only be received at your wallet when the UTXO was found on the blockchain, and our infrastructure carefully compares all addresses against all transactions on the network.

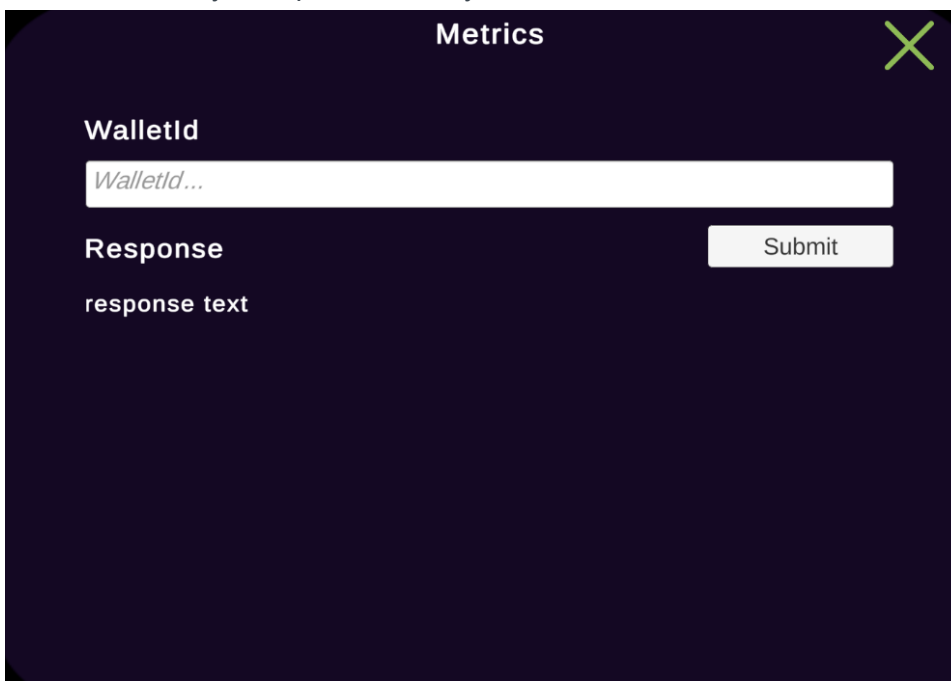
But as always, there are more details to it. Classical transactions give users security that the funds are received exactly at that address, and no server incident based rerouting happened. Moreover, it is easier to prove that you send someone funds if the user gave you exactly that address to receive your funds (in case that ever stands up to dispute).

We recommend paymail transactions for smaller transactions (or infrastructure internal transactions) and classical transactions higher transaction amounts to potentially untrusted 3rd parties.

Metrics

Check or Re-Sync your UTXO state

An essential part of managing a wallet is UTXO state management. The Relysia infrastructure is a native peer to peer system that continuously keeps track of its own UTXOs. Since we support classic addresses and transactions from external parties, we automatically run transaction listeners to always keep wallets in sync with the blockchain.



```
Tegment.Utility.Metrics.MetricsDetails(walletId.text,  
TegmentSessionHandler.Instance._authToken, MetricsResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_walletID	string	yes	
2	_authToken	string	yes	
3	Callback	function	yes	
4	enableLog	bool	no	

You can force refresh the UTXO state by calling the metrics function

When refreshing the UTXO state less than 15 seconds after the last transaction was submitted, it is likely that miners (such as TAAL) haven't indexed the current state. A force refresh would thus sync an out of date UTXO and it is therefore not advisable to regularly force refresh the UTXO state.

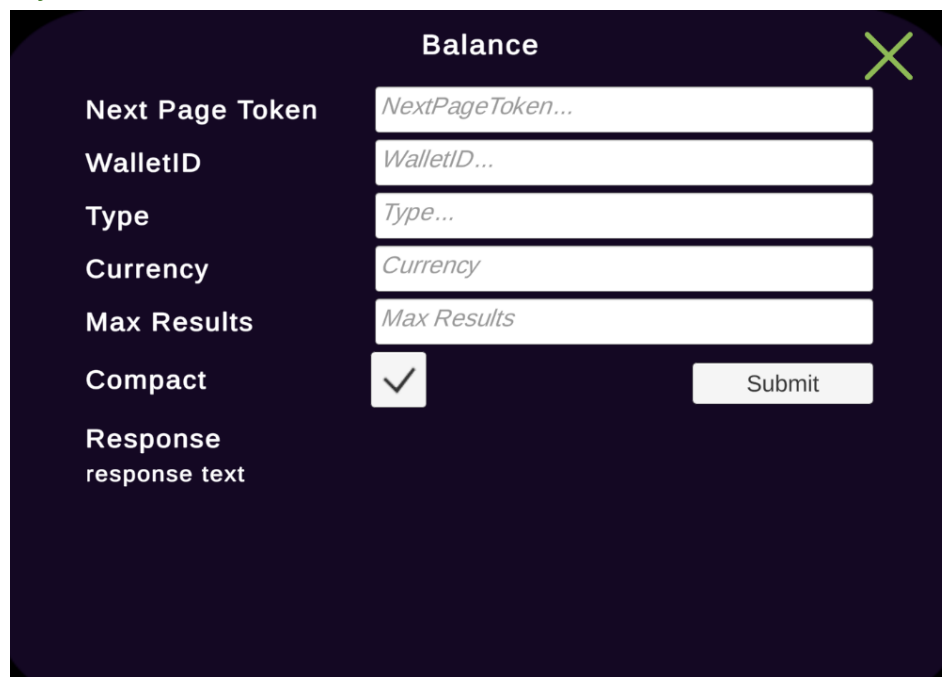
Balance

How many currencies and NFTs do you own?

Looking at the most common platforms, users often have a mix of BSV coins, a variety of token balances, the Personal NFT collection and of course want to know the equivalent US dollar balance. Many systems make it just a tick too complicated and that's why we put them all together in one function. With standardized format Tegment SDK can handle any kind of token with just the same function. So when you check your balance and then decide to directly use the “send” function ascended away, that works too. Amazingly, even without any BSV in your account that most other wallets would require.

After the address, the second most essential feature of wallets is the display of the current wallet **balance**. Next to the standard inputs, you can define a fiat **currency** display of the balance.

The “balance” function returns the BSV balance and fiat equivalent and STAS token balances of your wallet.



```
Tegment.Wallet.Balance.GetBalanceData(nextPageToken.text,walletID.text,_type.text,currency.text,compact.isOn.ToString().ToLower(),maxResultVal,TegmentSessionHandler.Instance._authToken, BalanceCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_nextPageToken	string	no	
2	_walletID	string	yes	
3	_currency	string	no	
4	_compact	string	no	
5	_maxResults	int	no	
6	_authToken	string	yes	
7	Callback	function	yes	
8	enableLog	bool	no	

History

Check what transactions you made in the past, or export it.

The transaction history displays all deposits and withdrawals from the user wallet. When building applications, try to keep the data as shallow as possible to reduce the data fetched. We added filter parameters to reduce the number of entries per page or pick specific coin types that are referenced by their respective tokenID.

We currently integrate token transactions into the “history function.”

History

Next Page Token

TokenId

WalletID

Type

NextPageToken...

TokenId...

WalletID...

Type...

