Index

Introduction

- Getting Started
- Video Tutorials
- Instrument Details
- Postman Collection and SDKs
- Developer’s Community
- Base URL
- Request and Response Structure

User

- Login Flow
- Get User Session
- Profile
- Limits
- Logout

Margin

- Pre-order Margin
- SPAN and Exposure

Order Management

- Place
- Modify
- Cancel
- Order Book
- Order History
- Trade Book
- Order and Trade Updates
- Order Update
- Trade Update

Portfolio

- Holdings
- Positions

Market Data Stream

- Steps to receive the Market Data events with Python example
- Market Feed
- Open Interest Stream
- Market Status
- Upper Circuit Change
- Lower Circuit Change
- Limit Price Protection(LPP) Change
- 52 Week High Change
- 52 Week Low Change

Market Data APIs

- Historical Candlestick Chart Data
- Market Quotes
- Market Depth
- Open Interest

Annexure
Instrument Details
Postman Collection
SDKs
Video Tutorials
Sample conversion of a binary event to decimal format
Developer’s Community
Trading API Error Codes
Python SDK errors and packets
RMS Order Rejections
Rate Limits
FAQs

Introduction

Getting Started

IIFL Markets’ APIs provide fast, efficient, and easy-to-use trading solutions for retail traders and fintech platforms. These APIs enable trading, investment, wealth generation, automation, and algorithmic strategy execution with low complexity.

As REST-like APIs, they integrate seamlessly with our trading ecosystem, allowing real-time order execution, modification, and cancellation. Users can manage portfolios, access live market data, and monitor order status. The APIs use resource-based URLs, accept JSON or form-encoded requests, and return JSON responses using standard HTTP codes. Toolkits are available in multiple programming languages for quick integration.

Video Tutorials

1. Trading API Document overview: A quick walkthrough of the key sections and API details in our documentation.
https://youtu.be/t7TMp4SldJ4

2. Daily client login flow: Step-by-step guide for clients to perform daily logins on the trading applications.
https://youtu.be/DMvSVncJOZA

Instrument Details

Get comprehensive exchange and segment-wise data of all active market instruments in **CSV** and **JSON** formats.

CSV Links:

Exchange & Segment	**Instrument Details CSV File Link**
NSEEQ	https://api.iiflcapital.com/v1/contractfiles/NSEEQ.csv

NSEFO	https://api.iiflcapital.com/v1/contractfiles/NSEFO.csv
NSECOMM	https://api.iiflcapital.com/v1/contractfiles/NSECOMM.csv
MCXCOMM	https://api.iiflcapital.com/v1/contractfiles/MCXCOMM.csv
INDICES	https://api.iiflcapital.com/v1/contractfiles/INDICES.csv
NSECURR	https://api.iiflcapital.com/v1/contractfiles/NSECURR.csv
BSEEQ	https://api.iiflcapital.com/v1/contractfiles/BSEEQ.csv
BSEFO	https://api.iiflcapital.com/v1/contractfiles/BSEFO.csv
BSECURR	https://api.iiflcapital.com/v1/contractfiles/BSECURR.csv

JSON Links:

Exchange & Segment	**Instrument Details JSON Link**
NSEEQ	https://api.iiflcapital.com/v1/contractfiles/NSEEQ.json
NSEFO	https://api.iiflcapital.com/v1/contractfiles/NSEFO.json
NSECOMM	https://api.iiflcapital.com/v1/contractfiles/NSECOMM.json
MCXCOMM	https://api.iiflcapital.com/v1/contractfiles/MCXCOMM.json
INDICES	https://api.iiflcapital.com/v1/contractfiles/INDICES.json
NSECURR	https://api.iiflcapital.com/v1/contractfiles/NSECURR.json
BSEEQ	https://api.iiflcapital.com/v1/contractfiles/BSEEQ.json
BSEFO	https://api.iiflcapital.com/v1/contractfiles/BSEFO.json
BSECURR	https://api.iiflcapital.com/v1/contractfiles/BSECURR.json

Note: These **JSON** links can also serve as **APIs**, allowing you to retrieve instrument details using the GET method. No authorization is required to make these API calls.

Postman Collection & SDKs

You can download the official Postman API Collection here.
Below is a list of pre-built official libraries for IIFL Markets' APIs developed in popularly used programming languages that can be used to interact with the APIs without having to make raw HTTP calls.

- Python library
- .Net Framework 4.6 library
- .Net Core 8 library
- Golang library
- Java library
- Node JS library

Developer's Community

🐞 Found a bug or facing an issue? Raise it on our GitHub Issues page. Alternatively, you may write to us at openapisupport@iiflcapital.com or reach us by phone or Whatsapp at +91-7718830851.

Base URL

The base URL is the common URL used as a prefix for all the API calls.
https://api.iiflcapital.com/v1

Request and Response Structure

Request Structure

- Users need to pass the generated access token as a Bearer token to authorise the request.

- All parameters for GET and DELETE requests are passed as query parameters.

Example:

```
DELETE
https://api.iiflcapital.com/v1/orders/{brokerOrderId}
```

- All POST and PUT requests are sent in raw JSON format in the request body.

Example:

```
POST
https://api.iiflcapital.com/v1/orders
```

Body:

```
[
  {
    "instrumentId": "1594",
    "exchange": "NSEEQ",
    "transactionType": "BUY",
    "quantity": "1",
    "orderComplexity": "REGULAR",
    "product": "INTRADAY",
    "orderType": "MARKET",
    "validity": "DAY"
  }
]
```

Response Structure

- **status:** Indicates the success, failure or error of the overall API call.
- **message:** Provides additional explanation or error description for the status.
- **result:** A list of individual results, each containing specific details about a single operation (e.g., order details). The sequence of the responses in the result array matches the sequence of the requests. For example, if leg A was sent first and leg B was sent second, the result array will list the response for leg A first, followed by the response for leg B inside the result array.
- **result[].status:** Provides the status or error of the individual operation within the response
- **result[].message:** Provides additional details or error description about the status of the individual operation

Example:

```
{
"status": "Ok", // Overall status of the Place Order API Call
"message": "Success", // Provides additional explanation for the status
"result": [ // List of individual results, each containing specific details about placing that particular order.
```

```
    {
      "status": "Success", // Shows the status of 1st leg, shows that it has passed all the validations
      "message": "Success", // provides additional information on the 1st leg status
      "brokerOrderId": "240919000000041", // response data, brokerOrderId in this case
      "requestTime": "19-Sep-2024 18:48:46", // response data, requestTime in this case
     },
{
      "status": "EC901", // Code for the occurred error while validating 2nd leg's parameters
      "message": "Invalid parameter: 'exchange' Accepts only
{'NSEEQ','NSEFO','BSEEQ','BSEFO','NSECURR','BSECURR','MCXCOMM','NCDEXCOMM','NSECOMM','
BSECOMM'}" // Error Description for the above error code
          }
    ]
}
```

User

There are 4 user APIs which enable login, logout, provide access to the client's account details, and display their trading margins and available limits.

Method	Endpoint	Processing Mode	Action
POST	/getusersession	Single	Generates access token for authenticating API calls
GET	/profile	Single	Retrieves client's account details and personal information
GET	/limits	Single	Provides the user's available trading limits and margins
POST	/profile/logout	Single	Terminates the user's active session

Login Flow

To log in, a client requires the following:

1. **Trading Account Credentials**: Email/Phone/ClientId/PAN and password. The client can reset their password directly on the web login page if they have forgotten it.
2. **OTP or TOTP**: For two-factor authentication.
3. **App Key and App Secret**: Specific to the application the client is logging into (contact your Relationship Manager or Point of Contact to obtain these).

Daily Login Steps:

1. **Redirect to Login Endpoint**:
   The client must be redirected to the IIFL Capital web login URL:
   `https://markets.iiflcapital.com/?v=1&appkey=xxx&redirecturl=abc`
   Replace `xxx` with the **appKey** of the application being used. This appKey can belong to the client's own application or a third-party fintech platform(contact your Relationship Manager or Point of Contact to obtain appKey and appSecret)

If you have a use case to redirect the user dynamically to different URLs after login, you can pass an **optional** `redirecturl` parameter in the OAuth URL. If this parameter is provided, users will be redirected to that specific URL post-login; otherwise, they will be redirected to the default URL registered with your trading application.

2. **Log In Using Credentials**:
   The client should enter their IIFL trading account credentials (username, password, and OTP/TOTP) on the login portal.
3. **Redirection Post Login**:
   After a successful login, the client will be redirected to the URL specified during application creation, with the `authCode` and `clientId` appended to it.

   If no redirect URL was specified during the app creation, they will be redirected to an IIFL webpage displaying an `authCode`.

4. **Generate Access Token**:
   Use any open source website(e.g., https://emn178.github.io/online-tools/sha256.html) or code to encrypt the combination of **clientId+authCode+appSecret** using **SHA-256** and pass it as a `checkSum` in the **getusersession API** to generate the access token.
5. **Authorise Subsequent API Calls**:
   Use the generated access token(userSession) as a **Bearer token** in the header of every subsequent API request for authorization.

A new login is required on each trading day to regenerate the token.

Get User Session

Method	**Endpoint**	**Processing Mode**	**Action**
POST	/getusersession	Single	Generates access token for authenticating API calls

Request Structure

```
{
    "checkSum":"bec46c08a04f7c8f1ea355b85d0f32e9250d83ffe42a147400fc4d0bdb9aee2b"
}
```

Request Parameters

Parameter	**Data Type**	**Description**	**Approved/Example Values**
checkSum *Required*	varchar	SHA-256 encrypted value of concatenation of (clientId,authCode,apiSecret)	SHA256(CONCATENATION(TEST101,IUBTQFRGKC0NV9EHKCPT,LY47CQpl9N1T4NLBEYiwd0F5tdI9B0f7xNM18rfjSTELztsSfvfK3lNZQU7b3MQbCwclFd4EUw0pO9ufWv3aOFdw6jgW9W4Jck8E)) → 92568ec06bbdc2c549c394c277c81319e35d

			5d57b88db1cba6469f492818f1e3

Response Structure

```
{
    "status": "Ok",
    "userSession": "eyJhbGciOi*******bx844uFwWA"
}
```

Response Parameters

Param	Data Type	Description	Example Values
status	varchar	Indicates whether the session creation was successful ("Ok") or not ("Not_Ok")	Ok, Not_Ok
userSession	varchar	A JWT (JSON Web Token) representing the client's authenticated session, to be used for the subsequent API authorization	eyJhbGciOi*******bx844uFwWA

Profile

Method	Endpoint	Processing Mode	Action
GET	/profile	Single	Retrieves client's account details and personal information

Headers

Authorization: Bearer <userSession>

Request Structure

No Body

Response Structure

```
{
    "status": "Ok",
    "message": "Success",
    "result": {
        "clientId": "3******4",
        "clientName": "Ishan Arora",
        "isTotpEnabled": "Y",
        "isPoaProvided": "Y",
        "accountStatus": "Na",
        "exchanges": [
            "BSECURR",
            "BSEEQ",
            "BSEFO",
            "NSECURR",
            "NSEEQ",
            "NSEFO"
        ],
        "products": [
            "NORMAL",
            "INTRADAY",
            "DELIVERY",
            "BNPL"
        ],
        "orderComplexity": [
            "REGULAR",
            "AMO",
            "BO",
            "CO"
        ],
        "email": "is**********@gmail.com",
        "phoneNumber": "8********4"
    }
}
```

Response Parameters

Param	Data Type	Description	Example Values
clientId	varchar	Unique identifier assigned to the client	TEST102
clientName	varchar	First name of the client as registered with the broker	Vishal
isTotpEnabled	varchar	Indicates if TOTP-based 2FA is enabled for the client	Y/N
isPoaProvided	varchar	Indicates if the Power of Attorney has been provided	Y/N
accountStatus	varchar	Current status of the client's trading account	Active/Inactive
exchanges	varchar	List of stock exchanges the client is authorized to trade on	[“BSEFO”]
products	varchar	Trading products enabled for the client	[“DELIVERY”]
orderComplexity	varchar	Types of orders the client can place	["REGULAR"]

Limits

**For Indian resident clients, trading limits are pooled across segments. For NRI clients, trading limits are maintained separately for Equity and F&O segments. All three APIs below return the same response values for Indian resident clients, but differ for NRI clients.*

Method	Endpoint	Processing Mode	Action
GET	/limits	Single	Provides overall trading limits for Indian resident clients and F&O segment trading limits for NRI clients.
GET	/limits/fno	Single	Provides overall trading limits for Indian resident clients and F&O segment trading limits for NRI clients.
GET	/limits/equity	Single	Provides overall trading limits for Indian resident clients and Equity segment trading limits for NRI clients.

Headers

Authorization: Bearer <userSession>

Request Structure

No Body

Response Structure

```
{
    "status": "Ok",
    "message": "Success",
    "result": {
        "tradingLimit": 19990000.00,
        "openingCashLimit": 0.0,
        "intradayPayin": 20000000.00,
        "collateralMargin": 0.0,
        "creditForSell": 0.0,
        "adhocMargin": 0.0,
        "utilizedMargin": 0.00,
        "blockedForPayout": 10000.0,
        "utilizedSpanMargin": 0.0,
        "utilizedExposureMargin": 0.0
    }
}
```

Response Parameters

Param	Data Type	Description	Example Values
tradingLimit	double	Maximum funds available for trading	19990000.00
openingCashLimit	double	Cash balance at the start of the trading day	0.0
intradayPayin	double	Funds added during the day for intraday trades	20000000.00
collateralMargin	double	Margin available from pledged securities	0.0
creditForSell	double	Credit received for securities sold but not yet settled	0.0

adhocMargin	double	Additional margin provided by the broker	0.0
utilizedMargin	double	Total margin already used for open positions and open orders	0.0
blockedForPayout	double	Amount blocked for payout requests	10000.0
utilizedSpanMargin	double	Margin used to cover SPAN requirements for derivatives	0.0
utilizedExposureMargin	double	Margin used for exposure requirements in derivatives	0.0

Logout

Method	**Endpoint**	**Processing Mode**	**Action**
POST	/profile/logout	Single	Terminates the user's active session

Headers

Authorization: Bearer <userSession>

Request Structure

No Body

Response Structure

```
{
    "status": "Ok",
    "message": "Success",
    "result": {
        "status": "Success",
        "message": "Success"
    }
}
```

Response Parameters

Param	**Data Type**	**Description**	**Example Values**
status	varchar	Overall status of the Place Order API Call	Ok
message	varchar	Provides additional explanation for the status	Success
result[].status	varchar	Provides the status or error of the logout operation	Success
result[].message	varchar	Provides additional details or error description about the status of the logout operation	Success

Margin

There are 2 margin APIs that calculate the required margin for an individual order or a basket of orders.

