Amazon Honeycode

User Guide

Amazon Honeycode: User Guide

Table of Contents

What Is Amazon Honeycode?

Amazon Honeycode is a fully managed service that allows you to quickly build mobile and web apps for teams—without programming. Build Amazon Honeycode apps for managing almost anything, like projects, customers, operations, approvals, resources, and even your team.

To learn more about Amazon Honeycode, visit Getting Started with Honeycode

Connecting Amazon Honeycode to AWS

The AWS Management Console is where you can connect Amazon Honeycode to AWS. The console enables you to connect a Honeycode team to an AWS account and upgrade or downgrade your team plans.

If you do not have an AWS account, start by creating one following instructions here . Once you have an AWS account, you can connect it to Honeycode by following the instructions here.

Accessing Developer Tools and APIs

Upgrading to a Plus or Pro plan is no longer required as a prerequisite to accessing developer tools and Honeycode APIs.

AWS Software Development Kit (SDK)

The AWS Software Development Kit (SDK) is a software library that enables you to interact with AWS Application Programming Interfaces (APIs). The AWS SDK supports Amazon Honeycode APIs. To use the SDK you will need to connect your Amazon Honeycode team to an AWS account.

Identity and Access Management (IAM)

AWS Identity and Access Management (IAM) is a service that helps you securely control access to AWS resources. You can use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

Authentication

Amazon Honeycode APIs use SigV4 to authenticate callers. The endpoints reject any request that doesn't have the authorization header in the HTTP request. If you use the AWS SDK to send your requests, the SDK clients authenticate your requests by using access keys that you provide.

Authenticating Requests on AWS documentation describes step-by-step how to calculate the signature and include it in the request.

Authorization

Amazon Honeycode API requests are authorized using IAM policies. These IAM policies can specify the specific actions and resources callers are allowed to access.

You can use either of these managed policies:

- AmazonHoneycodeWorkbookFullAccess
- AmazonHoneycodeFullAccess

Or, you can create your own policies using *honeycode:<action-name>* actions. For more information about policies, see IAM Documentation.

To set up the managed, full-access policy:

- Create a role in your AWS account to give access to all your workbooks.
- Make sure you give the new role a meaningful name.
- Attach either of the managed policies (*AmazonHoneycodeWorkbookFullAccess* or *AmazonHoneycodeFullAccess*) to the role you just created.

- Assign the role to the IAM user who will be making the API calls.

Resource ARNs

Honeycode resource ARNs are of the format
`arn:aws:honeycode:AWS_Region:AWS_account_ID:Resource_Type:Resource_Path`

Resource Type

Resource_Type indicates the type of resource represented by the ARN. The following resource types are supported by Honeycode APIs.

- workbook
- table
- screen
- screen-automation

Resource Path

Resource_Path indicates the path to the resource represented by the ARN. Resource paths for workbook resources start with workbook/. Each resource type will have a different path to the resource. For example: `workbook/Workbook_ID` , `workbook/Workbook_ID/table/Table_ID` , `workbook/Workbook_ID/app/App_ID/screen/Screen_ID`

Note that writing an IAM policy to give a user access to a *workbook* resource doesn't automatically give the user access to all the resource types (tables, screens etc) in that workbook. You need to use wildcards with specific resource types that you want to grant access for, or use wildcards on all resource types to give broader access. For example the wildcard resource ARN below gives a user access to all tables in all workbooks but it doesn't give the user access to other resource types like workbook, screen or screen-automation. `arn:aws:honeycode:AWS_Region:AWS_account_ID:table:workbook/*` On the flip side, using the wildcard resource ARN below in a DENY policy prevents the user from accessing any table in any workbook but it doesn't prevent them from accessing other resource types like workbook, screen or screen-automation.

If you wish to allow or deny access to all resources in a single workbook, you need to use a wildcard resource ARN that specifies * in both the *Resource_Type* and *Resource_Path*. For example:

`arn:aws:honeycode:AWS_Region:AWS_account_ID:*:workbook/Workbook_ID*`

Workbook ARN format

The ARN format for a workbook is

`arn:aws:honeycode:AWS_Region:AWS_account_ID:workbook:workbook/Workbook_ID`

You can also use wildcards to specify all workbooks.