Submit

Response

response text

```
Tegment.Wallet.History.GetHistory(nextPageToken.text, tokenID.text,
walletID.text, _type.text, TegmentSessionHandler.Instance._authToken,
HistoryCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_nextPageToken	string	no	
2	_tokenId	string	no	
3	_walletID	string	yes	
4	_type	string	no	
5	_authToken	string	yes	
6	Callback	function	yes	
7	enableLog	bool	no	

Lookup

How to convert address formats to sensible inputs

A Uniform Resource Identifier

In Bitcoin, there are over 30 combinations sharing address formats. Some include **memos**, which are short messages to the receiver, others include payment **requests** of specified amounts.

You sit at a bar and someone says, look here, I got a QR code. You scan it and it fails. Not because your app is faulty, but rather because it doesn't recognize the format. With Tegment, we tried to build a robust system from the ground up. And that's why we integrated a Uri system. Uri stands for Uniform Resource identifier. That means it can take a variety of different address formats and return to you. What's insight in a standardized way? Using the "URI" function, you don't have to worry about ever not being able to scan a QR code with your app and not knowing what to do with it. So far, we support address paymails BIP 21, BIP 272, BIP 275 and BIP 282.

*The "lookup" function resolves a variety of address formats such as **63@relysia.com** or **1McLoHnGZnXBinS82dqPnRDV9VQJjJpBZF** and it automatically resolves and returns the underlying information.*

Since p2p paymails resolve differently than normal paymails, invoices can sometimes be just a web link instead of an address. To make it easier for app developers, we created a "URI" function to catch them all, no matter the format. This function enables developers to scan QR codes without worrying about ever not having the right resolver at hand.

Let's have a closer look at what addresses can look like:

Address Format variations

//// address ////

19KXpbJMt1SjUNh3ATMgj8FkzvSX7C83gq

//// bip21 ////

bitcoin:19KXpbJMt1SjUNh3ATMgj8FkzvSX7C83gq?amount=10000

//// paymail ////

payto:90@relysia.com?purpose=cashback&amount=10000

//// paymail-noScheme ////

90@relysia.com

//// bip272 ////

bitcoin:?sv=&r=https%3A%2F%2Fstaging.centi.ch%2Fpayment%2Fapi%2Fpayment_request%2F2662e521-ff52-418a-b7e5-c98aefd7295a

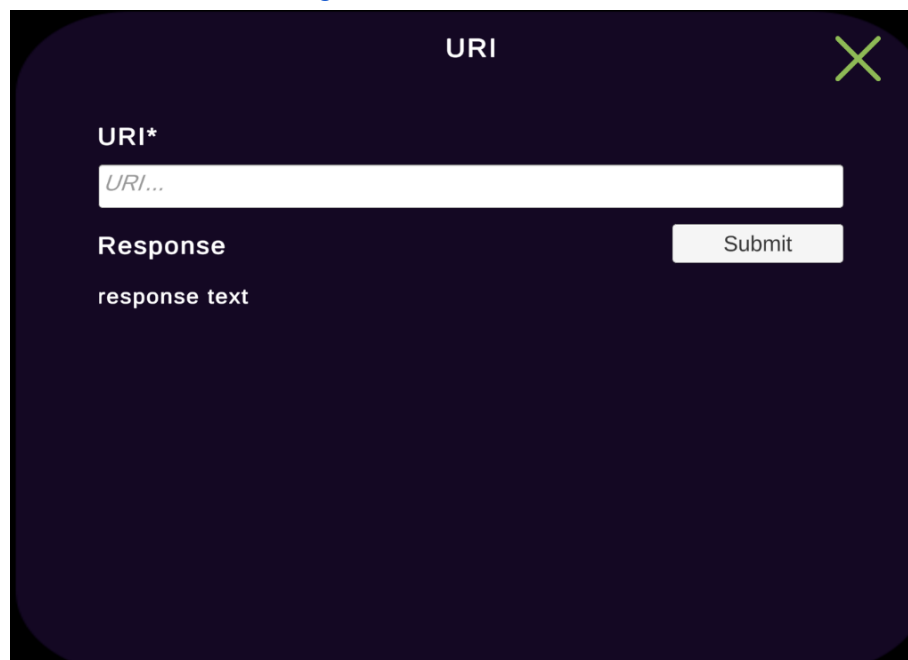
//// DPP ////

pay:?r=https://api.relysia.com/v1/paymentRequest/8f9f4c27-3782-46e2-b76f-1ef956f52c61

URI

As you can see above, a unified resolver reduces the uncertainty of interpreting QR codes and address inputs. Instead of enforcing a single format, the URI lookup function helps to enhance app user friendliness and compatibility with the greater ecosystem.

The result body of the URI can be used as input for the “pay” function (for invoices) or simply be used to decode the string of a QR code.



The screenshot shows a mobile application window with a dark background. At the top, the title 'URI' is centered, and a green 'X' icon is in the top right corner. Below the title, there is a text input field labeled 'URI*' with a light gray border and placeholder text 'URI...'. Underneath the input field, the word 'Response' is displayed in a bold, light gray font. To the right of 'Response' is a white rectangular button with the text 'Submit'. Below the 'Response' label, the text 'response text' is visible in a light gray font.

```
Tegment.Utility.URI.GETURI(_uri.text, URIResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_URI	string	yes	address
2	_authToken	string	yes	
3	Callback	function	yes	
4	enableLog	bool	no	

The result body schema looks as follows, independent of the input.

```
{
  "statusCode": 200,
  "data": {
    "status": "success",
    "msg": "Operation completed successfully",
    "data": {
      "uri": "90@relysia.com",
      "type": "paymail",
      "mainProtocol": "paymail",
      "outputs": [
        {
          "script":
            "76a9149196f6ef3234d17080ebf16460c502c21157125988ac",
          "satoshis": 10000
        }
      ],
      "inputs": [],
      "memo": "Send to 90@relysia.com",
      "isBSV": true,
      "peer":
        "https://api.relysia.com/v1/bsvalias/receive-transaction/90@relysia.com",
      "peerData": "il64psehd1a",
      "peerProtocol": "paymail"
    }
  }
}
```

Transactions

Make a transaction, of any kind or type.

Sending transactions is essential for every wallet. We therefore designed the send function to be easy for those who send BSV and those who send other token formats such as Stas, Tokens

and NFTs. For standard transactions, you just need to specify the amount and the address or Paymail. In case you want to transact a token, you need to add the token ID. While all other inputs stay the same, you might wonder, how do I send the number five of my new NFT collection? Don't worry. Just add the serial number short SN to the input, and we selectively send that NFT. As you might have noticed, the input format is an array. You currently can send BSV and Token to up to 250 receivers in a single function call to distribute your valuables in just a few seconds.

Transactions are a significant aspect of every wallet interaction. By default, we try to transact every transaction bundle in a p2p fashion to reduce computational load and reduce latency. Classical address transactions will be long term supported for legacy reasons.

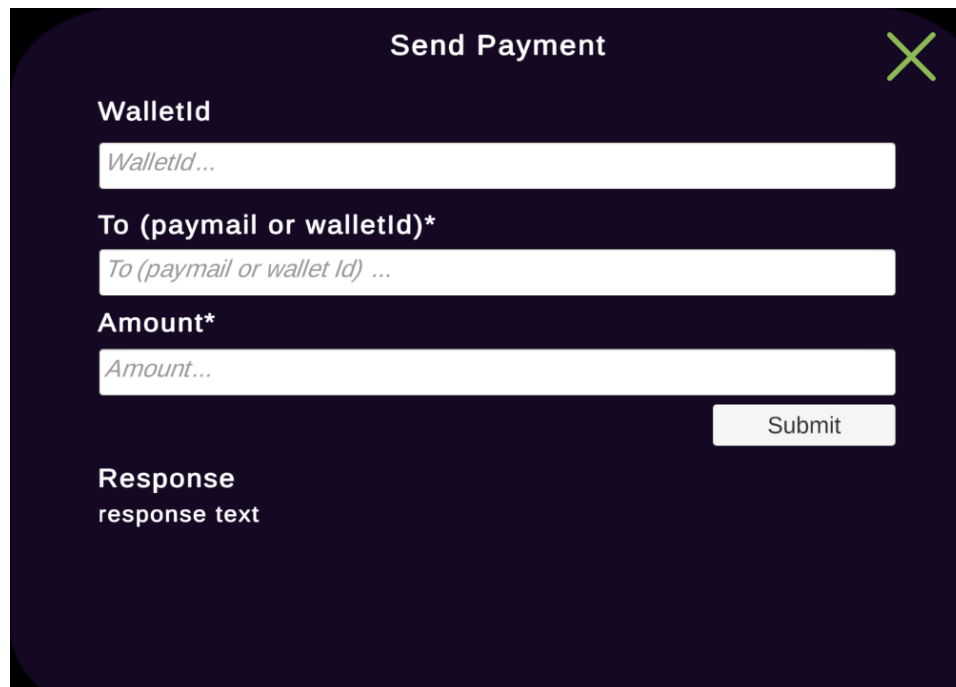
Send Transaction

You can also send transactions without the URI function, by simply composing an array of receiver, amount and type. The note is optional and is sent with the p2p bundle as a message. Since the input is an Array, you can indicate many receivers at the same time.

The receiver indicates it can be a paymail alias or classical address.

Non Bundle Transactions

For BSV transactions there is also the option to create separate transactions for each element in the dataArray. By adding in the bundle field and setting it to false it will then create a transaction for each entry in the data array. Note this is only valid for BSV type transactions. See Multi Output Non Bundle tab for an example.

A dark-themed user interface for sending a payment. At the top, it says "Send Payment" with a close button (X) on the right. Below this are four input fields: "WalletId" with a placeholder "WalletId...", "To (paymail or walletId)*" with a placeholder "To (paymail or wallet Id) ...", and "Amount*" with a placeholder "Amount...". To the right of the "Amount*" field is a "Submit" button. Below the input fields is a section labeled "Response" with the text "response text" underneath it.