Method	Endpoint	Processing Mode	Action
POST	/preordermargin	Single	Calculate the required margin for an order considering the existing positions
POST	/spanexposure	Single/Bulk	Calculate the SPAN, exposure and total margin required for 1 or more instruments without considering the existing positions and open orders

Pre-order Margin

Method	Endpoint	Processing Mode	Action
POST	/preordermargin	Single	Calculate the required margin for an order considering the existing positions

Headers

Authorization: Bearer <userSession>

Request Structure

*Note: Parameters marked in **red** are **required** in the request payload whereas parameters marked in **grey** are either **conditionally required or optional.***

```
{
   "instrumentId": "54786",
   "exchange": "NSEFO",
   "transactionType": "SELL",
   "quantity": "25",
   "orderComplexity": "REGULAR",
   "product": "NORMAL",
   "orderType": "MARKET",
   "validity": "DAY",
    "price": "1501.55",
    "slTriggerPrice": "1505.2",
    "slLegPrice": "1500",
    “targetLegPrice”: “1600”
}
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values
instrumentId *Required*	varchar	Unique identifier for the financial instrument	1594

exchange *Required*	varchar	Exchange & segment of the instrument to be traded	NSEEQ,NSEFO,BSEEQ,BSEFO,NSECURR,BSECURR,MCXCOMM,NSECOMM,BSECOMM,NCDEXCOMM
transactionType *Required*	varchar	Defines whether the order is a buy or sell	BUY,SELL
quantity *Required*	int	Number of units to be traded	100
orderComplexity *Required*	varchar	Specifies the complexity of the order	REGULAR,AMO,BO,CO
product *Required*	varchar	Product type for the order	NORMAL,INTRADAY,DELIVERY,BNPL
orderType *Required*	varchar	Order type	LIMIT,MARKET,SL,SLM
price *Conditionally Required*	double	The specified price for limit orders *Req. if orderType=LIMIT,SL*	1501.55
slTriggerPrice *Conditionally Required*	double	Trigger price for stop-loss orders *Req. if orderType=SL,SLM*	1505.2
slLegPrice *Conditionally Required*	double	The exit price forBracket/Cover orders *Req. if orderComplexity=BO,CO*	1500
targetLegPrice *Conditionally Required*	double	The book profit price for Bracket orders *Req. if orderComplexity=BO,CO*	1600

Response Structure

```
{
        "totalCashAvailable": "10006024.00",
        "preOrderMargin": "76624.00",
        "postOrderMargin": "88710.73",
        "currentOrderMargin": "12086.73",
        "rmsvalidationMessage": "OK",
        "fundShort": "0.00"
    }
```

Response Parameters

Param	Data Type	Description	Example Values
totalCashAvailable	double	Total Cash Available	10006024.00
preOrderMargin	double	Portfolio margin used before this order is executed	76624.00

postOrderMargin	double	Portfolio margin used after this order gets executed	88710.73
currentOrderMargin	double	Margin required to place this order	12086.73
rmsvalidationMessage	varchar	Rms check indicating whether the RMS will allow or reject the order, if placed	OK, NOT_OK
fundShort	double	Shortage of funds, if any	0.00

SPAN and Exposure Margin

Method	Endpoint	Processing Mode	Action
POST	/spanexposure	Single/Bulk	Calculate the SPAN, exposure and total margin required for 1 or more instruments without considering the existing positions and open orders

Headers

Authorization: Bearer <userSession>

Request Structure

```
[
  {
    "instrumentId": "35089",
    "exchange": "NSEFO",
    "transactionType": "BUY",
    "quantity": 25
  },
  {
    "instrumentId": "35005",
    "exchange": "NSEFO",
    "transactionType": "SELL",
    "quantity": 25
  }
]
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values
instrumentId *Required*	varchar	Unique identifier for the financial instrument	1594
exchange *Required*	varchar	Exchange & segment of the instrument to be traded	NSEEQ,NSEFO,BSEEQ,BSEFO,NSECURR,BSECURR,MCXCOMM,NSECOMM,BSECOMM,NCDEXCOMM
transactionType *Required*	varchar	Defines whether the order is a buy or sell	BUY,SELL

quantity *Required*	int	Number of units to be traded	100

Response Structure

```
{
    "span": "11090.00",
    "exposureMargin": "4122.17",
    "totalMargin": "15212.17",
     "buyPremium":  "0.00"
}
```

Response Parameters

Param	Data Type	Description	Example Values
span	double	The minimum required margin based on worst-case risk scenarios for derivative positions	11090.00
exposureMargin	double	An additional cushion margin for unexpected market fluctuations	4122.17
totalMargin	double	The combined margin requirement, totaling SPAN and exposure margins for full risk coverage	15212.17
buyPremium	double	Displays the total premium required to purchase the option contracts in the request	0.00

Order Management

There are 6 Order Management APIs that let you place, modify, and cancel orders. They also provide access to the order book, trade book, order history, and pre-order margin.

Method	Endpoint	Processing Mode	Action
POST	/orders	Single/Bulk	Place new orders
PUT	/orders/{brokerOrderId}	Single	Modify pending orders
DELETE	/orders/{brokerOrderId}	Single	Cancel a Pending order
GET	/orders	Single	Fetch details of all orders placed today
GET	/orders/{brokerOrderId}	Single	Fetch the history of a specific order
GET	/trades	Single	Fetch details of all executed trades for today

Place Order

Method	Endpoint	Processing Mode	Action
POST	/orders	Single/Bulk	Place new orders

Headers

Authorization: Bearer <userSession>

Request Structure

Note:

1. *Parameters marked in **red** are **required** in the request payload whereas parameters marked in **grey** are either **conditionally required or optional.***
2. *All commodity segments (MCXCOMM, NSECOMM, BSECOMM, NCDEXCOMM) **accept quantity in lots** in the Order and Margin APIs. All other segments **accept quantity as absolute quantity.***
3. ***NCDEXCOMM accepts the tradingSymbol as the instrumentId**, while all other segments use the numeric instrument token as the instrumentId. For example, use instrumentId = DHANIYA10JUL25PE8600FJUL25 for NCDEXCOMM, and instrumentId = 2885 for NSEEQ.*

```
[
{
    "instrumentId": "1594",
    "exchange": "NSEEQ",
    "transactionType": "BUY",
    "quantity": "1",
    "orderComplexity": "REGULAR",
    "product": "INTRADAY",
    "orderType": "MARKET",
    "validity": "DAY",
    "price": "1501.55",
    "slTriggerPrice": "1505.2",
    "slLegPrice": "1500",
    "targetLegPrice ": "1600",
    "disclosedQuantity": "500",
    "marketProtectionPercent ": "1400",
    "apiOrderSource": "XYZ",
    "algoId": "1234",
    "orderTag": "Stangle Leg 1"
  }
]
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values

instrumentId *Required*	varchar	Unique identifier for the financial instrument	1594
exchange *Required*	varchar	Exchange & segment of the instrument to be traded	NSEEQ,NSEFO,BSEEQ,BSEFO,NSECURR,BSECURR,MCXCOMM,NSECOMM,BSECOMM,NCDEXCOMM
transactionType *Required*	varchar	Defines whether the order is a buy or sell	BUY,SELL
quantity *Required*	int	Number of units to be traded	100
orderComplexity *Required*	varchar	Specifies the complexity of the order	REGULAR,AMO,BO,CO
product *Required*	varchar	Product type for the order	NORMAL,INTRADAY,DELIVERY,BNPL
orderType *Required*	varchar	Order type	LIMIT,MARKET,SL,SLM
price *Conditionally Required*	double	The specified price for limit orders *Req. if orderType=LIMIT,SL*	1501.55
slTriggerPrice *Conditionally Required*	double	Trigger price for stop-loss orders *Req. if orderType=SL,SLM*	1505.2
slLegPrice *Conditionally Required*	double	The exit price for stop-loss orders *Req. if orderComplexity=BO,CO*	1500
targetLegPrice *Conditionally Required*	double	Price at which to exit the position for profit *Req. if orderComplexity=BO*	1600
validity *Required*	varchar	The time duration the order should remain active	DAY,IOC
disclosedQuantity *Optional*	int	The partial quantity disclosed to the market	500
marketProtectionPercent *Optional*	double	Market protection ensures that this sell order is not executed below (or buy order above) the specified price, safeguarding against unfavourable price movements.	1400
apiOrderSource *Optional*	varchar	The source from which the API order is placed. Fintech partners should specify their name in this parameter.	PlatformName1
algoId *Optional*	varchar	The identifier for the algorithm placing the order	1234
orderTag *Optional*	varchar	A custom label or tag assigned to the order	Stangle Leg 1

Response Structure

```
{
    "status": "Ok",
    "message": "Success",
    "result":
[
{
            "status": "Success",
            "message": "Success",
            "brokerOrderId": "240919000000041",
            "requestTime": "19-Sep-2024 18:48:46"
}
]
}
```

Response Parameters

Parameter	Data Type	Description	Example Values
status	varchar	Status of the request or error code	Success, EC004
message	varchar	Any additional message or error description	Success, User blocked
brokerOrderId	varchar	IIFL's unique order number	240919000000041
requestTime	varchar	Request Timestamp	19-Sep-2024 18:48:46

Modify Order

Method	Endpoint	Processing Mode	Action
PUT	/orders/{brokerOrderId}	Single	Modify pending orders

Headers

Authorization: Bearer <userSession>

Request Structure

Note: Parameters marked in ***red*** *are* ***required*** *in the request payload whereas parameters marked in* ***grey*** *are either* ***conditionally required or optional.***

```
[
{
      "quantity": "1",
      "orderType": "MARKET",
      "validity": "DAY",
      "price": "1501.55",
      "slTriggerPrice": "1505.2",
      "disclosedQuantity": "500",
      "marketProtectionPercent ": "1400"
  }
]
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values
quantity *Optional*	int	Number of units to be traded	100
orderType *Optional*	varchar	Order type	LIMIT,MARKET,SL,SLM
price *Conditionally Required*	double	The specified price for limit orders *Req. if orderType=LIMIT,SL*	1501.55
slTriggerPrice *Conditionally Required*	double	Trigger price for stop-loss orders *Req. if orderType=SL,SLM*	1505.2
validity *Optional*	varchar	The time duration the order should remain active	DAY,IOC
disclosedQuantity *Optional*	int	The partial quantity disclosed to the market	500
marketProtectionPercent *Optional*	double	Market protection ensures that this sell order is not executed below (or buy order above) the specified price, safeguarding against unfavourable price movements.	1400

****Note: You only need to pass the parameters which you want to modify or any additional related parameter which is required for the modification.***

Use Case I: If you want to change the quantity of an order to 15, you only need to pass the below request payload.

```
{
        "quantity": "15"
    }
```

Use Case II: If you want to change the orderType from MARKET to LIMIT and set the limit price as 100, you need to pass orderType and price.

```
{
        "orderType": "LIMIT",
    "price": "100"
}
```

Use Case III: If you want to change the orderType from LIMIT to SL and set the stop loss trigger price at 99 keeping price the same, you need to pass orderType and slTriggerPrice.

```
{
    "orderType": "SL",
    "slTriggerPrice": "99"
}
```

Response Structure

```
{
  "status": "Ok",
  "message": "Success",
  "result":
    {
      "status": "Success",
      "message": "Success",
      "brokerOrderId": "240919000000041",
      "requestTime": "19-Sep-2024 18:48:46"
    }
}
```

Response Parameters

Parameter	Data Type	Description	Example Values
status	varchar	Status of the request or error code	Success, EC004
message	varchar	Any additional message or error description	Success, User blocked
brokerOrderId	varchar	IIFL's unique order number	240919000000041
requestTime	varchar	Request Timestamp	19-Sep-2024 18:48:46

Cancel Order

Method	Endpoint	Processing Mode	Action
DELETE	/orders/{brokerOrderId}	Single	Cancel a Pending order

Headers
Authorization: Bearer <userSession>

Request Structure

No Body

Response Structure

```
{
  "status": "Ok",
  "message": "Success",
  "result":
    {
      "status": "Success",
      "message": "Success",
      "brokerOrderId": "240919000000041",
      "requestTime": "19-Sep-2024 18:48:46"
    }
}
```

Response Parameters

Parameter	Data Type	Description	Example Values
status	varchar	Status of the request or error code	Success, EC004
message	varchar	Any additional message or error description	Success, User blocked
brokerOrderId	varchar	IIFL's unique order number	240919000000041
requestTime	varchar	Request Timestamp	19-Sep-2024 18:48:46

Order Book

Method	Endpoint	Processing Mode	Action
GET	/orders	Single	Fetch details of all orders placed today

Headers

Authorization: Bearer <userSession>

Request Structure

No Body

Response Structure

```
[
{
      "clientId": "TEST102",
      "placedBy": "TEST102",
      "brokerOrderId": "240920000000124",
      "exchangeOrderId": "1100000000089930",
      "orderStatus": "cancelled",
      "formattedInstrumentName": "INFOSYS LIMITED",
      "tradingSymbol": "INFY-EQ",
      "instrumentId": "1594",
      "exchange": "NSEEQ",
      "transactionType": "BUY",
      "quantity": "1",
      "product": "INTRADAY",
      "orderComplexity": "REGULAR",
      "orderType": "LIMIT",
      "price": "180000.0",
      "averageTradedPrice": "0.0",
      "slTriggerPrice": "0.0",
      "validity": "DAY",
      "disclosedQty": "0",
      "marketProtectionPercent ": "0.0",
      "exchangeTimestamp": "20-Sep-2024 16:05:33",
      "exchangeUpdateTime": "20-Sep-2024 16:05:32",
      "rejectionReason": "",
      "mainLegOrderId ": "NA",
      "pendingQuantity": "1",
      "filledQuantity": "0",
      "appKey": "NA",
      "apiOrderSource": "API",
      "algoId": "",
      "source": "API",
      "orderTag": "NA",
      "brokerUpdateTime": "17-Feb-2025 06:48:39"
  }
]
```