`arn:aws:honeycode:AWS_Region:AWS_account_ID:workbook:workbook/*`

Table ARN format

The ARN format for a table is

`arn:aws:honeycode:AWS_Region:AWS_account_ID:table:workbook/Workbook_ID/table/Table_ID`

You can also use wildcards to specify parts of the ARN and include multiple resources underneath. To specify all tables in a workbook, you can use the following resource ARN:

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:table:workbook/Workbook_ID/table/*
```

To specify all tables in all workbooks, you can use the following resource ARN:

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:table:workbook/*
```

Screen ARN format

The ARN format for a screen is

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen:workbook/WorkbookID/app/App
ID/screen/ScreenID
```

You can also use wildcards to specify parts of the ARN and include multiple resources underneath.

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen:workbook/WorkbookID/
app/AppID/screen/*
```

To specify all screens in a single app you can use the following resource ARN:

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen:workbook/WorkbookID/app/*
```

Screen Automation ARN format

The ARN format for a screen automation is

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen-
automation:workbook/WorkbookID/app/AppID/screen/ScreenID/
automation/AutomationID
```

You can also use wildcards to specify parts of the ARN and include multiple resources underneath.

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen-
automation:workbook/WorkbookID/app/AppID/screen/ScreenID/automation/*
```

Similarly, to specify all automations in a single app you can use the following resource ARN in your IAM policy:

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen-
automation:workbook/WorkbookID/app/AppID/screen/*
```

To specify all automations in all apps in a workbook, use the following resource ARN:

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen-
automation:workbook/WorkbookID/app/*
```

Finally, to specify all automations in all workbooks in your account, you can either use the resource ARN "*" or an ARN in the following format:

```
arn:aws:honeycode:AWS_Region:AWS_account_ID:screen-automation:workbook/*
```

Authorizing Team Connections

Honeycode provides actions to view, approve, and reject teams associated with your account in AWS Console. These actions are not available in AWS SDK.

You can use either of these managed policies:

- *AmazonHoneycodeTeamAssociationFullAccess*
- *AmazonHoneycodeFullAccess*

Or you can create your own policies using *honeycode:<action-name>* actions. Read more about access management permissions and policies.

ListTeamAssociations

This action lists all pending and approved team connection requests for your AWS account.

ApproveTeamAssociation

This action allows you to approve a pending team connection request and connect the team to your AWS account. When connected, the AWS account will be used to bill any usage on your Amazon Honeycode team. Refer to Amazon Honeycode pricing

RejectTeamAssociation

This action allows you to reject a pending team connection request.

Interacting with Honeycode workbooks via SDK

Amazon Honeycode has many APIs that let you programmatically interact with Honeycode workbooks. You can use these APIs to read, write, update or delete data stored in Honeycode workbooks.

Honeycode APIs can be broadly divided into four categories. Each category has several APIs underneath them.

App Screen APIs

These APIs allow you to read, write, update or delete data stored in Honeycode workbooks as you would interacting with Honeycode apps. As with any Honeycode app, you can control exactly the data from your workbook that you want to expose to the APIs. The APIs are:

- GetScreenData
- InvokeScreenAutomation

Table Metadata APIs

These APIs allow you to retrieve metadata about tables in Honeycode workbooks. The APIs are:

- ListTables
- ListTableColumns

Table Row Operation APIs

These APIs allow you to read, append, update or delete data stored in Honeycode table rows. The APIs are:

- ListTableRows
- QueryTableRows
- BatchCreateTableRows
- BatchUpdateTableRows
- BatchUpsertTableRows
- BatchDeleteTableRows

Import APIs

These APIs allow you to import data into tables in Honeycode workbooks and check on the status of previously submitted import requests. The APIs are:

- StartTableDataImportJob
- DescribeTableDataImportJob

App Screen APIs