Here are few example of Send transaction

1. BSV

```
Tegment.Transaction.Send.SendAmount(walletId.text, sendRequestDataArray,  
TegmentSessionHandler.Instance._authToken, SendPaymentCallBack,true);
```

```
serviceId: 'optional',  
walletID: 'optional',  
data: {  
    dataArray: [  
        {  
            to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",  
            amount: 0.0001,  
            notes: "pizza payment"  
        }  
    ]  
}
```

2. Token

```
serviceId: 'optional',  
walletID: 'optional',  
data: {  
    dataArray: [  
        {  
            to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",  
            tokenId: "1SJDKAS34DKSDKD1OD-SBD",  
            amount: 2  
        }  
    ]  
}
```

3. NFT

```
serviceId: 'optional',  
walletID: 'optional',  
data: {  
    dataArray: [  
        {  
            to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",  
            tokenId: "1SJDKAS34DKSDKD1OD-SBD",  
            amount: 1,  
            sn: 1 //optional  
        }  
    ]  
}
```

4. Multi Output

```
serviceId: 'optional',
```

```

    walletID: 'optional',
    data: {
      dataArray: [
        {
          to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
          amount: 0.00001,
          notes: "pizza payment"
        },
        {
          to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
          amount: 0.00001,
          notes: "grocery"
        },
        {
          to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
          amount: 0.0005
        }
      ]
    }
  }

```

5. Multi Output Non Bundle

```

    serviceId: 'optional',
    walletID: 'optional',
    data: {
      bundle : false,
      dataArray: [
        {
          to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
          amount: 0.00001,
          notes: "pizza payment"
        },
        {
          to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
          amount: 0.00001,
          notes: "grocery"
        },
        {
          to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
          amount: 0.0005
        }
      ]
    }
  }

```

Other than just BSV, you can send an arbitrary token (such as a NFT or festival ticket token). The fee manager is handling all transaction fees for the user (up to infrastructure provider determined limits) to improve the general onboarding and user experience.

Data Append (STAS-789)

With the STAS-789 tokens, it is possible to include extra data in the token script during a transaction. To facilitate this feature, we have introduced a new field in the "send" function named "data", which is an array format. Each element of this array must be a string, and each string will be added as individual hexadecimal chunks of data that can be segmented while in ASM script format. The STAS-789 tokens are formatted as NFTs and will each have a unique serial number value, denoted as "sn" in the corresponding field.

```
    serviceId: 'optional',
    walletID: 'optional',
    data: {
      dataArray: [
        {
          to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
          tokenId: "1SJDKAS34DKSDKD1OD-SBD",
          amount: 1,
          sn : 1,
          data : ["some data1", "some data2"]
        }
      ]
    }
  }
```

Raw Transactions

Create an unbroadcast raw transaction

Once a transaction is created using the "send" function and broadcasted to the network, there may be situations where access to the raw transaction data before broadcasting is necessary. To address this use case, the "rawtx" function becomes relevant. By utilizing the "rawtx" function, you can generate one or multiple transactions and retrieve the un-broadcasted raw transaction data. This data can be employed for alternative processes as needed.

The transactions generated through the "rawtx" function are constructed in their pure form, devoid of any change outputs that might be required for subsequent transactions. It's important to understand that any change outputs included in the transactions are not considered valid on the network until the previous transaction has been successfully broadcasted.

Similar to the "send" function, the "rawtx" function allows you to create transactions using the same input parameters in the request body. This facilitates consistency and familiarity in the process of generating transactions between the two functions.

Raw Transaction

WalletId

WalletId...

To (paymail or walletId)*

To (paymail or wallet Id) ...

Amount*

Amount...

Submit

Response

response text

Here are few example of RawTx transaction

1. BSV

```

serviceId: 'optional',
walletID: 'optional',
data: {
  dataArray: [
    {
      to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
      amount: 0.0001
    }
  ]
}

```

2. Token

```

serviceId: 'optional',
walletID: 'optional',
data: {
  dataArray: [
    {
      to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
      tokenId: "1SJDKAS34DKSDKD1OD-SBD",
      amount: 2
    }
  ]
}

```

3. NFT

```

serviceId: 'optional',
walletID: 'optional',

```

```

data: {
  dataArray: [
    {
      to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
      tokenId: "1SJDKAS34DKSDKD1OD-SBD",
      amount: 1,
      sn: 1 //optional
    }
  ]
}

```

4. Multi Output

```

serviceId: 'optional',
walletID: 'optional',
data: {
  dataArray: [
    {
      to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
      amount: 0.00001
    },
    {
      to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
      amount: 0.00001
    },
    {
      to: "17n2JVhrCf1oYSMkZtZNjcf1deteUEKQsH",
      amount: 0.0005
    }
  ]
}

```

The rawtx result is a valid signed transaction that can be used in conjunction with paying invoices or awaiting other logic before broadcasting it to the network.

Invoices

Create a BitcoinSV or Token Invoice

When buying goods in supermarkets or online shops, customers are not asked to type in the exact amount to spend from their credit card, instead, they just get to see a fixed amount that they can decide to pay or not pay. We call systems that pre-determines an amount and requests payments "invoice systems"

The advantage of an "invoice" based system is that there is no room for failure outside payment success or decline.

The structured process to create invoices is called BIP270 and is adopted by a variety of wallets such as SimplyCash, Centi or Handcash. An invoice created via the Tegment SDK can thus be understood by a variety of wallets and be universally used for BSV gateway systems.

DPP (Direct Payment Protocol) is an expanded version of the BIP270 process that allows payment invoices to include Stas tokens and additional data for the beneficiaries of the payment. The "modelId" value of "ef63d9775da5" is a standard identifier for DPP payments that is an accepted data format for the invoice. The "native" or "stas" objects can be used to specify the desired transaction type and amount. The "amount" value in the "native" object is used to specify an amount in BSV.

Whenever selling a product for BSV or Stas token, the invoice system is the system of choice.

In the next sections we will look at how to create the different types of invoices, how to resolve them and then ultimately how to settle them.

Create Invoice

In this section, we will explore the various methods for generating invoices, namely the DPP and BIP270 processes, each with its own set of input fields. It's important to note that while DPP enables the creation of invoices using Tokens, BIP270 is specifically designed for generating BSV-type invoices.

Below, you will find examples of the invoice input bodies for both DPP and BIP270. For DPP, it is important to select and include either the "native" or "stas" section in the invoice input body, but does not support both at the same time.

DPP

The abbreviation "DPP" stands for Direct Payment Protocol, which is utilized for generating invoices that encompass BSV (Bitcoin SV) or STAS tokens. Note that the payment options cannot include both native and stas and only one is required for the invoice to be created. The following is the "invoice" request example to be employed when utilizing DPP:

```
"expirationTimeInMinuts": 0,
"memo": "string",
"paymentOptions": [
  {
    "transactions": [
      {
        "native": {
          "amount": 0,
          "to": "string"
        },
        "stas": {
          "tokenAmount": 0,
          "tokenRecipient": "string",
          "tokenId": "string"
        }
      }
    ]
  }
]
```

```

    }
  ],
  "modeId": "ef63d9775da5",
  "beneficiary": {
    "name": "string",
    "email": "string",
    "address": "string",
    "paymentReference": "string"
  }
}

```

Following table mentions about the parameters used in above function

Sr. No.	Param Name	Type	IsRequired	details
1	expirationTimeInMinuts	int	yes	Number of minutes the Invoice is valid for
2	memo	string		Notes about the transaction
3	paymentOptions	Class Array	yes	Either Native BSV or STAS token
4	modeId	string		Specific identifier for the DPP protocol format

				used
5	beneficiary	Class Object		Information about the merchant receiving the transaction
6	Callback	function	yes	
7	enableLog	bool	no	

BIP270

BIP270 is a widely recognized invoice standard extensively adopted within the BSV (Bitcoin SV) ecosystem. When utilizing BIP270, the following are the inputs required for the "invoice" function:

```
"type": "string",
"amount": 0,
"address": "string",
"merchantData": "string",
"description": "string",
"expirationTimeInMinuts": 0,
"memo": "string"
```

Invoice

Type

Amount

Address

Description

ExpirationTime

Memo

Merchant Data

Payment Options

Transactions

Native

Amount

To

Type...

Amount...

Address...

Description...

ExpirationTime in Minutes...

Memo...

Merchant Data...

Amount...

To

Following table mentions about the parameters used in above function

Sr. No.	Param Name	Type	IsRequired	details
1	type	int	no	Supported Currencies such as "USD" , "BSV", etc...
2	amount	double	yes	amount in currency type
3	address	string	yes	receiving address for the payment
4	description	string	no	Description about the transaction
5	expirationTimeInMinuts	int	yes	Number of minutes the Invoice is valid for
6	memo	string		Notes about the transaction
7.	merchantData	classObject		Information about the receiving party/merchant
6	Callback	function	yes	
7	enableLog	bool	no	

Get Invoice

Resolving an invoice to make a payment

Once an invoice is created, it is essential to provide a means for another party to receive the invoice and proceed with the payment. The result from the "invoice" function, typically in JSON format, can be utilized to extract the relevant information required for this process. There are 2 ways to resolve an invoice:

- Using an internal invoice ID query for any invoices that may be stored locally.
- Using the URL extension in the "URI" function that can be resolved externally.