Response Parameters

Parameter	Data Type	Description	Example Values
`clientId`	`varchar`	Unique identifier of the client	31625881
`placedBy`	`varchar`	User(client or dealer) who placed the order	31625881
`brokerOrderId`	`varchar`	Unique order ID assigned by IIFL Capital	240807000000068
`exchangeOrderId`	`varchar`	Unique order ID assigned by the exchange	9000000000000000

orderStatus	varchar	Current status of the order (e.g., Open, Complete, Rejected, Canceled)	rejected
formattedInstrumentName	varchar	Display name of the financial instrument	IDEA VODAFONE LIMITED
tradingSymbol	varchar	Trading symbol of the instrument	IDEA-EQ
InstrumentId	varchar	Unique identifier for the financial instrument	14366
exchange	varchar	The exchange where the order is placed	NSEEQ
transactionType	varchar	Specifies whether the order is a buy or sell	BUY
quantity	int	Number of shares or contracts in the order	1
product	varchar	Type of product used for the order (e.g., intraday or delivery)	NORMAL
orderComplexity	varchar	Complexity of order, such as Regular, AMO, BO, or CO	REGULAR
orderType	varchar	Specifies the execution type of the order (e.g., Market or Limit)	MARKET
price	double	Price at which the order is placed for limit orders	1500
averageTradedPrice	double	Average price at which the order was executed	1510
slTriggerPrice	double	Stop-loss trigger price for the order	1600
validity	varchar	Duration for which the order is valid (e.g., DAY, IOC)	DAY
disclosedQuantity	int	Partial quantity disclosed to the market	10
marketProtectionPercent	double	Market protection ensures that this sell order is not executed below (or buy order above) the specified price, safeguarding against unfavourable price movements.	1300
exchangeTimestamp	varchar	Timestamp when the order was placed	07-Aug-2024 16:14:17
exchangeUpdateTime	varchar	Timestamp when the exchange updated the order status	07-Aug-2024 16:14:17
rejectionReason	varchar	Reason for order rejection, if applicable	RMS:Margin Exceeds,****duct
mainLegOrderId	varchar	ID of the parent order, if this order is part of a Bracket Order or Cover Order	240807000000068
pendingQuantity	int	Quantity of the order that remains unfilled	25
filledQuantity	int	Quantity of the order that has been filled	50
appKey	varchar	Key identifying the application that placed the order	UXjPaQVPYyeVDEZ
apiOrderSource	varchar	The source from which the API order is placed. Fintech partners may specify their name in this parameter.	Trade Tron
algoId	varchar	Unique identifier for the algorithm that placed the order	algo123
source	varchar	The platform or system from which the order was placed(e.g., Web, App, API)	API

orderTag	varchar	Custom tag or label assigned to the order	123abc
brokerUpdateTime	varchar	Timestamp of the broker's last update on the order	17-Feb-2025 06:48:39

Order History

Method	Endpoint	Processing Mode	Action
GET	/orders/{brokerOrderId}	Single	Fetch the history of a specific order

Headers

Authorization: Bearer <userSession>

Request Structure

No Body

Response Structure

```
[
{
        "clientId": "TEST102",
        "placedBy": "TEST102",
        "brokerOrderId": "240920000000124",
        "exchangeOrderId": "1100000000089930",
        "orderStatus": "cancelled",
        "formattedInstrumentName": "INFOSYS LIMITED",
        "tradingSymbol": "INFY-EQ",
        "instrumentId": "1594",
        "exchange": "NSEEQ",
        "transactionType": "BUY",
        "quantity": "1",
        "product": "INTRADAY",
        "orderComplexity": "REGULAR",
        "orderType": "LIMIT",
        "price": "180000.0",
        "averageTradedPrice": "0.0",
        "slTriggerPrice": "0.0",
        "validity": "DAY",
        "disclosedQty": "0",
        "marketProtectionPercent ": "0.0",
        "exchangeTimestamp": "20-Sep-2024 16:05:33",
        "exchangeUpdateTime": "20-Sep-2024 16:05:32",
        "rejectionReason": "",
        "mainLegOrderId ": "NA",
        "cancelledQuantity": "1",
        "pendingQuantity": "1",
        "filledQuantity": "0",
        "appKey": "NA",
        "apiOrderSource": "API",
        "algoId": "",
        "source": "API",
        "orderTag": "NA",
        "brokerUpdateTime": "17-Feb-2025 06:48:39"
    }
]
```

Response Parameters

Parameter	**Data Type**	**Description**	**Example Values**
clientId	varchar	Unique identifier of the client	31625881
placedBy	varchar	User(client or dealer) who placed the order	31625881
brokerOrderId	varchar	Unique order ID assigned by IIFL Capital	240807000000068
exchangeOrderId	varchar	Unique order ID assigned by the exchange	9000000000000000
orderStatus	varchar	Current status of the order (e.g., Open, Complete, Rejected, Canceled)	rejected
formattedInstrumentName	varchar	Display name of the financial instrument	IDEA VODAFONE LIMITED
tradingSymbol	varchar	Trading symbol of the instrument	IDEA-EQ
InstrumentId	varchar	Unique identifier for the financial instrument	14366
exchange	varchar	The exchange where the order is placed	NSEEQ
transactionType	varchar	Specifies whether the order is a buy or sell	BUY
quantity	int	Number of shares or contracts in the order	1
product	varchar	Type of product used for the order (e.g., intraday or delivery)	NORMAL
orderComplexity	varchar	Complexity of order, such as Regular, AMO, BO, or CO	REGULAR
orderType	varchar	Specifies the execution type of the order (e.g., Market or Limit)	MARKET
price	double	Price at which the order is placed for limit orders	1500
averageTradedPrice	double	Average price at which the order was executed	1510
slTriggerPrice	double	Stop-loss trigger price for the order	1600

validity	varchar	Duration for which the order is valid (e.g., DAY, IOC)	DAY
disclosedQuantity	int	Partial quantity disclosed to the market	10
marketProtectionPercent	double	Market protection ensures that this sell order is not executed below (or buy order above) the specified price, safeguarding against unfavourable price movements.	1300
exchangeTimestamp	varchar	Timestamp when the order was placed	07-Aug-2024 16:14:17
exchangeUpdateTime	varchar	Timestamp when the exchange updated the order status	07-Aug-2024 16:14:17
rejectionReason	varchar	Reason for order rejection, if applicable	RMS:Margin Exceeds,****duct
mainLegOrderId	varchar	ID of the bracket order, if this order is part of a Bracket Order	240807000000068
cancelledQuantity	int	Quantity of the order that was cancelled	30
pendingQuantity	int	Quantity of the order that remains unfilled	25
filledQuantity	int	Quantity of the order that has been filled	50
appKey	varchar	Key identifying the application that placed the order	UXjPaQVPYyeVDEZ
apiOrderSource	varchar	The source from which the API order is placed. Fintech partners may specify their name in this parameter.	Trade Tron
algoId	varchar	Unique identifier for the algorithm that placed the order	algo123
source	varchar	The platform or system from which the order was placed(e.g., Web, App, API)	API
orderTag	varchar	Custom tag or label assigned to the order	123abc
brokerUpdateTime	varchar	Timestamp of the broker's last update on the order	17-Feb-2025 06:48:39

Trade Book

Method	Endpoint	Processing Mode	Action
GET	/trades	Single	Fetch details of all executed trades for today

Headers
Authorization: Bearer <userSession>

Request Structure
No Body

Response Structure

```
[
{
“clientId”: "31625881",
“placedBy”: "31625881",
“brokerOrderId”: "240807000000068",
“exchangeOrderId”: "900000000000000",
“exchangeTradeId”: "893487609000000",
“formattedInstrumentName”: "IDEA VODAFONE,
“tradingSymbol”: "IDEA-EQ",
“instrumentId”: "14366",
“exchange”: "NSEEQ",
“transactionType”: "BUY",
“product”: "NORMAL",
“orderComplexity”: "REGULAR",
“orderType”: "MARKET",
“validity: "DAY",
“tradedPrice”: "1560",
“filledQuantity”: "50",
“fillTimestamp: "07-Aug-2024 16:14:17",
“algoId”: "algo123",
“orderTag”: "123abc"
}
]
```

Response Parameters

Param	**Data Type**	**Description**	**Example Values**
clientId	varchar	Unique identifier of the client	31625881
placedBy	varchar	User(client or dealer) who placed the order	31625881
brokerOrderId	varchar	Unique order ID assigned by IIFL Capital	240807000000068
exchangeOrderId	varchar	Unique order ID assigned by the exchange	900000000000000
exchangeTradeId	varchar	Unique trade ID assigned by the exchange	893487609000000
formattedInstrumentName	varchar	Display name of the financial instrument	IDEA VODAFONE LIMITED
tradingSymbol	varchar	Trading symbol of the instrument	IDEA-EQ
instrumentId	varchar	Unique identifier for the financial instrument	14366
exchange	varchar	The exchange where the order is placed	NSEEQ
transactionType	varchar	Specifies whether the order is a buy or sell	BUY
product	varchar	Type of product used for the order (e.g., intraday or delivery)	NORMAL

orderComplexity	varchar	Complexity of order, such as Regular, AMO, BO, or CO	REGULAR
orderType	varchar	Specifies the execution type of the order (e.g., Market or Limit)	MARKET
validity	varchar	Duration for which the order is valid (e.g., DAY, IOC)	DAY
tradedPrice	double	The price at which the order was filled in the market	1560
filledQuantity	int	Quantity of the order that has been filled	50
fillTimestamp	varchar	Timestamp when the order was filled	07-Aug-2024 16:14:17
algoId	varchar	Unique identifier for the algorithm that placed the order	algo123
orderTag	varchar	Custom tag or label assigned to the order	123abc

Order and Trade Updates

Our Bridge Package offers two distinct events: one for delivering order updates and another for delivering trade updates. These events provide real-time updates on orders and executed trades.

To receive these events, you must register their designated callback functions.

Event Type	Setter Property to register callbacks	Request Param	Response Format	Description
Order Updates	on_order_updates_received	Client Id	JSON	Provides real-time updates on the status of placed orders, including modifications, cancellations, and execution progress
Trades Updates	on_trade_updates_received	Client Id	JSON	Delivers instant notifications of executed trades with details such as price, quantity, and trade time

Follow the steps below to receive order and trade update events. The example uses the Python SDK for illustration and does not include the complete code. Please refer to the full SDK implementation available in multiple languages on our GitHub portal for integration.

1. Install BridgePy package

```
pip install bridgePy
```

2. Import connector module from the package

```
from bridgePy import connector as connector
```

3. Create an object of the connect class

```
connection_object = connector.Connect()
```

4. Before establishing a connection, complete the below steps to receive and handle acknowledgments and error messages effectively. These acknowledgements and error messages are for connection, subscription, unsubscription and disconnection requests.

   a. Create an `acknowledgment_handler` function

   ```
   def acknowledgment_handler(response: str):
       print(f"Acknowledgment: {response}")
   ```

   b. A setter property is responsible for registering a callback function. When you pass a function to it, the setter will save the function so that it can be used later when the event arrives.

      Use the setter property `on_acknowledge_response` to register the `acknowledgment_handler` function created in the previous step as the callback function for the acknowledgement event.

   ```
   connection_object.on_acknowledge_response =acknowledgment_handler
   ```

   c. When the actual acknowledgment is received, the bridge package will call the `acknowledgment_handler` function.

      For example, when the acknowledgement of a successful connection:

   ```
   {"packetType": 2, "packetName": "CONNACK", "status": 0, "message":
   "Success"}
   ```

      is received, the bridge package will call the `acknowledgment_handler` function.

   d. Create an `error_handler` function

   ```
   def error_handler(code: int, message: str):
       print(f"Error {code}: {message}")
   ```

   e. Use the setter property `on_error` to register the `error_handler` function created in the previous step as the callback function for the error event.

   ```
   connection_object.on_error = error_handler
   ```

   f. When the actual error is received, the bridge package will call the `error_handler` function.

      For example, when an error occurs while establishing a connection:

   ```
   (-1, The parameter 'host' should not be empty)
   ```

the bridge package will call the `error_handler` function.

5. Establish a connection by passing

    a. `bridge.iiflcapital.com` as `host`
    b. `9906` as `port`
    c. `<userSession>` as `token`

    `<userSession>` is the session token received in the response after calling the `getUserSessions` API

```
conn_req = '{"host": "bridge.iiflcapital.com", "port": 9906, "token": 
<userSession>}'
connection_object.connect_host(conn_req)
```

6. Subscribe to the required events

```
connection_object.subscribe_order_updates(req)
connection_object.subscribe_trade_updates(req)
```

    Pass your client Id in `req`. For example, if your client ID is CLIENT101, pass

```
req = '{"subscriptionList": ["CLIENT101"]}’
```

7. To receive and process the events

    a. Create `order_updates_handler` and `trade_updates_handler` callback functions

```
def order_updates_handler(data: bytearray, topic: str):
     print(f"Order updates data received on topic {topic}: {data}")

def trade_updates_handler(data: bytearray, topic: str):
    print(f"trade update data received on topic {topic}:{data}")
```

    b. Use the setter property `on_order_updates_received` to register the `order_updates_handler` function created in the previous step as the callback function for the order updates event.

```
connection_object.on_order_updates_received=order_updates_handler
```

    c. Use the setter property `on_trade_updates_received` to register the `trade_updates_handler` function created in the previous steps as the callback functions for the trade updates event.

```
connection_object.on_trade_updates_received=trade_updates_handler
```

d. When the actual order updates and trade updates events are received, the bridge package will call the `order_updates_handler` and `trade_updates_handler` functions respectively.