Get Invoice by paymentRequest invoice function

In this method simply pass in the invoice Id value returned from the create invoice step which will return an invoice JSON related to that service ID.

Payment Request

Invoiceld*

Invoiceld...

Response

Submit

response text

```
Tegment.Transaction.PaymentRequest.GetPaymentRequest (invoiceID.text,
PaymentRequestV1TransactionCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_invoiceld	string	yes	invoiceld received from create invoice
2	Callback	function	yes	
3	enableLog	bool	no	

Get Invoice by URI function

In this method, invoices can be obtained by utilizing a “URI” function with a specific URL extension. This approach serves as a way to externally receive invoices from the system. Here is an example of the URL extended format that is used in the “URI” function to resolve the invoice and can be found in the "uri" field when creating the invoice.

The screenshot shows a mobile application window with a dark background. At the top, the title 'URI' is centered, and a green 'X' icon is in the top right corner. Below the title, there is a text input field with the placeholder text 'URI...'. To the right of the input field is a white button with the text 'Submit'. Below the input field, the word 'Response' is displayed in a bold font. Underneath 'Response', the text 'response text' is visible, indicating where the response will be shown.

```
Tegment.Utility.URI.GETURI(_uri.text, URIResponseCallBack, true);
```

Example uri=

```
"pay:?r=https://api.relysia.com/v1/paymentRequest/8f9f4c27-3782-46e2-b76f-1ef956f52c61";
```

Following is the example result of both type of request

1. DPP

```
{
  "statusCode": 200,
  "status": "success",
  "msg": "Operation Completed Successfully",
  "network": "mainnet",
  "uri":
"pay:?r=https://api.relysia.com/v1/paymentRequest/8f9f4c27-3782-46e2-b76f-1ef956f52c61",
  "memo": "paying for testing",
  "type": "bip272",
  "paymentUrl":
"https://api.relysia.com/v1/paymentRequest/pay/8f9f4c27-3782-46e2-b76f-1ef956f52c61",
  "mainProtocol": "bip272",
  "creationTimestamp": 1686814941,
  "modes": {
    "ef63d9775da5": {
      "PaymentOption_0": {
        "transactions": [
          {
```

```

        "outputs": {
          "stas": [
            {
              "tokenId":
"00b91626e0a4b97f624bc1f0d8fa3a3ef35ac664-TIGLdA",
              "tokenAmount": 1,
              "tokenRecipient": "1HQNj5FfCjcgvr6AAqNPYhY7NKuZXByMF"
            }
          ]
        },
        "policies": {
          "fees": {
            "standard": {
              "bytes": 1000,
              "satoshis": 50
            },
            "data": {
              "bytes": 1000,
              "satoshis": 50
            }
          }
        }
      }
    ]
  }
},
"beneficiary": {
  "name": "vaionex test",
  "email": "test@vaionex.com",
  "paymentReference": "8f9f4c27-3782-46e2-b76f-1ef956f52c61",
  "address": "19702 Newark, Delaware, USA"
},
"expirationTimestamp": 1686816741,
"version": "2.0.0"
}

```

2. BIP270

```

{
  "statusCode": 200,
  "status": "success",
  "msg": "Operation Completed Successfully",
  "mainProtocol": "bip272",
  "merchantData": "a3621ebf-97a4-476f-a028-a7a9c55b27e8",
  "expirationTimestamp": 1686560853,
  "memo": "paying for testing",
  "outputs": [

```

```

    {
      "description": "paying 1$ for testing",
      "script": "76a9140bb9eb13c821d10af1f5fbf01c04577305c24d3788ac",
      "satoshis": 3971132
    }
  ],
  "version": "2.0.0",
  "uri":
"pay:?r=https://api.relysia.com/v1/paymentRequest/f3a7f350-2e2e-4e8b-af0d-58eb5eb3728a",
  "creationTimestamp": 1686560793,
  "type": "bip272",
  "network": "mainnet",
  "paymentUrl":
"https://api.relysia.com/v1/paymentRequest/pay/f3a7f350-2e2e-4e8b-af0d-58eb5eb3728a"
}

```

These results can be directly used to make a payment. In the next section we will see how to use these result values.

Pay Invoice

Make a payment on an invoice

As we seen in the earlier section about invoices, there are advantages to pre-determine request amounts. From an infrastructure perspective, it is advantageous to create invoice-like schemas even if no invoice was given to exactly determine what script to use to make certain payments.

The “pay” function takes up “lookup” results formats and triggers a payment. While that might sound a bit complex, we can imagine it as a standardization machine that can take up a variety of formats without worrying about their compatibility.

Similar to the get invoice methods, there are also two ways to pay for an invoice. This includes an internal method and external method.

Whereas the payment function is essential to complete invoice requests, we created utility wrappers around the “pay” function, such as the “send” function that lets you define address, type and amount to make transactions in a simpler format.

Pay Invoice using by URI (external)

To make an invoice payment, take the result of the “URI” function (the **data**) without any additional changes to pay a payment invoice.

DPP using “pay”

Pay

WalletId

URI

Type

MainProtocol

Outputs

Script

Satoshis

Inputs

TxID

VOut

Satoshis

ScriptPubKey

WalletId...

URI...

Type...

MainProtocol

Script...

Satoshis...

TxID

vout...

Satoshis...

ScriptPub Key...

```
Tegment.Transaction.Pay.PayTransaction(URI.text, _type.text,
mainProtocol.text, payOutputRequestArray, payInputRequestsArray,
network.text,paymentUrl.text, creationTimeVal, expirationTimeVal,
memo.text, isBSV.isOn.ToString().ToLower(),peer.text, peerData.text,
peerProtocol.text, walletID.text,
TegmentSessionHandler.Instance._authToken, PayTransactionCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_uri	string	yes	
3	_type	string	yes	
4	_mainProtocol	bool	no	
5	_Outputs	ClassObject Array		
6	_Inputs	ClassObject Array		
7	_network		yes	
8	_paymentURL			
9	_creationTimeStamp	int		

10	_expirationTimeStamp	int		
11	_memo	string	yes	
12	_isBSV	string		
13	_peer	string		
14	_peerData	string		
15	_peerProtocol	string		
16	_walletID	string		
17	_authToken	string		
18	callback	function		
19	enableLog	bool		

BIP270 using “pay”

This is an example of paying for a BIP270 invoice using the “pay” function. Please check in Get Invoice to get extended values for the function input using BIP270.

Pay

WalletId

URI

Type

MainProtocol

Outputs

Script

Satoshis

Inputs

TxID

VOut

Satoshis

ScriptPubKey

WalletId...

URI...

Type...

MainProtocol

Script...

Satoshis...

TxID

vout...

Satoshis...

ScriptPub Key...

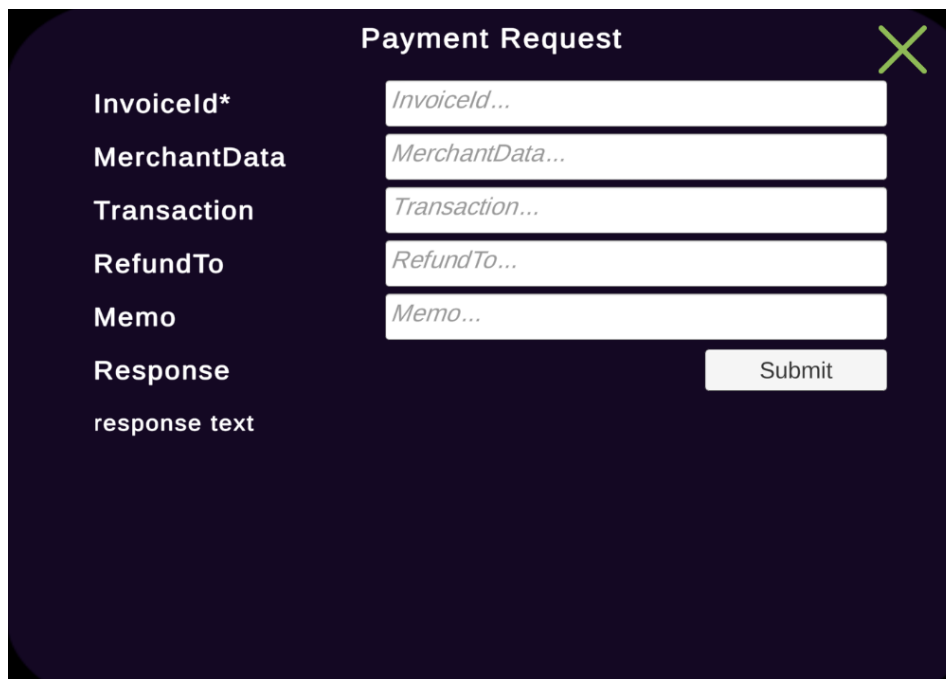
```
Tegment.Transaction.Pay.PayTransaction (URI.text, _type.text,
mainProtocol.text, payOutputRequestArray, payInputRequestsArray,
network.text, paymentUrl.text, creationTimeVal, expirationTimeVal,
memo.text, isBSV.isOn.ToString().ToLower(), peer.text, peerData.text,
peerProtocol.text, walletID.text,
TegmentSessionHandler.Instance._authToken, PayTransactionCallBack, true);
```


Pay Invoice using payment-request Invoice (internal)

This method involves constructing a transaction using the “RawTx” function and using it as a parameter in another function to associate it with an internal invoice ID for payment confirmation.