8. Unsubscribe from the events when they are no longer required

```
connection_object.unsubscribe_order_updates(req)
connection_object.unsubscribe_trade_updates(req)
```

Pass your client Id in `req`. For example, if your client ID is CLIENT101, pass

```
req = '{"subscriptionList": ["CLIENT101"]}'
```

9. Disconnect from the host when it is no longer required

```
connection_object.disconnect_host()
```

The packet structure for both the Order Updates and Trade Updates events are detailed below.

Order Updates

Event Type	**Callback Function**	**Request Param**	**Response Format**	**Description**
Order Updates	on_order_updates_received	Client Id	JSON	Provides real-time updates on the status of placed orders, including modifications, cancellations, and execution progress

Packet JSON

```
{
"clientId: "31625881",
"validity: "DAY",
"orderComplexity: "REGULAR",
"product: "NORMAL",
"orderType: "MARKET",
"tradingSymbol: "IDEA-EQ",
"transactionType: "BUY",
"instrumentId: "14366",
"price: "1500",
"slTriggerPrice: "1600",
"quantity: "1",
"disclosedQuantity: "10",
"cancelledQuantity: "30",
"algoId: "algo123",
"marketProtectionPercent : "1300",
"placedBy: "31625881",
"averageTradedPrice: "1510",
"filledQuantity: "50",
"pendingQuantity: "25",
"brokerOrderId: "240807000000068",
"exchangeOrderId: "900000000000000001",
"rejectionReason: "RMS:Margin Exceeds,****duct",
"orderStatus: "rejected",
"exchangeTimestamp: "07-Aug-2024 16:14:17",
"exchangeUpdateTime: "07-Aug-2024 16:14:17",
"mainLegOrderId : "240807000000068",
"validityDate: "45511",
"source: "API",
"comments: "Strangle~dQcOimFMAvsXlum~SELF",
"brokerUpdateTime": "17-Feb-2025 06:48:39"
}
```

Packet Parameters

Parameter	Data Type	Description	Example Values
clientId	varchar	Unique identifier of the client	31625881
validity	varchar	Duration for which the order is valid (e.g., DAY, IOC)	DAY
orderComplexity	varchar	Complexity of order, such as Regular, AMO, BO, or CO	REGULAR
product	varchar	Type of product used for the order (e.g., intraday or delivery)	NORMAL
orderType	varchar	Specifies the execution type of the order (e.g., Market or Limit)	MARKET
tradingSymbol	varchar	Trading symbol of the instrument	IDEA-EQ
transactionType	varchar	Specifies whether the order is a buy or sell	BUY
instrumentId	varchar	Unique identifier for the financial instrument	14366
price	double	Price at which the order is placed for limit orders	1500
slTriggerPrice	double	Stop-loss trigger price for the order	1600
quantity	int	Number of shares or contracts in the order	1
disclosedQuantity	int	Partial quantity disclosed to the market	10
cancelledQuantity	int	Quantity of the order that was cancelled	30
algoId	varchar	Unique identifier for the algorithm that placed the order	algo123
marketProtectionPercent	double	Market protection ensures that this sell order is not executed below (or buy order above) the specified price, safeguarding against unfavourable price	1300

		movements.	
placedBy	varchar	User(client or dealer) who placed the order	31625881
averageTradedPrice	double	Average price at which the order was executed	1510
filledQuantity	int	Quantity of the order that has been filled	50
pendingQuantity	int	Quantity of the order that remains unfilled	25
brokerOrderId	varchar	Unique order ID assigned by IIFL Capital	240807000000068
exchangeOrderId	varchar	Unique order ID assigned by the exchange	9000000000000001
rejectionReason	varchar	Reason for order rejection, if applicable	RMS:Margin Exceeds,****duct
orderStatus	varchar	Current status of the order (e.g., Open, Complete, Rejected, Canceled)	rejected
exchangeTimestamp	varchar	The timestamp indicating when the order was received by the exchange	07-Aug-2024 16:14:17
exchangeUpdateTime	varchar	Timestamp when the exchange updated the order status	07-Aug-2024 16:14:17
mainLegOrderId	varchar	ID of the bracket order, if this order is part of a Bracket Order	240807000000068
validityDate	varchar	The date until which the order remains valid, based on the specified order validity type (e.g., GTT, GTD, VTD)	8/7/2024
source	varchar	The platform or system from which the order was placed(e.g., Web, App, API)	API
comments	varchar	Additional remarks or notes provided for the order(e.g., order tag, app key, app name)	Strangle~dQcOimFMAvsXlum~SELF
brokerUpdateTime	varchar	Timestamp of the broker's last update on the order	17-Feb-2025 06:48:39

Trade Updates

Event Type	Callback Function	Request Param	Response Format	Description
Trades Updates	on_trade_updates_received	Client Id	JSON	Delivers instant notifications of executed trades with details such as price, quantity, and trade time

Packet JSON

```
{
"tradedPrice: "1560",
"filledQuantity: "50",
"exchangeTradeId: "893487609000000",
"instrumentId: "14366",
"exchange: "NSEEQ",
"clientId: "31625881",
"orderComplexity: "REGULAR",
"product: "NORMAL",
"tradingSymbol: "IDEA-EQ",
"fillDate: "45511",
"fillTime: "0.676585648148148",
"brokerOrderId: "240807000000068",
"exchangeOrderId: "900000000000000",
"transactionType: "BUY",
"orderType: "MARKET",
"placedBy: "31625881",
"algoId: "algo123"
}
```

Packet Parameters

Parameter	Data Type	Description	Example Values
tradedPrice	double	The price at which the order was filled in the market	1560
filledQuantity	int	Quantity of the order that has been filled	50
exchangeTradeId	varchar	Unique trade ID assigned by the exchange	893487609000000
instrumentId	varchar	Unique identifier for the financial instrument	14366
exchange	varchar	The exchange where the order is placed	NSEEQ
clientId	varchar	Unique identifier of the client	31625881
orderComplexity	varchar	Complexity of order, such as Regular, AMO, BO, or CO	REGULAR
product	varchar	Type of product used for the order (e.g., intraday or delivery)	NORMAL
tradingSymbol	varchar	Trading symbol of the instrument	IDEA-EQ
fillDate	varchar	Date when the order was filled	8/7/2024
fillTime	varchar	Time when the order was filled	16:14:17
brokerOrderId	varchar	Unique order ID assigned by IIFL Capital	240807000000068
exchangeOrderId	varchar	Unique order ID assigned by the exchange	900000000000001
transactionType	varchar	Specifies whether the order is a buy or sell	BUY
orderType	varchar	Specifies the execution type of the order (e.g., Market or Limit)	MARKET
placedBy	varchar	User(client or dealer) who placed the order	31625881
algoId	varchar	Unique identifier for the algorithm that placed the order	algo123

Portfolio

The portfolio APIs let you fetch holdings and positions in your portfolio.

Method	Endpoint	Processing Mode	Action
GET	/holdings	Single	The Holdings API returns long-term equity holdings of a client. All the financial instruments in the holdings reside in the customer's DEMAT account indefinitely until they are sold or delisted or altered by the exchanges. Changes to the DEMAT account are settled in T+1 days.
GET	/positions	Single	The Positions API contains all the open positions of the client for the day. This includes all F&O carryforward positions as well.

Holdings

Method	Endpoint	Processing Mode	Action
GET	/holdings	Single	The Holdings API returns long-term equity holdings of a client. All the financial instruments in the holdings reside in the customer's DEMAT account indefinitely until they are sold or delisted or altered by the exchanges. Changes to the DEMAT account are settled in T+1 days.

Headers

Authorization: Bearer <userSession>

Request Structure

No Body

Response Structure

```
{
        "isin": "INE002A01018",
        "nseInstrumentId": "2885",
        "bseInstrumentId": "500325",
        "nseTradingSymbol": "RELIANCE-EQ",
        "bseTradingSymbol": "RELIANCE",
        "formattedInstrumentName": "RELIANCE INDUSTRIES LTD.",
         "product": "DELIVERY",
        "totalQuantity": 300,
        "dpQuantity": 300,
        "collateralQuantity": 0,
        "t1Quantity": 0,
        "authorizedQuantity": "0",
        "averageTradedPrice": "280000.00",
        "previousDayClose": null
    }
```

Response Parameters

Param	Data Type	Description	Example Values
`isin`	varchar	The unique International Securities Identification Number for the financial instrument	`INE257A01026`
`nseInstrumentId`	varchar	The instrument identifier for the security on NSE	`438`
`bseInstrumentId`	varchar	The instrument identifier for the security on BSE	`500183`
`nseTradingSymbol`	varchar	The trading symbol for the security on NSE	`IDEA-EQ`
`bseTradingSymbol`	varchar	The trading symbol for the security on BSE	`IDEA`
`formattedInstrumentName`	varchar	The formatted name of the financial instrument	`IDEA VODAFONE LIMITED`
`product`	varchar	Type of product used for the order (BNPL or delivery)	`DELIVERY/BNPL`
`totalQuantity`	int	The total number of shares held in the account (`totalQuantity = dpQuantity + t1Quantity`)	`50`
`dpQuantity`	int	The number of shares held in the Demat account	`0`
`collateralQuantity`	int	The number of shares pledged as collateral	`0`
`t1Quantity`	int	The quantity of shares pending settlement from yesterday's trades	`0`
`authorizedQuantity`	int	The number of shares authorised for trading or transfer	`0`
`averageTradedPrice`	double	The average price at which the shares were acquired	`40.67`
`previousDayClose`	double	The closing price of the security from the previous trading day	`42.28`

Positions

Method	Endpoint	Processing Mode	Action
GET	/positions	Single	The Positions API contains all the open positions of the client for the day. This includes all F&O carryforward positions as well.

Headers
Authorization: Bearer <userSession>

Request Structure
No Body

Response Structure
{

```
        "instrumentId": "35382",
        "tradingSymbol": "NIFTY24OCTFUT",
        "formattedInstrumentName": "NIFTY 31 Oct 2024",
        "exchange": "NSEFO",
        "product": "NORMAL",
        "netQuantity": 26.0,
        "netAveragePrice": "0",
        "overnightQuantity": 1.0,
        "overnightPrice": "0",
        "buyQuantity": 25.0,
        "buyPrice": 652000.0,
        "sellQuantity": 0.0,
        "sellPrice": 0.0,
        "dayBuyQuantity": "25",
        "dayBuyPrice": 652000.0,
        "dayBuyValue": "627961.54",
        "daySellQuantity": "0",
        "daySellPrice": "0.00",
        "daySellValue": "0.00",
        "multiplier": "1",
        "lotSize": "25",
        "tickSize": "0.05",
        "previousDayClose": "26308.85"
    }
```

Response Parameters

Param	Data Type	Description	Example Values
`instrumentId`	`varchar`	Unique identifier for the financial instrument	`35382`
`tradingSymbol`	`varchar`	Trading symbol of the instrument	`NIFTY24OCTFUT`
`formattedInstrumentName`	`varchar`	Display name of the financial instrument	`NIFTY 31 Oct 2024`
`exchange`	`varchar`	The exchange where the order is placed	`NSEFO`
`product`	`varchar`	Type of product used for the order (e.g., intraday or delivery)	`NORMAL`
`realizedPnl`	`double`	The profit or loss booked from completed trades by closing positions	`1000`
`netQuantity`	`int`	The total number of shares or contracts held after netting off buy and sell quantities `(netQuantity = overnightQuantity + dayBuyQuantity - daySellQuantity)`	`3`

netAveragePrice	double	Average price at which the net quantity was acquired (netAveragePrice = (overnightQuantity*overnightPrice + dayBuyValue - daySellValue)/ netQuantity)	1
overnightQuantity	int	The net quantity of shares or contracts carried forward from the previous trading day	3
overnightPrice	double	The net average price at which the overnight positions were carried forward	1
buyQuantity	int	The total number of shares or contracts purchased (buyQuantity = max(overnightQuantity,0) + dayBuyQuantity)	3
buyPrice	double	The weighted average price at which the buy transactions were executed (buyPrice = (overnightPrice * max(overnightQuantity,0) + dayBuyPrice * dayBuyQuantity)/buyQuantity)	8
sellQuantity	int	The total number of shares or contracts sold (buyQuantity = max(-1*overnightQuantity,0) + daySellQuantity)	8
sellPrice	double	The weighted average price at which the sell transactions were executed (sellPrice = (overnightPrice * max(-1*overnightQuantity,0) + daySellPrice * daySellQuantity)/SellQuantity)	0
dayBuyQuantity	int	The total number of shares or contracts bought today	25
dayBuyPrice	double	The weighted average price of shares or contracts bought today	652000.00
dayBuyValue	double	The total value of buy transactions executed today (dayBuyQuantity * dayBuyPrice)	627961.54
daySellQuantity	int	The total number of shares or contracts sold today	0
daySellPrice	double	The weighted average price of shares or contracts sold today	0
daySellValue	double	The total value of sell transactions executed today (daySellQuantity * daySellPrice)	0
multiplier	double	The quantity or lot size multiplier used for calculating P&Ls	1
lotSize	int	The minimum number of shares or contracts that can be traded as a single unit	25
tickSize	double	The minimum price movement allowed for a given stock or contract	0.05
previousDayClose	double	The closing price of the stock or contract from the previous trading day	487.5

Market Data Stream

Our Bridge Package provides 8 distinct types of events, each tailored to deliver specific types of market data. To receive these events, you must register their designated callback functions. All Market Data events are encoded in binary format, with a fixed length and a fixed sequential structure. The sequential structure of each event is provided separately.

There can be a maximum of 4000 event subscriptions per client but only a maximum of 1024 event subscription requests can be sent at once.

Below is an overview of each event and the information it delivers.