Whether DPP or BIP270 is used, the process remains consistent. The “RawTx” function is used to create the transaction, and its data is passed as a parameter in the relevant function to link it with an internal invoice ID.

Regardless of the invoice standard (DPP or BIP270) being utilized, the transaction construction and referencing process remains the same, ensuring compatibility and consistency across both methods.

A screenshot of a "Payment Request" form. The form has a dark blue header with the title "Payment Request" and a green close button (X) in the top right corner. Below the header, there are five input fields with labels on the left: "InvoiceId*", "MerchantData", "Transaction", "RefundTo", and "Memo". Each input field contains a placeholder text: "InvoiceId...", "MerchantData...", "Transaction...", "RefundTo...", and "Memo...". Below these fields is a "Response" label and a "response text" label. A "Submit" button is located to the right of the "Response" label.

```
Tegment.Transaction.PaymentRequestPay.SettleInvoice(invoiceID.text,merchantData.text,transaction.text,refundTo.text, memo.text,PaymentRequestPayTransactionCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_invoiceId	string	yes	Invoice ID to be paid
3	_merchantData	string	yes	merchant information
4	_transaction	string	yes	Transaction hexadecimal value
5	_refundTo	string	yes	Address for refund if required memo

6	_memo	string		Description about the transaction
7	callback	function	yes	Call back function
8	enableLog	bool		Enable logging if required

Payments

How to make general payments

To make a payment it requires an input, and rather than the requirement to build this input from scratch we can utilize the “URI” function as described in “lookup”. We can use the “URI” function as a way to construct these payment JSON values in a simplistic way.

Here is an example of creating a payment request using the “URI” function

```
Tegment.Utility.URI.GETURI(_uri.text, URIResponseCallBack, true);
```

_uri example

```
uri= "payto:90@relysia.com?purpose=cashback&amount=10000";
```

This will be the result which is a standard payment format.

```
{
  "statusCode": 200,
  "data": {
    "status": "success",
    "msg": "Operation completed successfully",
    "data": {
      "uri": "payto:90@relysia.com?purpose=cashback&amount=10000",
      "type": "paymail",
      "mainProtocol": "paymail",
      "outputs": [
        {
          "script": "76a914e2a15b75da46234f5e5943b1289b955842532ce088ac",
          "satoshis": 10000
        }
      ],
      "inputs": [],
      "memo": "cashback",
      "isBSV": true,
      "peer":
"https://api.relysia.com/v1/bsvalias/receive-transaction/90@relysia.com",
      "peerData": "2zwwsow4bfg",
      "peerProtocol": "paymail"
    }
  }
}
```

This can then be used to facilitate general payments. The advantages include using a simple format provided by the “URI” function, which allows for the retrieval of a comprehensive JSON schema conforming to widely accepted industry standards within the ecosystem. Alternatively, these formats can also be constructed from scratch if necessary.

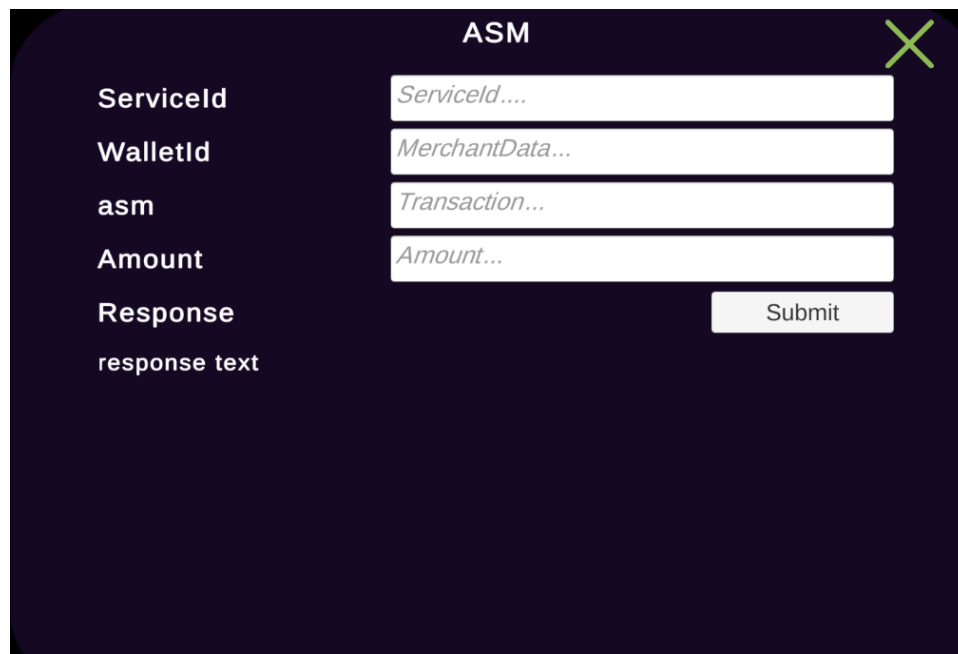
Asm

Create a Asm script transaction

Asm transaction:

With this function, users can create a transaction by passing a custom Asm script as input. This function accepts an Asm script and an amount of transfer as input in the function body. It sends an amount of BSV to the target destination.

Asm doesn't support Token or NFT transfer, It only supports BSV transfer.

A screenshot of a web form titled "ASM" with a green close button in the top right corner. The form has a dark blue background. It contains four input fields: "ServiceId" with placeholder text "ServiceId....", "WalletId" with placeholder text "MerchantData...", "asm" with placeholder text "Transaction...", and "Amount" with placeholder text "Amount...". Below these fields is a "Submit" button. At the bottom left, there is a label "Response" and a text area labeled "response text".

```
Tegment.Transaction.ASM.CreateASM(serviceID.text,walletID.text,ASM.text,amountVal, TegmentSessionHandler.Instance._authToken,asmRequestCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_serviceld	string		
2	_walletId	string		
3	_asm	string	yes	
4	_amount	double	yes	
5	_authToken	string	yes	
6	Callback	function	yes	
7	enableLog	bool		

Basics

Relysia Infrastructure and Tokens

Relysia was built from the ground up with token support in mind. You can let your users mint and share tokens without them requiring to own any BitcoinSV base currency.

As a platform, we focus on a portfolio of reliable and stable protocols to ensure the best experience for both developers and their users. In our initial release, we thus launched on STAS tokens that have the unique ability to only require the execution of bitcoin script smart contracts.

Similar to artwork, NFTs and other collectibles may require decade-long stability of their underlying token protocols. We at Relysia use inherently stable protocols to ensure ownership and the individual faith of companies are never connected.

STAS Token

STAS tokens are native bitcoin script tokens that get directly enforced by smart contracts. One of their main benefits over existing protocols is their long term stability and maintenance free functioning.

Smart contracts on the Bitcoin blockchain (written in script) are very similar to normal transactions, with the difference that they can only be transferred if certain conditions are met. Since they are normal transactions, they have the same limitations as a standard bitcoin transaction. The advantage is that a **STAS token natively scales** with improvements of the underlying network.

A transaction (and thus UTXO) can only be created with at least one satoshi within the STAS contract. When creating a token with 1 million supply, it will require 1 million satoshis.

STAS Types

The STAS protocol is versatile, capable of accommodating both NFTs and Tokens through configuration. The meta field associated with STAS tokens is fully customizable, enabling developers to input any necessary JSON data. In schema, the metadata format can be defined as either standard or non-standard. A standard metadata format is NFT1.0, which includes a media field containing a B protocol link, media type specified in MIME format, and altURI (a classical image URL).

A crucial distinction lies in the value of "splittable," which determines whether STAS tokens can be split ($2 \rightarrow 1 + 1$) and merged ($1 + 1 \rightarrow 2$), or not ($1 \rightarrow 1$). For NFTs, splittable should be set to false, resulting in serialization such as 1/100, 2/100, and so on.