Event Type	Setter Property to register callbacks	Response Format	Length (bytes)	Description
Market Feed	on_feed_data_received	binary	188	Provides updates on prices, trading volumes, and market depth for equities, FnO and indices
Open Interest	on_open_interest_data_received	binary	16	Provides updates on the current, day's highest and day's lowest open interest levels for F&O contracts
Market Status	on_market_status_data_received	binary	2	Provides opening and closing events of the multiple sessions of the normal market
Upper Circuit	on_upper_circuit_data_received	binary	12	Provides events when a security reaches the upper limit of its daily price range (DPR)
Lower Circuit	on_lower_circuit_data_received	binary	12	Provides events when a security reaches the lower limit of its daily price range (DPR)
LPP	on_lpp_data_received	binary	12	Provides updates on limit order price range for F&O contracts
52 Week High	on_high_52_week_data_received	binary	12	Provides alerts when an instrument reaches a new 52-week high
52 Week Low	on_low_52_week_data_received	binary	12	Provides alerts when an instrument reaches a new 52-week low

Follow the steps below to receive order and trade update events. The example uses the Python SDK for illustration and does not include the complete code. Please refer to the full SDK implementation available in multiple languages on our GitHub portal for integration.

1. Install the BridgePy package

```
pip install bridgePy
```

2. Import connector module from BridgePy package

```
from bridgePy import connector as connector
```

3. Create an object of the connect class

```
connection_object = connector.Connect()
```

4. Before establishing a connection, complete the below steps to receive and handle acknowledgments and error messages effectively:

a. Create an `acknowledgment_handler` function

```
def acknowledgment_handler(response: str):
    print(f"Acknowledgment: {response}")
```

b. A setter property is responsible for registering a callback function. When you pass a function to it, the setter will save the function so that it can be used later when the event arrives.

   Use the setter property `on_acknowledge_response` to register the `acknowledgment_handler` function created in the previous step as the callback function for the acknowledgement event.

```
connection_object.on_acknowledge_response =acknowledgment_handler
```

c. When the actual acknowledgment status is received, the bridge package will call the `acknowledgment_handler` function.

   For example, when the acknowledgement of a successful connection:

```
{"packetType": 2, "packetName": "CONNACK", "status": 0, "message":
"Success"}
```

   is received, the bridge package will call the `acknowledgment_handler` function.

d. Create an `error_handler` callback function

```
def error_handler(code: int, message: str):
    print(f"Error {code}: {message}")
```

e. Use the setter property `on_error` to register the `error_handler` function created in the previous step as the callback function for the error event.

```
connection_object.on_error =error_handler
```

f. When the actual error is received, the bridge package will call the `error_handler` function.

   For example, when an error occurs while establishing a connection:

```
(-1, The parameter 'host' should not be empty)
```

   the bridge package will call the `error_handler` function.

4. Establish a connection by passing

   a. `bridge.iiflcapital.com` as `host`

   b. `9906` as `port`
   c. `<userSession>` as `token`

`<userSession>` is the session token received in the response after calling the `getusersessions` API

```
conn_req = '{"host": "bridge.iiflcapital.com", "port": 9906, "token":
<userSession>}'
connection_object.connect_host(conn_req)
```

5. Subscribe to the required events

```
connection_object.subscribe_feed(req)
```

Market Feed example: if you wish to receive updates on prices, trading volumes, and market depth for the RELIANCE INDUSTRIES LTD stock and NIFTY 26th DEC FUT future, pass

```
req = '{"subscriptionList": ["nseeq/2885","nsefo/35005"]}'
```

6. To receive and process the events

   d. Create a market data stream handler callback function. For the Market Feed example given above, create a `market_feed_handler` function

   ```
   def market_feed_handler(data: bytearray, topic: str):
       print(f"Feed data received on topic {topic}: data {Data}"
   ```

   Refer to the annexure sample conversion of a binary event to decimal format, which demonstrates the conversion of a market feed event in binary format to a readable decimal format.

   The same steps apply to any type of market data event. Include the code for each step outlined in the example within all your market data handler functions.

   e. Use the setter property of the event to register the function created in the previous step as the callback function for the event.

   Continuing with the Market Feed example: Use the setter property `on_feed_data_received` to register the `market_feed_handler` function created in the previous step as the callback function for the market feed event.

   ```
   connection_object.on_feed_data_received =market_feed_handler
   ```

   f. When the actual market data event is received, the bridge package will call the market data handler function.

   In the above Market Feed example, when the actual market feed event is received, the bridge package will call the `market_feed_handler` function.

7. Unsubscribe from the events when they are no longer required

   ```
   connection_object.unsubscribe_feed(req)
   ```

   Continuing our Market Feed example, if you wish to stop receiving updates on prices, trading volumes, and market depth for the RELIANCE INDUSTRIES LTD stock and NIFTY 26th DEC FUT future, pass

   ```
   req = '{"subscriptionList": ["nseeq/2885","nsefo/35005"]}'
   ```

8. Disconnect from the host when it is no longer required

   ```
   connection_object.disconnect_host()
   ```

The packet structure for each market data event is detailed below, along with a few examples for clarification.

Market Feed

Event Type	Callback Function	Response Format	Length (bytes)	Description
Market Feed	on_feed_data_received	binary	188	Provides updates on prices, trading volumes, and market depth for equities, FnO and indices

All Market Feed events adhere to the template: `<exchange>/<instrumentId>`

Event Name Example	Description
nseeq/2885	Provides updates on prices, trading volumes, and market depth for the RELIANCE INDUSTRIES LTD stock
nsefo/35005	Provides updates on prices, trading volumes, and market depth for NIFTY 26th DEC FUT future
nsefo/35217	Provides updates on prices, trading volumes, and market depth for NIFTY 26th DEC 30000 CE option
bseeq/999901	Provides updates on prices for the SENSEX index(market depth and volume entries will be 0 for indices)

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	Data Type	Size	Bytes	Description
`ltp`	`Int32`	4	0-3	Last traded price of the instrument
`lastTradedQuantity`	`UInt32`	4	4-7	Quantity of the last trade executed
`tradedVolume`	`UInt32`	4	8-11	Total volume of trades for the instrument

high	Int32	4	12-15	Highest price reached during the session
low	Int32	4	16-19	Lowest price reached during the session
open	Int32	4	20-23	Opening price of the trading session
close	Int32	4	24-27	Closing price of the previous trading session
averageTradedPrice	Int32	4	28-31	Average price of all trades during the session
reserved	UInt16	2	32-33	bytes reserved for future usage, can be safely ignored
bestBidQuantity	UInt32	4	34-37	Quantity available at the best buy price
bestBidPrice	Int32	4	38-41	Highest price available for buying
bestAskQuantity	UInt32	4	42-45	Quantity available at the best sell price
bestAskPrice	Int32	4	46-49	Lowest price available for selling
totalBidQuantity	UInt32	4	50-53	Total quantity available for buying
totalAskQuantity	UInt32	4	54-57	Total quantity available for selling
priceDivisor	Int32	4	58-61	The integer by which all the prices should be divided to obtain the actual price in decimals
lastTradedTime	Int32	4	62-65	The timestamp of the most recently executed trade
marketDepth		120	66-185	[]byte - byte array containing 10 depth arrays
marketDepth[].bids[].quantity1	UInt32	4	66-69	Total quantity available at the highest bid
marketDepth[].bids[].price1	Int32	4	70-73	Highest price buyers are willing to pay
marketDepth[].bids[].orders1	Int16	2	74-75	Number of orders at the highest bid
To be ignored	Int16	2	76-77	To be ignored
marketDepth[].bids[].quantity2	UInt32	4	78-81	Total quantity available at the 2nd highest bid
marketDepth[].bids[].price2	Int32	4	82-85	2nd highest price buyers are willing to pay
marketDepth[].bids[].orders2	Int16	2	86-87	Number of orders at the 2nd highest bid
To be ignored	Int16	2	88-89	To be ignored
marketDepth[].bids[].quantity3	UInt32	4	90-93	Total quantity available at the 3rd highest bid
marketDepth[].bids[].price3	Int32	4	94-97	3rd highest price buyers are willing to pay
marketDepth[].bids[].orders3	Int16	2	98-99	Number of orders at the 3rd highest bid
To be ignored	Int16	2	100-101	To be ignored
marketDepth[].bids[].quantity4	UInt32	4	102-105	Total quantity available at the 4th highest bid
marketDepth[].bids[].price4	Int32	4	106-109	4th highest price buyers are willing to pay
marketDepth[].bids[].orders4	Int16	2	110-111	Number of orders at the 4th highest bid
To be ignored	Int16	2	112-113	To be ignored
marketDepth[].bids[].quantity5	UInt32	4	114-117	Total quantity available at the 5th highest bid
marketDepth[].bids[].price5	Int32	4	118-121	5th highest price buyers are willing to pay
marketDepth[].bids[].orders5	Int16	2	122-123	Number of orders at the 5th highest bid
To be ignored	Int16	2	124-125	To be ignored

`marketDepth[].asks[].quantity1`	`UInt32`	4	126-129	Total quantity available at the lowest ask
`marketDepth[].asks[].price1`	`Int32`	4	130-133	Lowest price sellers are willing to accept
`marketDepth[].asks[].orders1`	`Int16`	2	134-135	Number of orders at the lowest ask
`To be ignored`	`Int16`	2	136-137	To be ignored
`marketDepth[].asks[].quantity2`	`UInt32`	4	138-141	Total quantity available at the 2nd lowest ask
`marketDepth[].asks[].price2`	`Int32`	4	142-145	2nd lowest price sellers are willing to accept
`marketDepth[].asks[].orders2`	`Int16`	2	146-147	Number of orders at the 2nd lowest ask
`To be ignored`	`Int16`	2	148-149	To be ignored
`marketDepth[].asks[].quantity3`	`UInt32`	4	150-153	Total quantity available at the 3rd lowest ask
`marketDepth[].asks[].price3`	`Int32`	4	154-157	3rd lowest price sellers are willing to accept
`marketDepth[].asks[].orders3`	`Int16`	2	158-159	Number of orders at the 3rd lowest ask
`To be ignored`	`Int16`	2	160-161	To be ignored
`marketDepth[].asks[].quantity4`	`UInt32`	4	162-165	Total quantity available at the 4th lowest ask
`marketDepth[].asks[].price4`	`Int32`	4	166-169	4th lowest price sellers are willing to accept
`marketDepth[].asks[].orders4`	`Int16`	2	170-171	Number of orders at the 4th lowest ask
`To be ignored`	`Int16`	2	172-173	To be ignored
`marketDepth[].asks[].quantity5`	`UInt32`	4	174-177	Total quantity available at the 5th lowest ask
`marketDepth[].asks[].price5`	`Int32`	4	178-181	5th lowest price sellers are willing to accept
`marketDepth[].asks[].orders5`	`Int16`	2	182-183	Number of orders at the 5th lowest ask
`To be ignored`	`Int16`	2	184-185	To be ignored
`To be ignored`	`Int16`	2	186-187	To be ignored

Open Interest Stream

Event Type	**Callback Function**	**Response Format**	**Length (bytes)**	**Description**
Open Interest	on_open_interest_data_received	binary	16	Provides updates on the current, day's highest and day's lowest open interest levels for F&O contracts

All Open Interest events adhere to the template: `<exchange>/<instrumentId>`

Event Name Example	**Description**
nsefo/35005	Provides updates on open interest for NIFTY 26th DEC FUT future
nsefo/35217	Provides updates on open interest for NIFTY 26th DEC 30000 CE option

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	**Data**	**Size**	**Bytes**	**Description**

	Type			
`openInterest`	`Int32`	4	0-3	Total number of outstanding derivative contracts
`dayHighOi`	`Int32`	4	4-7	Highest open interest recorded during the day
`dayLowOi`	`Int32`	4	8-11	Lowest open interest recorded during the day
`previousOi`	`Int32`	4	12-15	Open interest from the previous trading session

Market Status

Event Type	Callback Function	Response Format	Length (bytes)	Description
Market Status	on_market_status_data_received	binary	2	Provides opening and closing events of the multiple sessions of the normal market

All Market Status events adhere to the template: `<exchange>`

Event Name Example	Description
nseeq	Provides the opening and closing events of the NSE equity segment's normal market

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	Data Type	Size	Bytes	Description
`MarketStatusCode`	`Int16`	2	0-1	A code for the exchange status

Find the Exchange status codes and their description below:

Exchange status Code	Exchange Status
0	Pre-Open Started
1	Pre-Open Closed
2	Market opened
3	Call Auction Started
4	Call Auction Closed
5	Auction Market Started
6	Auction Market Closed
7	Market Closed
8	Closing Session has opened
9	Closing Session has Closed
10	Halt

Upper Circuit Change

Event Type	Callback Function	Response Format	Length (bytes)	Description
Upper Circuit	on_upper_circuit_data_received	binary	12	Provides events when a security reaches the upper limit of its daily price range (DPR)

All Upper Circuit Change events adhere to the template: `<exchange>`

Event Name	Description
nseeq	Provides events when a security reaches the upper limit of its daily price range (DPR) in the equity segment of NSE

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	Data Type	Size	Bytes	Description
`instrumentId`	`UInt32`	4	0-3	Unique identifier for the financial instrument
`upperCircuit`	`UInt32`	4	4-7	the upper limit of the daily price range (DPR) reached by the security
`priceDivisor`	`Int32`	4	8-11	The integer by which the `upperCircuit` price should be divided to obtain the actual price in decimals

Lower Circuit Change

Event Type	Callback Function	Response Format	Length (bytes)	Description
Lower Circuit	on_lower_circuit_data_received	binary	12	Provides events when a security reaches the lower limit of its daily price range (DPR)

All Lower Circuit Change events adhere to the template: `<exchange>`

Event Name	Description
nseeq	Provides events when a security reaches the lower limit of its daily price range (DPR) in the equity segment of NSE

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	Data Type	Size	Bytes	Description
`instrumentId`	`UInt32`	4	0-3	Unique identifier for the financial instrument
`lowerCircuit`	`UInt32`	4	4-7	the lower limit of the daily price range (DPR) reached by the security
`priceDivisor`	`Int32`	4	8-11	The integer by which the `lowerCircuit` price should be divided to obtain the actual price in decimals

Limit Price Protection(LPP) Change

Event Type	Callback Function	Response Format	Length (bytes)	Description
LPP	on_lpp_data_received	binary	12	Provides updates on limit order price range for F&O contracts

All Limit Price Protection(LPP) Change events adhere to the template: `<exchange>/<instrumentId>`

Event Name Example	Description
nsefo/35005	Provides updates on limit order price range for the future contract NIFTY 26th DEC FUT
nsefo/35217	Provides updates on limit order price range for the Options contract NIFTY 26th DEC 30000 CE

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	Data Type	Size	Bytes	Description
`lppHigh`	`UInt32`	4	0-3	updated high execution band for limit order for the FnO contract
`lppLow`	`UInt32`	4	4-7	updated low execution band for limit order for the FnO contract
`priceDivisor`	`Int32`	4	8-11	The integer by which lppHigh and lppLow prices should be divided to obtain the actual price in decimals

52 Week High Change

Event Type	Callback Function	Response Format	Length (bytes)	Description
52 Week High	on_high_52_week_data_received	binary	12	Provides alerts when an instrument reaches a new 52-week high

All 52 Week High Change events adhere to the template: `<exchange>`

Event Name Example	Description
nseeq	Provides alerts when an instrument reaches a new 52-week high in the equity segment of NSE

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	Data Type	Size	Bytes	Description
`instrumentId`	`UInt32`	4	0-3	Unique identifier for the financial instrument

`52WeekHigh`	`UInt32`	4	4-7	updated 52 week high price for the stock
`priceDivisor`	`Int32`	4	8-11	The integer by which the `52WeekHigh` should be divided to obtain the actual price in decimals

52 Week Low Change

Event Type	Callback Function	Response Format	Length (bytes)	Description
52 Week Low	on_low_52_week_data_received	binary	12	Provides alerts when an instrument reaches a new 52-week low

All 52 Week Low Change events adhere to the template: `<exchange>`

Event Name Example	Description
nseeq	Provides alerts when an instrument reaches a new 52-week low in the equity segment of NSE

*Note: **All event names are case-sensitive and must be provided in lowercase.**

Packet Structure

Param	Data Type	Size	Bytes	Description
`instrumentId`	`UInt32`	4	0-3	Unique identifier for the financial instrument
`52WeekLow`	`UInt32`	4	4-7	updated 52 week low price for the stock
`priceDivisor`	`Int32`	4	8-11	The integer by which the `52WeekLow` should be divided to obtain the actual price in decimals

Market Data APIs

The 4 Market data APIs provide real-time and historical information, including prices, volumes, open interest, and market depth.

Method	Endpoint	Processing Mode	Action
POST	/marketdata/historicaldata	Single	Provides historical candlestick data (open, high, low, close and volume) for the specified instrument over a chosen time frame
POST	/marketdata/marketquotes	Single/Bulk	Delivers real-time market data, including price and volume, for selected instruments
POST	/marketdata/marketdepth	Single	Returns market depth details, showing the best buy/sell prices and order quantities at different price levels
POST	/marketdata/openinterest	Single	Provides open interest data for futures and options, indicating total number of outstanding futures or options contracts that have

			not been settled or closed

Historical Candlestick Chart Data

Method	Endpoint	Processing Mode	Action
POST	/marketdata/ historicaldata	Single	Provides historical candlestick data (open, high, low, close and volume) for the specified instrument over a chosen time frame

Headers

Authorization: Bearer <userSession>

Request Structure

```
{
“exchange”: “NSEEQ”,
“InstrumentId”: “1594”,
“interval”: “1 minute”,
“fromDate”: “19-Sep-2024”,
“toDate”: “20-Sep-2024”
}
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values
exchange *Required*	varchar	Exchange & segment of the instrument to be traded	NSEEQ,NSEFO,BSEEQ,BSEFO,NSECURR,BSECURR,MCXCOMM,NCDEXCOMM,NSECOMM,BSECOMM
instrumentId *Required*	varchar	Unique identifier for the financial instrument	1594
interval *Required*	varchar	Interval of each candle	1 minute, 5 minutes, 10 minutes, 15 minutes, 30 minutes, 60 minutes, 1 day, weekly, monthly
fromDate *Required*	varchar	from date	19-Sep-2024
toDate *Required*	varchar	to date	20-Sep-2024

Response Structure

Note: The Historical Candlestick Chart Data API response is returned in string format(& not in JSON format).

```
{
   "status": "Ok",
    "message": "Success",
    "result":
[
          {
           “2024-11-11T09:15:00”, //initial timestamp
           292.0, //open
           294.0, //high
           289.5, //low
           293.55, //close
           11025420 //volume
         },
         {
           "2024-11-11T09:16:00", //initial timestamp
           295.0, //open
           296.3, //high
           293.75, //low
           295.3, //close
           11315876 //volume
         },
         {
           "2024-11-11T09:17:00", //initial timestamp
           295.2, //open
           295.6, //high
           292.7, //low
           294.45, //close
           9149737 //volume
         }
]
}
```

Response Parameters

Param	Data Type	Description	Example Values
initialTimestamp	varchar	Starting time of the candlestick period	19-Sep-2024 09:17:00
open	double	Price at the start of the candlestick period	295.2
high	double	Highest price reached during the candlestick period	295.6
low	double	Lowest price reached during the candlestick period	292.7
close	double	Price at the end of the candlestick period	294.45
volume	int	Total number of shares or contracts traded during the candlestick period	9149737

Market Quotes

Method	Endpoint	Processing Mode	Action
POST	/marketdata/ marketquotes	Single/Bulk	Delivers real-time market data, including price and volume, for selected instruments

Headers
Authorization: Bearer <userSession>

Request Structure
```
[
{
“exchange”: “NSEEQ”,
“InstrumentId”: “1594”
},
{
“exchange”: “NSEEQ”,
“InstrumentId”: “2885”
}
]
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values
exchange *Required*	varchar	Exchange & segment of the instrument to be traded	NSEEQ,NSEFO,BSEEQ,BSEFO,NSECURR,BSECURR,MCXCOMM,NCDEXCOMM,NSECOMM,BSECOMM
instrumentId *Required*	varchar	Unique identifier for the financial instrument	1594

Response Structure
```
{
  "status": "Ok",
  "message": "Success",
  "result":
 [
{
“exchange”: "NSEEQ",
“instrumentId”: 1594,
“ltp”: 1412.95,
“lastTradedQuantity”: 5,
“averageTradedPrice”: 1412.47,
“tradedVolume”: 7360198,
“open”: 1396,
“high”: 1421.75,
“low”: 1395.55,
“close”: 1389.65,
"bestBidPrice": 2994.25,
"bestBidQuantity": 4,
"besAskPrice": 2994.75,
"bestAskQuantity": 13,
"totalBidQuantity": 404715,
"totalAskQuantity": 216809,
"tickTimestamp": "19-Sep-2024 09:15:00"
},
{
“exchange”: "NSEEQ",
“instrumentId”: 2885,
“ltp”: 1412.95,
“lastTradedQuantity”: 5,
“averageTradedPrice”: 1412.47,
“tradedVolume”: 7360198,
“open”: 1396,
“high”: 1421.75,
“low”: 1395.55,
“close”: 1389.65,
"bestBidPrice": 2994.25,
"bestBidQuantity": 4,
"bestAskPrice": 2994.75,
"bestAskQuantity": 13,
"totalBidQuantity": 404715,
"totalAskQuantity": 216809,
"tickTimestamp": "19-Sep-2024 09:15:00"
}
]
```

Response Parameters

Param	Data Type	Description	Example Values
`exchange`	`varchar`	The exchange where the order is placed	`NSEEQ`
`instrumentId`	`varchar`	Unique identifier for the financial instrument	`1594`
`ltp`	`double`	Last traded price of the instrument	`1412.95`

lastTradedQuantity	int	Quantity of the last trade executed	5
averageTradedPrice	double	Average price of all trades during the session	1412.47
tradedVolume	int	Total volume of trades for the instrument	7360198
open	double	Opening price of the trading session	1396
high	double	Highest price reached during the session	1421.75
low	double	Lowest price reached during the session	1395.55
close	double	Closing price of the previous trading session	1389.65
bestBidPrice	double	Highest price available for buying	2994.25
bestBidQuantity	int	Quantity available at the best buy price	4
bestAskPrice	double	Lowest price available for selling	2994.75
bestAskQuantity	int	Quantity available at the best sell price	13
totalBidQuantity	int	Total quantity available for buying	404715
totalAskQuantity	int	Total quantity available for selling	216809
tickTimestamp	varchar	Timestamp of the latest price update	19-Sep-2024 9:15:00

Market Depth

Method	Endpoint	Processing Mode	Action
POST	/marketdata/marketdepth	Single	Returns market depth details, showing the best buy/sell prices and order quantities at different price levels

Headers

Authorization: Bearer <userSession>

Request Structure

```
{
“exchange”: “NSEEQ”,
“InstrumentId”: “1594”
}
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values
exchange *Required*	varchar	Exchange & segment of the instrument to be traded	NSEEQ,NSEFO,BSEEQ,BSEFO,NSECURR,BSECURR,MCXCOMM,NCDEXCOMM,NSECOMM,BSECOMM
instrumentId *Required*	varchar	Unique identifier for the financial instrument	1594

Response Structure

```
{
   "status": "Ok",
   "message": "Success",
   "result":
{
“exchange”: "NSEFO",
“instrumentId”: 408065,
"totalBidQuantity": 404715,
"totalAskQuantity": 216809,
"marketDepth":
{
“bids”: [
{
“price”: 110,
“quantity”: 10,
“orders”: 30
},
{
“price”: 110,
“quantity”: 10,
“orders”: 30
},
{
“price”: 110,
“quantity”: 10,
“orders”: 30
},
{
“price”: 110,
“quantity”: 10,
“orders”: 30
},
{
“price”: 110,
“quantity”: 10,
“orders”: 30
}
],
asks: [
{
```

```
"price": 14311,
"quantity": 100,
"orders": 4
},
{
"price": 110,
"quantity": 10,
"orders": 30
},
{
"price": 110,
"quantity": 10,
"orders": 30
},
{
"price": 110,
"quantity": 10,
"orders": 30
},
{
"price": 110,
"quantity": 10,
"orders": 30
}
]
}
}
}
```

Response Parameters

Param	Data Type	Description	Example Values
exchange	varchar	The exchange where the order is placed	NSEFO
instrumentId	varchar	Unique identifier for the financial instrument	408065
totalBidQuantity	int	Total quantity available for buying	404715
totalAskQuantity	int	Total quantity available for selling	216809
price	double	Price at a specific bid or ask level	0
quantity	int	Total quantity available at the given price level	0
orders	int	Number of orders at the specified price level	0

Open Interest

Method	Endpoint	Processing Mode	Action
POST	/marketdata/openinterest	Single	Provides open interest data for futures and options, indicating total number of outstanding futures or options contracts that have not been settled or closed

Headers

Authorization: Bearer <userSession>

Request Structure

```
{
“exchange”: “NSEEQ”,
“InstrumentId”: “1594”
}
```

Request Parameters

Parameter	Data Type	Description	Approved/Example Values
exchange *Required*	varchar	Exchange & segment of the instrument to be traded	NSEEQ,NSEFO,BSEEQ,BSEFO,NSECURR,BSECURR,MCXCOMM,NCDEXCOMM,NSECOMM,BSECOMM
instrumentId *Required*	varchar	Unique identifier for the financial instrument	1594

Response Structure