STAS redemption

As previously established, each STAS UTXO smart contract requires a certain number of satoshis. However, when a token is no longer needed, the satoshis used in its creation are not lost; they can be released by redeeming the STAS smart contract, essentially dissolving the token. STAS tokens are redeemed by sending them back to their contract issuance address that is contained within every STAS token script.

Token Issuance

Create a STAS token

Creating a token requires you to first create a 'default' wallet to issue it to the wallet. If you didn't create one, you can jump back to the wallet section.

The token issuance function takes care of all the rest (including paying the mining fee to mint the token). All you need is to add the parameters to specify the type of STAS token you want.

Parameter	Description	Property
Name	The token name	UTF-8 standard, up to 32 characters in length.
Symbol	The token symbol	1-9 alphanumeric characters
Description	The Token description	up to 512 characters
Image	The mini icon image	250x250 pixels, type image
Token Supply	The amount of tokens to mint in the issuance	Any Integer
Decimals	Decimals indicate the formatting, e.g. decimals 2 for	Integer not bigger than 8

	a dollar token with cents.	
Sats Per Token	Satoshis to be used for each token.	Integer
Splitable	Tokens are split table and merge-able, NFTs are not.	bool
Data	Custom data string(s) to be added to the issued tokens	Object<Array>
Meta	Any extra data to indicate legals and terms.	RFC 3986 JSON formatted

Above stated are the main inputs for each STAS token, the extended schema can be found in below code example. There is only one STAS format but various settings that make it behave according to its purpose and use case.

Issuance function

To issue a STAS token, Following is an example

```

serviceId: 'optional',
protocol: 'optional',
  data: {
    "name": "Bonus Point Token",
    "protocolId": "STAS",
    "symbol": "BPT",
    "description": "A supermarket bonus point token to be used ay
https://yourproject.com",
    "image":
"https://firebasestorage.googleapis.com/v0/b/nftdev/o/nftTemp%2FWL1DdD?alt=media",
    "tokenSupply": 30,
    "decimals": 0,
    "satsPerToken": 1,
    "splitable": true,
    "data" : {"1" : ["custom data 1", "custom data 2"]},
    "properties": {
      "legal": {
        "terms": "STAS, Inc. retains all rights to the token script.
Use is subject to terms at https://stastoken.com/license.",
        "licenceId": "stastoken.com"
      },
      "issuer": {
        "organisation": "vaionex corp.",
        "legalForm": "Limited",
        "governingLaw": "US",

```

```

        "issuerCountry": "US",
        "jurisdiction": "US",
        "email": "info@vaionex.com"
    },
    "meta": {
        "schemaId": "STAS1.0",
        "website": "https://yourProject.com",
        "legal": {
            "terms": "Your personal token terms"
        }
    }
}
}
}

```

Issue_V1

Protocol

Protocol...

Reminting

☐

WalletId

WalletId...

Name

Store Bonus Points

ProtocolId

STAS

Symbol

SBP

Description

A supermarket bonus point.

Image

<https://commons.wikimedia.org/wiki/Catego>

Token Supply

3

Decimals

0

SatsPerToken

1

Properties

```

Tegment.SmartContracts.Issue.MintIssue(_name.text, protocolID.text,
symbol.text, description.text, image.text, tokenSupplyVal, decimalsVal,
satsPerTokenVal,
        terms_Legal.text, licenceID_Legal.text,
organisation_Issuer.text, legalForm_Issuer.text, governingLaw_Issuer.text,
issuerCountry_Issuer.text, jurisdiction_Issuer.text,
        email_Issuer.text, schemaId_Meta.text, website_Meta.text,
terms_Legal_Meta.text, URI_Media.text, type_Media.text, altURI_Media.text,
splitable.isOn, data, protocol.text,
        reMinting.isOn, walletId.text,
TegmentSessionHandler.Instance._authToken, Issue_V1CallBack, true);

```

Custom Data Field for Tokens

Incorporating personalized data into your Tokens can significantly enhance its functionality. The "data" field allows for the inclusion of an object value, which will be assigned as a numbered string key with an accompanying array property. Each element in the array will be added to the token script as its own data chunk.

```
data = {"1" : ["custom data 1", "custom data 2"]};
```

NFT Issuance

Create your first NFT

Non fungible tokens (NFTs) are a way to reward your fans, create collectors, distribute art and declare ownership. NFTs can also be used in extended use as access tokens or tickets to a festival, which enables countless business opportunities. The issuance of NFTs with the STAS protocol is very similar to normal tokens, with an extended metadata format (to store media files) and other freely customizable attributes.

The general issuance format is the same as the STAS token:

Parameter	Description	Property
Name	The token name	UTF-8 standard, up to 32 characters in length.
Symbol	The token symbol	1-9 alphanumeric characters
Description	The Token description	up to 512 characters
Image	The mini icon image	250x250 pixels, type image
Token Supply	The amount of tokens to mint in the issuance	Any Integer
Decimals	Decimals indicate the formatting, e.g. decimals 2 for a dollar token with cents.	Integer not bigger than 8
Sats Per Token	Satoshis to be used for each token.	Integer
Splitable	Tokens are split table and merge-able, NFTs are not.	bool
Data	Custom data string(s) to be added to the issued tokens	Object<Array>
Meta	Any extra data to indicate legals and terms.	RFC 3986 JSON formatted

This function will issue a STAS token in your wallet, depending on the parameters you assign.

NFTs need to be issued with splittable = false tag. While a token (splittable = true) can also be used as NFT, there is a risk of getting merged or split and thus destroying the value of the NFT.

```
serviceId: 'optional',
protocol: 'optional',
data: {
  "name": "Football Card 21",
  "protocolId": "STAS",
  "symbol": "FC21",
  "description": "A FC 21 season nft",
  "image":
"https://firebasestorage.googleapis.com/v0/b/nftdev/o/nftTemp%2FWL1DdD?alt
=media",
  "tokenSupply": 21,
  "decimals": 0,
  "satsPerToken": 1,
  "splittable": false,
  "data" : {"1" : ["custom data 1", "custom data 2"]},
  "properties": {
    "legal": {
      "terms": "STAS, Inc. retains all rights to the token script.
Use is subject to terms at https://stastoken.com/license.",
      "licenceId": "stastoken.com"
    },
    "issuer": {
      "organisation": "vaionex corp.",
      "legalForm": "Limited",
      "governingLaw": "US",
      "issuerCountry": "US",
      "jurisdiction": "US",
      "email": "info@vaionex.com"
    },
    "meta": {
      "schemaId": "NFT1.0",
      "website": "https://football21.com",
      "legal": {
        "terms": "the terms of your nft"
      },
      "media": [
        {
          "URI": "string",
          "type": "string",
          "altURI": "string"
        }
      ]
    }
  }
}
```

```
    ],  
  },  
},  
},
```

Custom Data for NFTs

Incorporating personalized data into your NFT can significantly enhance its functionality. The "data" field allows for the inclusion of an object value, which will be assigned as a numbered string key with an accompanying array property. Each element in the array will be added to the token script as its own data chunk. In this method below it will add the same data to each of the NFTs in a collection

```
data = {"1" : ["custom data 1", "custom data 2"]};
```

Custom Data for NFT collection

When building a collection of NFTs, you have the ability to incorporate distinct custom data into each of them using the following approach. In this method, each object key corresponds to the specific NFT number to which you are attaching the data.

```
"data" : {"1" : ["custom data"], "2" : ["custom data 1", "custom data 2"]},
```

When adding different custom data for each of your NFT collections it is required that the number of keys matches the number of NFTs in the collection.

Atomic Swaps

Exchanging one token against another.

When exchanging tokens, Atomic Swaps are essential to guarantee that both tokens are exchanged instantaneously and without the ability of any party to default on their commitment. Atomic swaps are single transactions that contain partially signed pieces of the transaction from both parties. Only when the transaction piece gets completed by the counterparty, the transaction can get sent to the mining network.

Atomic Swaps either accept the transaction and do the swap or make no transaction at all (thus neither party loses their respective asset). Since the function has no room for one-sided failure, it is ideal to be implemented in any kind of value exchange, since it reduces fraud levels and security risk. Moreover, the exchange can happen in a peer 2 peer fashion, without any additional intermediary other than the exchanging parties.

In the relaysia API, the first atomic swap partial transaction gets created with the "offer" function and later completed with the "swap" function by the counterparty.

The partially signed transaction hex, created by the "offer" function, is shared to the party wanting to complete the transaction (e.g. a buyer of an asset).