```
{
"exchange": "NSEFO",
"instrumentId": 408065,
"openInterest": 7239000,
"dayHighOi": 87863000,
"dayLowOi": 69892000
}
```

Response Parameters

Param	Data Type	Description	Example Values
exchange	varchar	The exchange where the order is placed	NSEFO
instrumentId	varchar	Unique identifier for the financial instrument	408065
openInterest	int	Total number of outstanding derivative contracts	7239000
dayHighOi	int	Highest open interest recorded during the day	87863000
dayLowOi	int	Lowest open interest recorded during the day	69892000

Annexure

Instrument Details

Get comprehensive exchange and segment-wise data of all active market instruments in **CSV** and **JSON** formats.

CSV Links:

Exchange & Segment	**Instrument Details CSV File Link**
NSEEQ	https://api.iiflcapital.com/v1/contractfiles/NSEEQ.csv
NSEFO	https://api.iiflcapital.com/v1/contractfiles/NSEFO.csv
NSECOMM	https://api.iiflcapital.com/v1/contractfiles/NSECOMM.csv
MCXCOMM	https://api.iiflcapital.com/v1/contractfiles/MCXCOMM.csv
INDICES	https://api.iiflcapital.com/v1/contractfiles/INDICES.csv
NSECURR	https://api.iiflcapital.com/v1/contractfiles/NSECURR.csv
BSEEQ	https://api.iiflcapital.com/v1/contractfiles/BSEEQ.csv
BSEFO	https://api.iiflcapital.com/v1/contractfiles/BSEFO.csv
BSECURR	https://api.iiflcapital.com/v1/contractfiles/BSECURR.csv

JSON Links:

Exchange & Segment	**Instrument Details JSON Link**
NSEEQ	https://api.iiflcapital.com/v1/contractfiles/NSEEQ.json
NSEFO	https://api.iiflcapital.com/v1/contractfiles/NSEFO.json
NSECOMM	https://api.iiflcapital.com/v1/contractfiles/NSECOMM.json
MCXCOMM	https://api.iiflcapital.com/v1/contractfiles/MCXCOMM.json
INDICES	https://api.iiflcapital.com/v1/contractfiles/INDICES.json
NSECURR	https://api.iiflcapital.com/v1/contractfiles/NSECURR.json
BSEEQ	https://api.iiflcapital.com/v1/contractfiles/BSEEQ.json
BSEFO	https://api.iiflcapital.com/v1/contractfiles/BSEFO.json
BSECURR	https://api.iiflcapital.com/v1/contractfiles/BSECURR.json

Note: These **JSON** links can also serve as **APIs**, allowing you to retrieve instrument details using the GET method. No authorization is required to make these API calls.

Postman Collection

You can download the official Postman API Collection here.

SDKs

Below is a list of pre-built official libraries for IIFL Markets' APIs developed in popularly used programming languages that can be used to interact with the APIs without having to make raw HTTP calls.

- Python library
- .Net Framework 4.6 library
- .Net Core 8 library
- Golang library

- Java library
- Node JS library

Video Tutorials

1. Trading API Document overview: A quick walkthrough of the key sections and API details in our documentation.
https://youtu.be/t7TMp4SldJ4

2. Daily client login flow: Step-by-step guide for clients to perform daily logins on the trading applications.
https://youtu.be/DMvSVncJOZA

Sample conversion of a binary event to decimal format

Each piece of data, such as prices, quantities, or timestamps, is encoded in **binary** and represented as **hexadecimal** strings for efficient storage and transmission. Binary encoding reduces the amount of space required and allows faster processing, which is essential for high-speed trading systems. To use this data effectively for trading, it must be converted from **hexadecimal** to **decimal** format.

Let's understand the 3 number formats first.
Binary (Base-2):
Uses two digits, 0 and 1. Commonly used in computing as it represents machine-level data.
Example: $Binary\,1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11$

Decimal (Base-10):
The standard number system used in daily life, consisting of digits 0-9.
Example: $Decimal\,155 = 1 \times 10^2 + 5 \times 10^1 + 5 \times 10^0 = 155$

Hexadecimal (Base-16):
Uses digits 0-9 and letters A-F to represent values 10-15. Commonly used in programming to represent large binary values compactly.
Example: $Hexadecimal\,9B = 9 \times 16^1 + 11 \times 16^0 = 155$

*Each binary digit is 1 bit, and each hexadecimal digit is $log_2(16) = 4$ bits. Hence, **1 byte(8 bits) is represented by 2 two hexadecimal characters** (e.g., 10, F3, etc.)

Follow the below steps to convert a binary event into a readable decimal format:

1. Read and Parse the Hex Data: Begin with the raw hex dump and read each byte in sequence. A byte is represented by two hexadecimal characters (e.g., 10, F3, etc.).

Example: Let's assume we receive a hexadecimal Market Quotes and Depth Feed like this

00000000: 10 F3 01 00 74 00 00 00 E4 54 75 00 58 F6 01 00 ....t....Tu.X...
00000010: EC EE 01 00 97 F3 01 00 77 F5 01 00 63 F2 01 00 ....w...c...
00000020: 4F 7D ED 02 00 00 06 F3 01 00 84 00 00 00 24 F3 O}............$.

00000030: 01 00 EA 8A 0B 00 1B AD 09 00 64 00 00 00 56 CD ....d...V.
00000040: 31 67 ED 02 00 00 06 F3 01 00 06 00 00 00 AE 01 1g..............
00000050: 00 00 FC F2 01 00 09 00 00 00 04 00 00 00 F7 F2 ....
00000060: 01 00 01 00 00 00 53 01 00 00 F2 F2 01 00 09 00 ......S.........
00000070: 00 00 51 01 00 00 ED F2 01 00 0B 00 00 00 84 00 ..Q.............
00000080: 00 00 24 F3 01 00 04 00 00 00 15 00 00 00 29 F3 ..$...........).
00000090: 01 00 03 00 00 00 46 05 00 00 2E F3 01 00 0A 00 ......F.........
000000a0: 00 00 C3 02 00 00 33 F3 01 00 06 00 00 00 48 03 ......3.......H.
000000b0: 00 00 38 F3 01 00 0A 00 00 00 00 00

2. Remove Address Offsets and ASCII interpretations: Remove any address offsets (like 00000000, 00000010, etc.) at the start of each line. Ignore any ASCII interpretations (like ....t....Tu.X..., etc.) at the end of each line. You only need the hex values.

Example: Remove address offsets and ASCII interpretations from each row

10 F3 01 00 74 00 00 00 E4 54 75 00 58 F6 01 00
EC EE 01 00 97 F3 01 00 77 F5 01 00 63 F2 01 00
4F 7D ED 02 00 00 06 F3 01 00 84 00 00 00 24 F3
01 00 EA 8A 0B 00 1B AD 09 00 64 00 00 00 56 CD
31 67 ED 02 00 00 06 F3 01 00 06 00 00 00 AE 01
00 00 FC F2 01 00 09 00 00 00 04 00 00 00 F7 F2
01 00 01 00 00 00 53 01 00 00 F2 F2 01 00 09 00
00 00 51 01 00 00 ED F2 01 00 0B 00 00 00 84 00
00 00 24 F3 01 00 04 00 00 00 15 00 00 00 29 F3
01 00 03 00 00 00 46 05 00 00 2E F3 01 00 0A 00
00 00 C3 02 00 00 33 F3 01 00 06 00 00 00 48 03
00 00 38 F3 01 00 0A 00 00 00 00 00

3. Remove Whitespace and Format as Continuous Hex String: Concatenate all the hex bytes into a single continuous string without spaces (e.g., 10F3010074000000E454750058F6...).

Example: Remove Whitespace and Format as Continuous Hex String
10F3010074000000E454750058F60100ECEE010097F3010077F5010063F201004F7DED02000006F30
1008400000024F30100EA8A0B001BAD09006400000056CD3167ED02000006F3010006000000AE010
000FCF201000900000004000000F7F201000100000053010000F2F201000900000051010000EDF2010
00B0000008400000024F301000400000015000000029F301000300000460500002EF301000A000000C
302000033F3010006000000480300038F301000A0000000000

4. Split into 4 or 2 Byte Groups: Divide the continuous hex string into groups of 4 or 2 bytes each as per the packet structure provided here. Since each byte is represented by 2 hexadecimal characters, a 4-byte group will consist of 8 hexadecimal characters, while a 2-byte group will contain 4 hexadecimal characters.

Example: Split into 4 or 2-Byte Groups as per the packet structure

10F30100
74000000
E4547500
58F60100
...

5. Reverse bytes: byte values are represented in little-endian format, so you need to reverse the bytes(set of 2 hex characters) within each byte group to get the correct value.(e.g., the 4 byte group 10F30100 should get reversed to 0001F310)

Example: Reverse the bytes within each group

0001F310
00000074
007554E4
0001F658
...

6. Convert Each Group to Decimal Number: Convert each 8 or 4-character (4 or 2-byte) group to its decimal number.

Example: Convert each byte group to its decimal number.

`ltp`	127760	$0001F310 = 1 \times 16^4 + 15 \times 16^3 + 3 \times 16^2 + 1 \times 16^1 + 0 \times 16^0$
`lastTradedQuantity`	116	$00000074 = 7 \times 16^1 + 4 \times 16^0$
`tradedVolume`	7689444	$07554E4 = 7 \times 16^5 + 5 \times 16^4 + 5 \times 16^3 + 4 \times 16^2 + 14 \times 16^1 + 4 \times 16^0$
`high`	128600	$0001F658 = 1 \times 16^4 + 15 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 8 \times 16^0$
...	...	$\ldots$

7. Divide by price divisor: Divide the converted decimal number by the price divisor if the byte group represents a price

Example: The price divisor value is located in bytes 58-61 of the Market Quotes and Depth Feed. For this example, the price divisor is **100**. Divide the ltp and high prices, **127760** and **128600** respectively, by the price divisor to get their final values.

`ltp`	1277.6

`lastTradedQuantity`	116
`tradedVolume`	7689444
`high`	1286
...	...

8. Convert into human timestamp: Timestamps are sent in Unix epoch time format, convert the decimal number into human readable format if the byte group represents a timestamp.

Example: None of the 4 byte groups in this example represent timestamps. So, this step should be skipped for these 4 byte groups.

9. Get additional information from the topic name: The topic name for this message is nseeq/2885. Extract exchange and instrumentId from the topic name following the topic structure given here.

`exchange`	NSEEQ
`instrumentId`	2885

10. Output the Results: Store each byte group final values into appropriate variables along with the topic details. Repeat until all data is processed.

`exchange`	NSEEQ
`instrumentId`	2885
`ltp`	1277.6
`lastTradedQuantity`	116
`tradedVolume`	7689444
`high`	1286
...	...

Developer's Community

🐞 Found a bug or facing an issue? Raise it on our GitHub Issues page. Alternatively, you may write to us at openapisupport@iiflcapital.com or reach us by phone or Whatsapp at +91-7718830851.

Trading API Error Codes

All the errors that the trading APIs can throw are mentioned below along with their error message. Note that Order Rejection reasons are not errors from APIs and have not been included in the below list.

Error Code	**Error Message**
EC003	Something Went Wrong, Please Try After Sometime.
EC900	Invalid parameter : 'exchange' cannot be empty or null.

EC901	Invalid parameter : 'exchange' Accepts only {'NSEEQ','NSEFO','BSEEQ','BSEFO','NSECURR','BSECURR','MCXCOMM','NCDEXCOMM','NSECOMM' ,'BSECOMM'}
EC902	Invalid parameter : 'tradingSymbol' cannot be empty or null.
EC903	Invalid parameter : 'quantity' cannot be empty or null.
EC904	Invalid parameter : 'quantity' must be a positive integer.
EC905	Invalid parameter : 'retention' cannot be empty or null.
EC906	Invalid parameter : 'product' cannot be empty or null.
EC907	Invalid parameter : 'transactionType' cannot be empty or null.
EC908	Invalid parameter : 'token' cannot be empty or null.
EC909	Invalid parameter : 'disclosedQty' cannot be empty or null.
EC910	Invalid parameter : 'price' cannot be empty or null.
EC911	Invalid parameter : 'triggerPrice' cannot be empty or null.
EC912	Error : Error while placing order.
EC913	Error : Error while fetching user details.
EC914	Invalid parameter : 'Request parameter' cannot be empty or null.
EC915	Error : Error while fetching order book.
EC916	Error : No order's found for this user.
EC917	Error : Error while fetching order history.
EC918	Error : No order history found for given order id.
EC919	Error : Error while fetching position book.
EC920	Error : No Positions found for this user.
EC921	Error : Error while fetching holdings.
EC922	Error : No holdings found for this user.
EC923	Error : Error while fetching profile details.
EC924	Error : Error while fetching RMS limits.
EC925	Invalid parameter : 'nestOrderNo' cannot be empty or null.
EC926	Error : No Trade's are found for this user.
EC927	Error : Error while fetching trade book.
EC928	Invalid parameter : 'product' Accepts only {'NORMAL','INTRADAY','DELIVERY','BNPL'}.
EC929	Invalid parameter : 'transactionType' Accepts only { 'BUY', 'SELL' }.
EC930	Invalid parameter : 'orderType' Accepts only { 'LIMIT', 'MARKET', 'SL', 'SLM'}.
EC931	Invalid parameter : 'orderType' Accepts only { 'REGULAR', 'BO', 'CO', 'MTF', 'AMO', 'BRACKETORDER', 'COVERORDER','MUTUALFUND', 'AFTERMARKETORDER' }.
EC932	Invalid parameter : 'validity' Accepts only { 'DAY', 'IOC' }.

EC933	Invalid parameter : 'priceType' cannot be empty or null.
EC934	Invalid parameter : 'orderType' cannot be empty or null.
EC935	Error : Error while fetching the single order margin.
EC936	Invalid parameter : 'product' cannot be empty or null.
EC937	Error : Error while canceling all order's.
EC938	Error : Error No Open order's to cancel the order's from orderbook.
EC939	Error : Error while fetching the span margin.
EC940	Error : Error while logging out.
EC941	Invalid parameter : 'instrumentId' cannot be empty or null.
EC942	Invalid parameter : 'orderComplexity' cannot be empty or null.
EC943	Invalid parameter : 'orderComplexity' Accepts only { 'REGULAR', 'AMO', 'BO' , 'CO' }.
EC944	Invalid parameter : 'validity' cannot be empty or null.
EC945	Invalid parameter : 'brokerOrderId' cannot be empty or null.
EC946	Error : Invalid : 'instrumentId' must contain only numeric characters.
EC947	Error : 'instrumentId' does not exist.
EC948	Error : 'quantity' should not be more than 50000000.
EC949	Error : 'quantity' must be a positive integer.
EC950	Invalid parameter: 'price' is required and cannot be empty or null.
EC951	Invalid parameter: 'slTriggerPrice' is required and cannot be empty or null.
EC952	Invalid parameter: 'slLegPrice' cannot be null or empty for BO/CO order .
EC953	Invalid parameter: 'targetLegPrice ' is required and cannot be empty or null.
EC954	Error : 'quantity' Should be multiple of lotsize.
EC955	Error : 'product' only Intraday is allowed for BO/CO orders.
EC956	Error : Orders for DELIVERY and BNPL are only allowed on NSEEQ and BSEEQ exchanges.
EC957	Error : Invalid 'price'.
EC958	Error : 'price' cannot be zero or negative.
EC959	Error : Invalid 'slTriggerPrice'.
EC960	Error : 'slTriggerPrice' cannot be zero or negative.
EC961	Error : Invalid 'slLegPrice'.
EC962	Error : 'slLegPrice' cannot be zero or negative.
EC963	Error : Invalid 'targetLegPrice '.
EC964	Error : 'targetLegPrice ' cannot be zero or negative.
EC965	Error : For BO orders, only LIMIT or SL 'orderType' is allowed.
EC966	Error : 'trailingSlAmount' cannot be empty or null for OrderType SL.
EC967	Error : 'trailingSlAmount' must be a positive number.

EC968	Error : 'trailingSlAmount' cannot be zero or negative.
EC969	Error : 'Product' only 'NORMAL' and 'INTRADAY' are allowed
EC970	Error : 'disclosedQuantity' allowed only for 'NSEEQ','BSEEQ','MCXCOMM','NSECURR','BSECURR'
EC970	Error : As compared to front end 'disclosedQuantity' should not be consider for F&O
EC971	Error : 'orderTag' characters should be less than or equal to 50 characters
EC972	Error : 'algoId' characters should be less than or equal to 12 characters
EC973	Error : Buy order 'slTriggerPrice' should be less than 'price'
EC974	Error : Sell order 'slTriggerPrice' should be greater than 'price'
EC975	Error : 'disclosedQuantity' cannot exceed the total order 'quantity'.
EC976	Error : 'slLegPrice' cannot be zero or null.
EC977	Error : 'slTriggerPrice' should be used only in 'SL' or 'SLM' OrderType
EC978	Error : 'price' should be used only in 'LIMIT' or 'SL' OrderType
EC979	Invalid 'brokerOrderId'.
EC980	Invalid 'instrumentId'.
EC981	Error : Invalid 'disclosedQty'.
EC982	Error : For 'AMO' 'disclosedQuantity' should be zero.
EC983	Invalid 'algoId'
EC984	Invalid 'orderTag'
EC985	Invalid 'brokerOrderId'.
EC986	Invalid parameter : SpanMargin not allowed for 'NSEEQ' and 'BSEEQ'
EC987	Invalid parameter : 'exchange' Accepts only {'NSEFO','BSEFO','NSECURR','BSECURR','MCXCOMM','NCDEXCOMM','NSECOMM','BSECOMM'}
EC988	Error : 'marketProtectionPercent ' must be a positive number.
EC989	Error : For CO orders, only LIMIT or MARKET 'orderType' is allowed.
EC990	Error : 'quantity' Should be multiple of lotsize.
EC991	Error : 'disclosedQuantity' Should be multiple of lotsize.
EC992	Error : Not able modify the given order. 'brokerOrderId' is
EC993	Error : Provided 'brokerOrderId' not in valid state to modify the order. 'brokerOrderId' is
EC994	Error : Given 'brokerOrderId' is not in your ordebook.
EC995	Error : There is no order to cancel.
EC996	Error : 'validity' IOC is not allowed for AMO order.
EC997	Error : The specified order is not available in the order book and cannot be canceled. Please verify the order details and try again.
EC998	Error : The specified order is not available in the order book, and order history cannot be retrieved. Please verify the order ID and try again.

EC999	Error : The specified order is not available in the order book and cannot be modified. Please verify the order details and try again.
EC701	Error : Orders with Exchange 'BSEEQ/BSEFO/BSECURR', Cannot be modified to Order Type 'SL' (Stop Loss).
EC702	Error : Only price field can be modified in Sub leg Order's.
EC703	Error : Your account is in a dormant state and trading is not permitted. Please contact support to reactivate your account.
EC704	Error : Your account is in a frozen state. Trading and related activities are currently restricted. Please contact customer support to resolve this issue.
EC705	Error : Your account is in a voluntary freeze state. Trading is restricted as per your request. Please contact customer support if you wish to lift this restriction.
EC900	Invalid parameter: 'exchange' cannot be empty or null.
EC901	Invalid parameter: 'exchange' Accepts only {'NSEEQ','NSEFO','BSEEQ','BSEFO','NSECURR','BSECURR','MCXCOMM','NCDEXCOMM','NSECOMM','BSECOMM'}
EC900	Invalid parameter: 'instrumentId' cannot be empty or null.
EC801	Invalid parameter: 'interval' cannot be empty or null.
EC802	Invalid parameter: 'interval' Accepts only {“1 minute”, “5 minutes”, “10 minutes”, “15 minutes”, “30 minutes”, “60 minutes”, “1 day”}
EC803	Invalid parameter: 'fromDate' cannot be empty or null.
EC804	Invalid fromDate format. Please enter the fromDate in DD-MMM-YYYY format (e.g., 20-Sep-2020).
EC805	Invalid parameter: 'toDate' cannot be empty or null.
EC806	Invalid toDate format. Please enter the toDate in DD-MMM-YYYY format (e.g., 20-Sep-2020).
EC807	Invalid date range. The 'fromDate' cannot be greater than the 'toDate'.
EC808	Error : Invalid : 'instrumentId' must be greater than 0.
EC809	Maximum permissible date range limit reached
EC810	From date has to be less than end date
EC001	Invalid parameter : 'UserIdentity' cannot be empty or null.
EC002	Client does not exist.
EC004	User blocked.
EC005	Invalid parameter : Request is null or empty.
EC006	Invalid parameter : 'IsPANEntered' cannot be empty or null.
EC007	Invalid parameter : 'IsPANEntered', IsPANEntered will consists only Y and N.
EC008	Invalid parameter : 'UserId' cannot be empty or null.
EC009	Invalid parameter : 'Password' cannot be empty or null.
EC010	Invalid parameter : 'appName' cannot be empty or null.
EC011	Multiple client IDs associated with the provided details. Please specify a client code to proceed further.

EC012	Multiple client IDs associated with the provided details. Please specify a client code to proceed further.
EC013	Multiple client IDs associated with the provided details. Please specify a client code to proceed further.
EC014	Your account has been blocked by adminstrator. Please contact admin(Unblock accout option wont work).
EC015	Not able to send OTP for the registered mobile and email.
EC016	Invalid userId or password.
EC017	Invalid parameter : 'osName' cannot be empty or null.
EC018	Your OTP has been initiated. Please try again after 60 seconds.
EC019	You have exceeded the maximum limit for OTP requests. Please try again later.
EC020	We are currently unable to send OTP. Please contact the administrator for assistance.
EC021	Invalid parameter : 'ClientCode' cannot be empty or null.
EC022	Invalid parameter : 'ReceivedOTP' cannot be empty or null.
EC023	Invalid parameter : 'IsPANEntered' cannot be empty or null.
EC024	Your OTP is expired.
EC025	Invalid OTP.
EC026	Invalid parameter : 'userId' cannot be empty or null.
EC027	Invalid parameter : 'pan' cannot be empty or null.
EC028	Invalid parameter : 'osName' cannot be empty or null.
EC029	Invalid parameter : 'dob' cannot be empty or null.
EC030	Invalid user.
EC031	The PAN provided is invalid. Please check and enter a valid PAN.
EC032	The PAN provided is invalid. Please check and enter a valid PAN.
EC033	Invalid parameter : 'otp' cannot be empty or null.
EC034	Invalid parameter : 'totp' cannot be empty or null.
EC035	The T-OTP is already enabled for this account.
EC036	Invalid T-OTP.
EC037	Only MOB keys are accepted for this operation.
EC038	Invalid parameter : 'token' cannot be empty or null.
EC039	Invalid parameter : 'deviceId' cannot be empty or null.
EC040	Invalid parameter : 'deviceType' cannot be empty or null.
EC041	Invalid parameter : 'enable' cannot be empty or null.
EC042	Error :
EC043	T-OTP Not enabled for this account
EC044	Error : Token is not valid.
EC045	Token is valid.
EC046	Invalid parameter : 'osName' cannot be empty or null.

EC047	Invalid parameter : 'versionNo' cannot be empty or null.
EC048	Invalid parameter : 'deviceId' cannot be empty or null.
EC049	Invalid parameter : 'deviceName' cannot be empty or null.
EC050	Invalid parameter : 'macAddress' cannot be empty or null.
EC051	Error : Only mobile OS are allowed to call this method.
EC052	Error : You have reached the MAX OTP limit, Kindly try again after 15 minutes.
EC053	Invalid parameter : 'appKey' cannot be empty or null.
EC054	Error : Given appkey is InCorrect, Please enter the valid appkey
EC055	Error : Unauthorized access: Please authorize the application to obtain the auth code.
EC056	Invalid parameter : 'ceData' cannot be empty or null.
EC057	Invalid parameter : 'cEncData' cannot be empty or null.
EC058	Error : Invalid 'deviceId'.
EC059	Error : Invalid 'versionNo'.
EC060	Error : Invalid 'appName'.
EC061	Error : Invalid 'osName'.
EC062	Error : Illegal or Expired Encryption key.
EC063	Error : 'password' should not contain spaces.
EC064	Multiple client IDs associated with the provided email. Please specify a client code to proceed further.
EC065	Multiple client IDs associated with the provided mobile. Please specify a client code to proceed further.
EC066	Multiple client IDs associated with the provided details. Please specify a client code to proceed further.
EC067	Error : Provide pan is not mapped with any client Id.
EC068	Error : Provide userId is not mapped with given pan.
EC069	Error : Your password has expired. Please change your password to continue.
EC070	Invalid parameter : 'version' cannot be empty or null.
EC071	Invalid parameter : 'os' cannot be empty or null.
EC072	Mobile Number and email ID are invalid, please write to cs@iifl.com to update your Mobile Number & email ID.
EC073	Error: Please enter the date of birth (DOB) in MMDDYYYY format.
EC074	Error: Invalid 'osName'.
EC075	Error: Unauthorized access.
EC076	Error: This service is not available for your account type.
EC077	Error : Provide PAN is not mapped with given user Id.
EC078	Error : Provide PAN is not mapped with given user Id.
EC079	Error: Password must have one allowed special character out of @ # $ % & * / \\
EC080	Error : Too many incorrect OTP attempts. Request a new OTP after 15 minutes.

EC081	Error : Your account is in a voluntary freeze state, and login is not permitted. Please contact the administrator for assistance.

Python SDK errors and packets

Below are the status codes sent by the Python SDK and their corresponding messages:

Status Codes	Message
101	Request cannot be null
102	TopicList cannot be null and no. of topics should be less than 1024
103	Subscripton failed
105	Client is already connected
-1	Generic error
1	Tokin in Invalid
0	Success, no error
4	The client is not currently connected

Below are the result codes sent by the Python SDK and their corresponding messages:

result Codes	Message
0	Topic granted
104	Invalid topic
128	Topic not granted

Below are the Acknowledgement packetTypes sent by the Python SDK and their corresponding names:

packetType	PacketName
2	Connection acknowledgement
9	Subscription acknowledgement
11	Unsubscription acknowledgement
14	Disconnect acknowledgement

RMS Order Rejections

All the RMS rejections starts with a string RMS: ,
if any rejection message doesn't contain RMS: , that means its not rejected because of RMS.
RMS rejection reasons are generated dynamically, based on "Block Details" or "Rules" or/and "order validation level".

List of Possible rejections from RMS are as below:

- "RMS:Blocked for " -> Block details will be appended to this.
- "RMS:Rule: " -> Rule name and order validation level will be appended to this.

- "RMS:Margin Exceeds, Required:<Value>, Available:<Value>" , order validation level will be appended to this.
- "RMS:MtoM Exceeds, Required:<Value>, Available:<Value>", order validation level will be appended to this.
- "RMS:Lt rate not found for rule: "

Rule name and order validation level will be appended to this:

- "RMS:Asset exchange segment not found for rule: "
- "RMS:Field Not Found <MNM_ID>"
- "RMS:Bad Input"
- "RMS:Scrip Not found in Mrv master"
- "RMS:Mrv Master DOWN"
- "RMS:User not enabled on product"
- "RMS:Client not enabled on product"
- "RMS: NO Last Trade Price"
- "RMS: Auto Square Off Block"
- "RMS:Scrip <Symbol> is in Ban Period."
- “RMS:Index value not found”
- “RMS:Index close value not found”
- “RMS:Entity is not loaded properly”

Rate Limits

1. Order per second rate limits to be put **for each session** is given below:

Session level rate limiting(OPS)			
	API	**Non-Registered (<10 OPS)**	**Registered (>10 OPS)**
User	Logout	2	2
	Get User Session	3	3
	Profile	3	3
	Limits	10	20
Margin	Pre order margin	10	20
	SPAN exposure	10	20
Order Management	Place Order	10(*Combined rate limit to be set as 10 OPS)	20
	Modify Order		20
	Cancel Order		20
	Cancel All Orders	3	3
	Get Order Book	3	3

	Get Order History	10	10
	Trade Book	3	3
Portfolio	Positions	3	3
	Holdings	3	3
Market Data API	Historical Data	3	3
	Market Depth	10	10
	Open Interest	10	20
	Market Quotes	10	10
Contract Master APIs	NSEEQ	2	2
	NSEFO	2	2
	NSECOMM	2	2
	MCXCOMM	2	2
	INDICES	2	2
	NSECURR	2	2
	BSEEQ	2	2
	BSEFO	2	2
	BSECURR	2	2

FAQs

- What are IIFL Markets Trading APIs?

IIFL Markets Trading APIs allow individual traders and fintech platforms to connect directly to IIFL's broker systems for order placements, market data retrieval, and portfolio management.

- Is there any cost to use the IIFL Markets APIs?

No, the APIs are free for individual traders, developers, and platforms to integrate with.

- How do I get access to the APIs?

IIFL Capital clients can log in to the developer portal using their Trading Account credentials, create a new application, and start using the APIs right away.
Fintech platforms can register on the portal, after which the IIFL team will review and approve the application for API access.

- How does client login work for a trading application?

Clients can follow the login steps provided in the API Documentation, with a video tutorial also available for reference.

- Do I need to log in daily, and will a new session token be generated each time?

Yes, you need to log in every day. A new, unique session token is generated with each login.

- How does authentication work with the API?

After a successful login, a session token is generated and must be included in the header of each API request to authenticate the user.

- What do the two status and message parameters in API responses mean?

The two status and message parameters are explained in this section.

- Can I convert MIS (intraday) trades to CNC (delivery) or vice versa using the API?

No, order conversions between MIS and CNC/NRML are not supported through the API. These conversions can be done via the IIFL Markets Mobile Application.

- Where can I access real-time market data, and can I use third-party data sources?

Market data is available through the APIs and bridge package. You can also use authorized third-party vendors for market data.

- Is the use of the bridge package mandatory for accessing the Market Data stream?

Yes, the bridge package is mandatory for accessing the Market Data stream. It contains pre-built functions and sample implementations, simplifying the integration process.

- In what all languages the Bridge package is available in?

Find the list of Bridge Packages available in multiple programming languages here. If your preferred programming language package is not available, you can request it through your RM/POC.

- How can I access the full list of tradable instruments?

You can access the instrument master list in both CSV and JSON formats here. This list provides comprehensive, segment-wise details on all available tradable instruments.