Swap Types

There are three swap types that can be used to create an offer. Note that the token quantities will be represented as Integers (without decimal places), while the BSV amounts will be of type Number with native decimal representation (e.g. 0.0000001)

Token for BSV

This type of swap will return an offer hex containing a token input from a user and a desired output amount in BSV. The "sn" parameter is optional and will specify a specific serial number value for an NFT collection.

```
dataArray = [{  
  "tokenId": "d4b5ec377fa847b53b0e08459326817000886af4-USD",  
  "amount" : 10,  
  "wantedAmount" : 0.001  
}];
```

Token for Token

This type of swap will return an offer hex containing a token input from a user and a specified token to swap with. The "sn" and "wantedSn" parameters are optional and will specify a specific serial number value for an NFT collection. Alternatively, the "wantedScript" parameter can be used instead of "wantedTokenId" to specify an entire script hex value of a token, regardless of the owner of that token script.

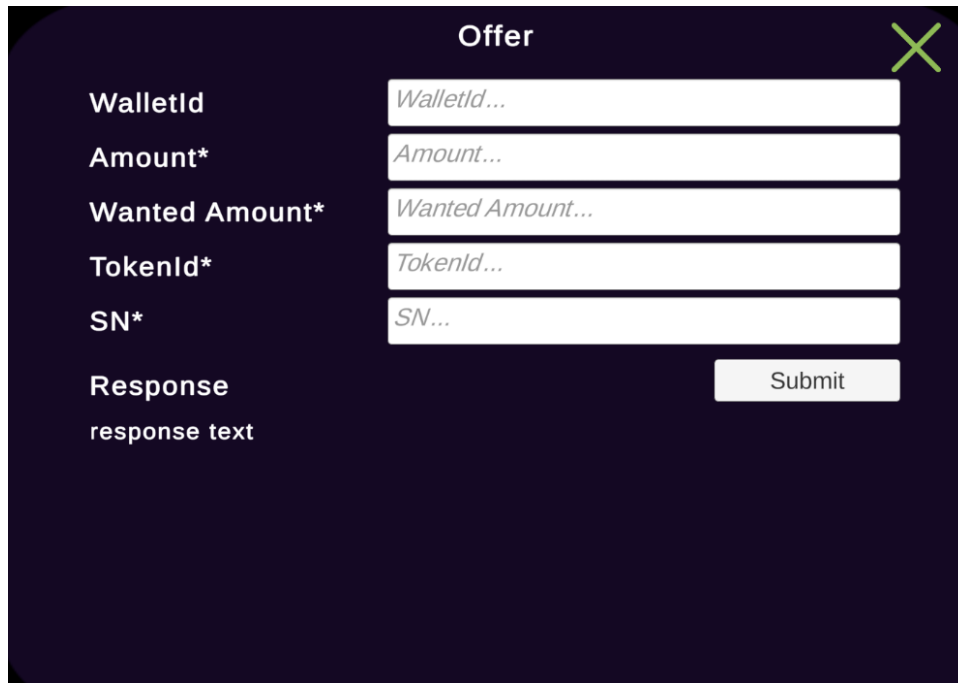
```
dataArray = [{  
  "tokenId": "d4b5ec377fa847b53b0e08459326817000886af4-USD",  
  "amount" : 10,  
  "wantedTokenId" : "ca7a5c64543850de318639334a5aaa65947914cb-MYNFT",  
  "wantedSn" : 1, // optional  
  "wantedAmount" : 1  
}];
```

BSV for Token

This type of swap will return an offer hex containing a specified BSV amount UTXO as an input and a specified token to swap with. Like the previous type, the "wantedSn" parameter is optional and will specify a specific serial number value for an NFT collection. The "wantedScript" parameter can also be used instead of "wantedTokenId" to specify an entire script hex value of a token, regardless of the owner of that token script.

```
dataArray = [{  
  "amount" : 0.001,  
  "wantedTokenId" : "d4b5ec377fa847b53b0e08459326817000886af4-USD",
```

```
"wantedAmount" : 10
}];
```

A screenshot of a mobile application interface titled "Offer" with a green close button in the top right corner. The form contains five input fields: "WalletId" with placeholder "WalletId...", "Amount*" with placeholder "Amount...", "Wanted Amount*" with placeholder "Wanted Amount...", "TokenId*" with placeholder "TokenId...", and "SN*" with placeholder "SN...". Below these fields is a "Response" section with a label "response text" and a "Submit" button.

```
Tegment.Transaction.Offer.OfferTransaction(snVal,tokenId.text, amountVal,
wantedAmountVal,walletId.text, TegmentSessionHandler.Instance._authToken,
OfferTransactionCallBack,true);
```

After defining the token asset to sell and the token asset or BSV and amount to receive, a partially signed transaction is created, and the hex value returned. This signature will bind the input provided by the user to the single output value in the offer hex

The hex value can be read by any third party services to complete the transaction in a decentralized way without reliance on the Relyisa infrastructure.

Inspect

Before accepting an atomic swap, the 2nd party might want to inspect the partially signed transaction hex to ensure they actually receive the asset they expect to get at the right conditions. For that we created the "inspect" function. Add any partially signed transaction hex to the function to decode the insights/conditions.

Inspect

SwapHex*

SwapId...

Response

Submit

response text

```
Tegment.Transaction.InspectV2.InspectTransactionV2 (swapHex.text,
TegmentSessionHandler.Instance._authToken,
InspectV2TransactionCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_swapHex	string	yes	
2	callback	function	yes	
3	enableLog	bool	no	

Swap

Finally, after ensuring the partially signed transaction is indeed having the assets one wants to buy for the given price, the atomic swap gets completed by the counterparty with the “swap” function

The “swap” function accepts the partially signed transaction hex as input (and thus also works cross platforms with external atomic swap offers).

Always ensure the asset you buy is actually the asset you want. While we try to warn users of "fake" assets, the user is signing whatever the counter party provides (especially when from external counterparties).

Swap

WalletID

WalletID...

SwapHex*

SwapHex...

Response

response text

Submit

```
Tegment.Transaction.Swap.SwapTransaction(SwapHex.text,walletId.text,Tegmen  
tSessionHandler.Instance._authToken, SwapTransactionCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_swapHex	string	yes	
2	callback	function	yes	
3	enableLog	bool	no	

After completing the atomic swap by counter signing the offer transaction (and adding their piece of information to the puzzle), the transaction is directly submitted and the transaction id returned.

Details

Verify that you got the correct STAS token

Fetch the latest state of your STAS token and the issuance details. The only parameter required is tokenID, which you will have if you issued a token via the “issue” function or can be found on the managed tokens section of Relysia Wallet.

Token V2

TokenId*

TokenId...

Response

Submit

response text

```
Tegment.SmartContracts.TokenV2.GetTokenV2(tokenId.text, Token_V2CallBack, true);
```


Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	tokenID	string	yes	
2	callback	function	yes	
3	enableLog	bool	no	

Leaderboard

To display the distribution of a particular token, we implemented the “leaderboard” function. A leaderboard is typically an information page on a platform page that shows how you rank among others. Distributing tokens in a competition and then displaying ownership is thus an interesting way to enrich the user experience.





Since tokens are on a public blockchain, we can aggregate all transactions and balances from the public blockchain to display the leaderboard. Thanks to p2p paymail integration, updating the leaderboard happens in real time.



Compete for a spot on the leaderboard!

Relysia API now offers leaderboard endpoint. This endpoint can be useful in competitions.

[Read Docs](#)

Ranking	User	Token Balance
1	 rudie	10.12
2	 norman2	9.29
3	 esra11	9.10
4	 mikej	8.76

Leaderboard

NextPageToken

TokenId*

Submit

Response

response text

```
Tegment.Wallet.Leadersboard.GetLeadersboard(nextPageToken.text,
TokenID.text, TegmentSessionHandler.Instance._authToken,
LeadersboardCallBack,true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_nextPageToken	string	no	

2	_tokenId	string	yes	
3	callback	function	yes	
4	enableLog	bool	no	

Post

Upload Data to the Blockchain

The bitcoin blockchain is a timestamp server. That means it can prove that certain documents were added to the blockchain at exactly that time and not modified later, which led to a variety of applications within the BSV space. From certifying documents against the blockchain to later proof no modification of it occurred to social media posts.

While miners are not required to keep a copy of your blockchain post for all eternity on their mining servers, a whole ecosystem spawned over the years that uses the blockchain as a decentralized storage system.

OP_Return data on the blockchain can be pruned by individual miners since OP_return transactions can't create another transaction (thus irrelevant for the validation process).

Even though the hosting of uploaded data is not guaranteed, the hash of transactions is always verifiable via SPV networks and presents a long term reliable path to verify that B://cat data is unaltered since upload.

The business of maintaining and hosting OP_Return data onchain has found some traction with services such as Project Babbage, Metashard and Planaria.

Post

CustomToken

Custom Token...

WalletId

WalletID...

Notes [Enter multiple values with comma separated]*

Notes...

Response

response text

Submit

```
Tegment.Utility.PostUtility.PostUtil(notesData, customToken.text,
walletId.text, TegmentSessionHandler.Instance._authToken,
PostResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_notesData	String array	yes	
2	_customToken	string	no	
3	_walletID	string	no	
4	_authToken	string	yes	
5	callback	function	yes	
6	enableLog	bool	no	

Upload

Upload Data to the Blockchain

Every transaction on the blockchain is a data bundle. Instead of integrating locking and unlocking scripts to the UTXO, one can also upload arbitrary data.

A common use case for uploads is to store NFT images or videos in B://cat protocol format directly on-chain instead of on a server, to ensure the long term availability of the media file.

Our media “upload” function is currently available to seamlessly upload files.

Upload

WalletId

FileURL

FileName

Response

response text

WalletId...

FileURL....

FileName...

Submit

```
Tegment.Utility.UploadUtility.UploadFile(fileURL.text, fileName.text,
walletId.text, TegmentSessionHandler.Instance._authToken,
UploadResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_fileUrl	string	yes	
2	_fileName	string	yes	
3	_walletID	string	no	
4	_authToken	string	yes	
5	callback	function	yes	
6	enableLog	bool	no	

We currently support uploads up to 20MB on the production server and keep extending the limits. In our prototypes we recently successfully uploaded >1GB files.

Same pruning considerations shall apply to the “upload” function.

Conversion

Get to know how many Satoshis are how much balance in any fiat currency

Currency Conversion

Since users often interact in their wallet with fiat currency denominations, while the wallet infrastructure takes Satoshis as the base unit of account, we created the “currencyconversion” function. Convert your Satoshi Balance in any FIAT currency of choice (e.g. USD).

Currency Conversion

Satoshis*

Satoshis...

Currency*

Currency...

Response

Submit

response text

```
Tegment.Utility.CurrencyConversion.ConvertSatoshiToCurrency(satoshis.text,
currency.text, CurrencyConversionResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_satoshis	string	yes	
2	_currency	string	yes	
3	callback	function	yes	
4	enableLog	bool	no	

The currency conversion rate gets updated every 10 minutes.

Paymail

Customize your paymail domain.

Paymail is an alias protocol that let's users pick an easier identifiable address (such as sato@relysia.com) and increases computational efficiency due with the peer-to-peer (P2P) approach.

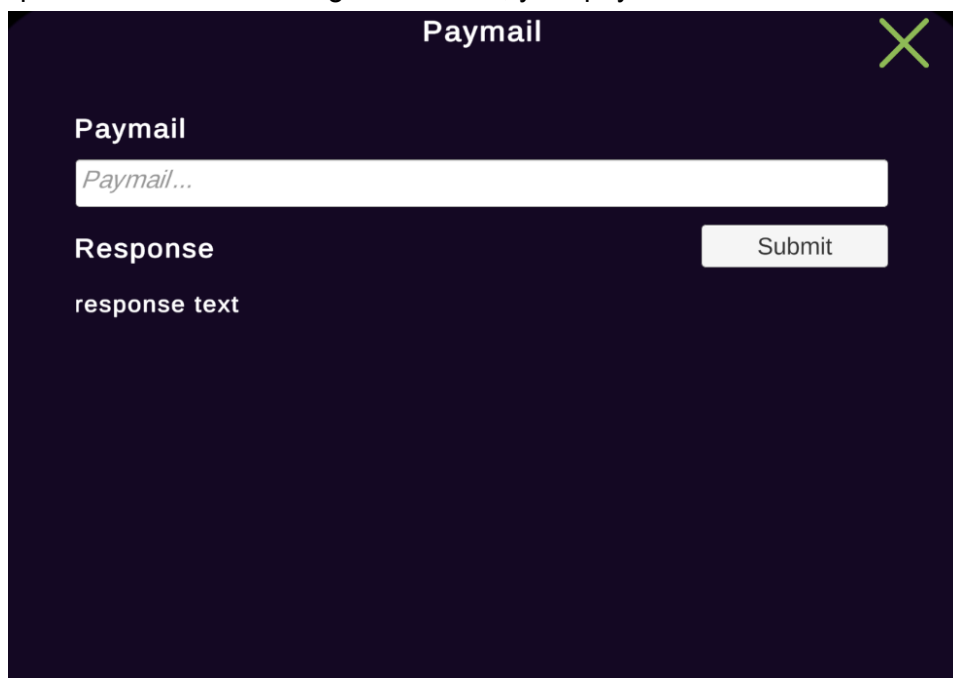
P2P means that instead of the receiver needing to search the blockchain for incoming transactions (potentially among billions), the receiver gets directly the transaction bundle which the receiver then submits to the miners and to instantly get a payment confirmation.

Well Known

To add your own paymail domain to your dedicated server (with a serviceID), you need to append to the `your-domain.com/.well-known/bsvalias` path the following json file.

```
{"bsvalias":"1.0","capabilities":{"6745385c3fc0":false,"pki":"https://api.relysia.com/v1/bsvalias/id/{alias}@{domain.tld}","paymentDestination":"https://api.relysia.com/v1/bsvalias/address/{alias}@{domain.tld}","a9f510c16bde":"https://api.relysia.com/v1/bsvalias/verifypubkey/{alias}@{domain.tld}/{pubkey}","5f1323cddf31":"https://api.relysia.com/v1/bsvalias/receive-transaction/{alias}@{domain.tld}","2a40af698840":"https://api.relysia.com/v1/bsvalias/p2p-payment-destination/{alias}@{domain.tld}"}}
```

you can find examples of the entry at all major wallets, including <https://relysia.com/.well-known/bsvalias>. Once you have added the entry to your domain & set up the serviceId with the given domain, your paymail will be active.

The image shows a dark-themed user interface for a 'Paymail' feature. At the top, the word 'Paymail' is centered in white, with a green 'X' icon in the top right corner. Below this, there is a section titled 'Paymail' in white. Under the title is a white text input field with the placeholder text 'Paymail...'. Below the input field is a 'Response' label in white. To the right of the 'Response' label is a white 'Submit' button. Below the 'Response' label is a 'response text' label in white.

```
Tegment.Paymail.Paymail_Get.GetPaymailDetails(paymail.text,TegmentSessionH  
andler.Instance._authToken, GetPaymailResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_paymailID	string	yes	
2	_authToken	string	yes	
3	callback	function	yes	
4	enableLog	bool	no	

Update Paymail

You can change your paymail to any of your personalized choices, but each paymail should be unique so before setting your personalized paymail we will check if it already does not exist in our system.

Edit Paymail

WalletId

WalletId...

New Paymail

New Paymail...

Response

response text

Submit

```
Tegment.Paymail.Paymail_Put.PutPaymailDetails(walletId.text,newPaymail.text, TegmentSessionHandler.Instance._authToken, PutPaymailResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_walletId	string	no	
2	_newPaymailId	string	yes	
3	_authToken	string	yes	
4	callback	function	yes	
5	enableLog	bool	no	

Activate/Deactivate Paymail

We have another function to activate or deactivate paymail

Paymail Activate/ Deactivate

WalletId

WalletId

Activate / Deactivate

Response

response text

Submit

```
Tegment.Paymail.Paymail_Activate.PaymailActivate_DeActivate(  
activate.isOn, walletId.text, TegmentSessionHandler.Instance._authToken,  
PaymailActivateResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_activate	bool	yes	
2	_walletID	string	no	
3	_authToken	string	yes	
4	callback	function	yes	
5	enableLog	bool	no	

Transpile-Compile

Transpile

This function is used to convert solidity script into sCrypt. We need to pass base64 version of the solidity script and in result we will get sCrypt script.

Transpile

Force

☐

SourceCode

Source Code...

Response

Submit

response text

```
Tegment.Utility.Transpile.Transpile_sCrypt(force.isOn, sourceCode.text,
TranspileResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_force	bool	yes	
2	_sourceCode	string	no	
3	callback	function	yes	
4	enableLog	bool	no	

Compile

This function is used to convert sCrypt script into bitcoin script. We need to pass the base64 version of sCrypt script and in result we will get bitcoin script.

Compile

SourceCode

Y29udHJhY3QgQ291bnRlciB7DQogIEBzdGF0ZQ0KICBwdWJsaWMgaV

Response

Submit

response text

```
Tegment.Utility.CompileUtility.Compile_sCrypt(sourceCode.text,  
CompileResponseCallBack, true);
```

Following table mentions about the parameters used in above code

Sr. No.	Param Name	Type	IsRequired	details
1	_sourceCode	string	no	
2	callback	function	yes	
3	enableLog	bool	no	