App Screen APIs allow you to read, write, update or delete data stored in Honeycode workbooks as you would interacting with Honeycode apps. As with any Honeycode app, you can control exactly the data from your workbook that you want to expose to the APIs. The APIs are:

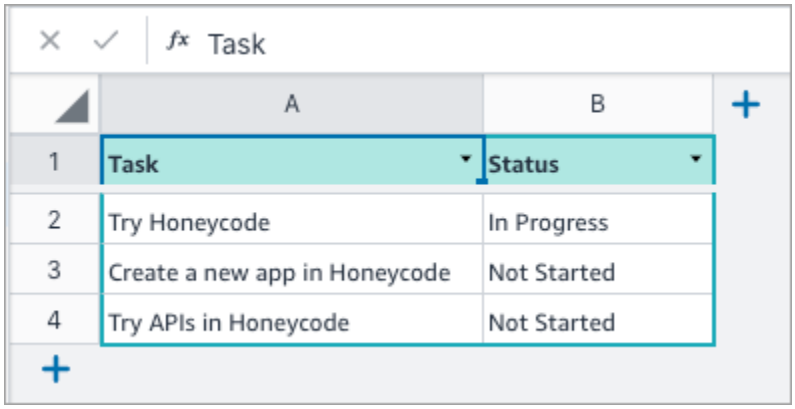
- GetScreenData
- InvokeScreenAutomation

Setting up for App Screen APIs

To use App Screen APIs you'll need to first create a table and add some data. Then build an app and add a screen specific to each API action or automation. Although you can use any app for this purpose, we recommend you create an app solely for API access.

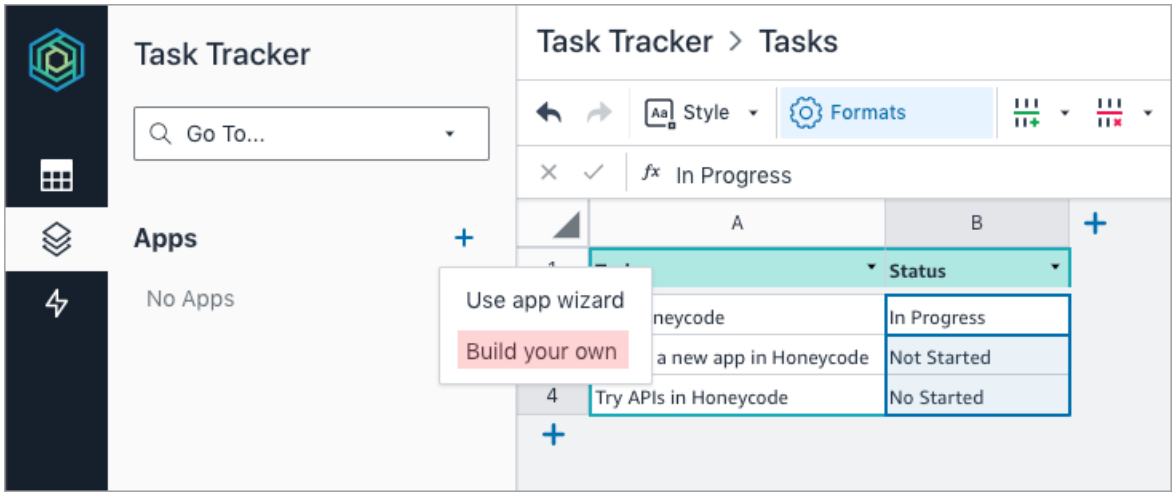
Create table

You will need a workbook with tables to set up APIs. If you haven't already done so, create a new workbook and add a new table either by importing a CSV file or by adding data manually. In the example below, we start with a simple tasks table.



Start building your app

Open your workbook from Honeycode Builder, and from the left navigation, click the + and select "Build your own". Formatting, font, and colors aren't important for this app, so you can leave the app styling as default, if you wish.



Name your app objects

Honeycode APIs return the names of blocks and data cells, along with the values of data cells. Be thoughtful as you name your objects as they are visible in the API response.

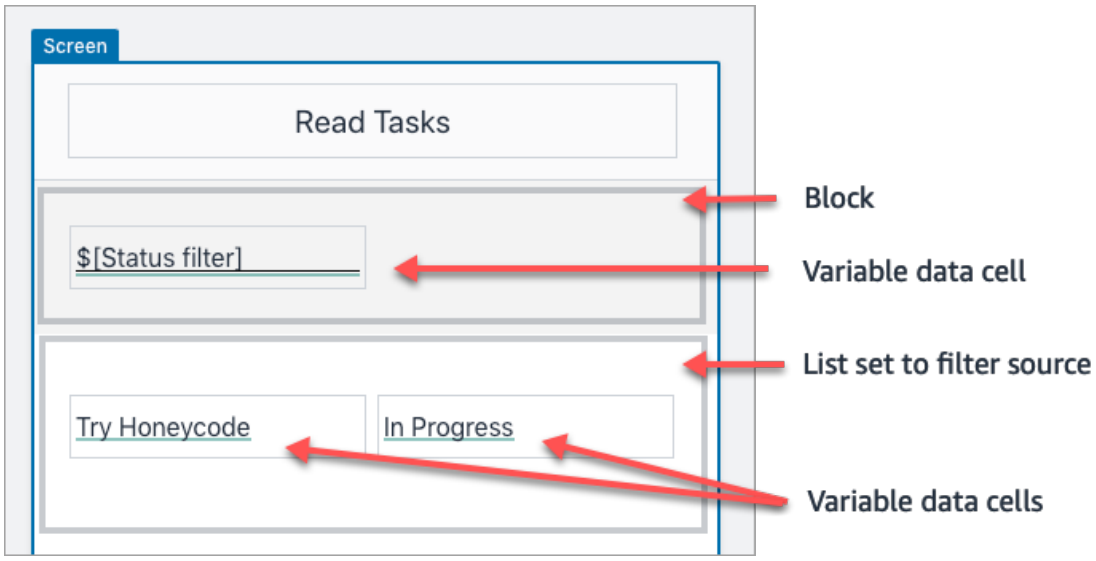
Configuring app screens

There are many ways you can configure your app, but here we'll show you how to set up screens for some common actions that you can use APIs to execute.

For the examples below, we used the managed policy, *AmazonHoneycodeWorkbookFullAccess*.

Read data

Based on the Tasks table above, let's say you want to read items filtered by the Status column. Here's an example of one way to set up a screen that allows you to use the *GetScreenData* API to read screen data programmatically.



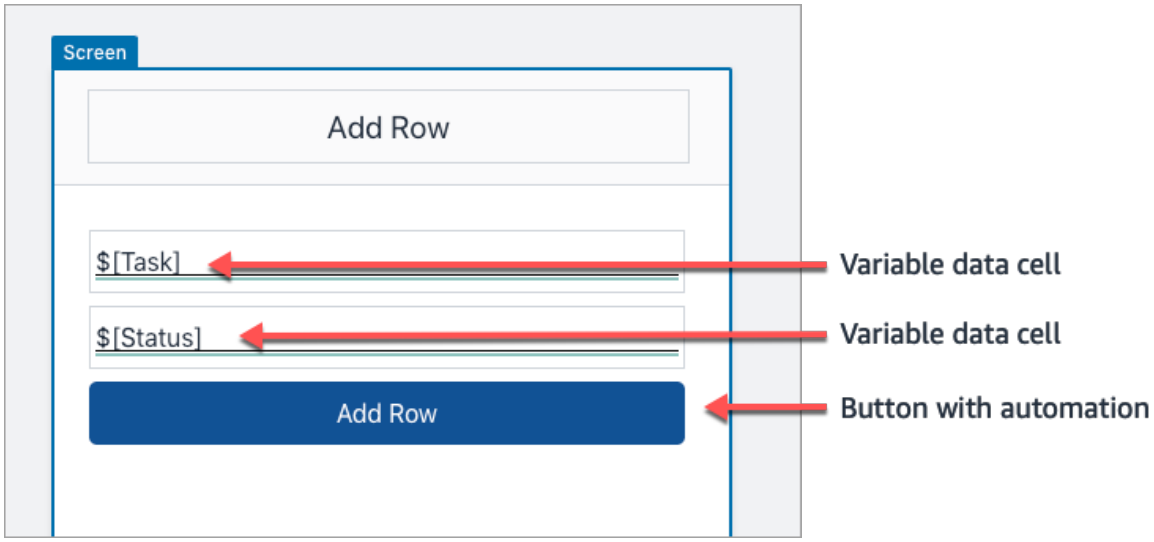
- Insert a Column list object, which can include a preconfigured filter
- Choose to filter by the Status column *=$[Status filter]*

- Make the *=$[Status filter]* data cell editable, if it isn't already
- Check that the list source is set to *=FILTER(Tasks,"(Tasks[Status]=$[Status filter] OR $[Status filter]="""")")*
- Confirm that both data cells in the list shared, and set the sources to the Tasks column and the Status column

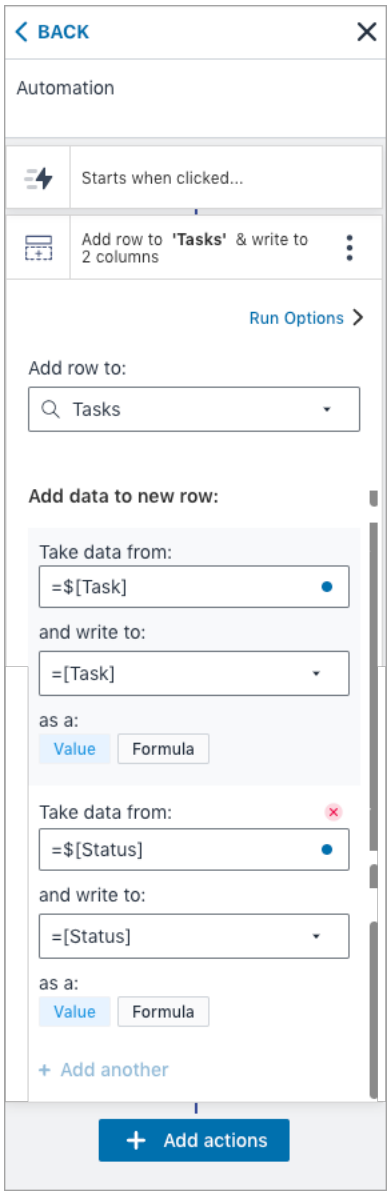
> **Note**
> The Column list is a preconfigured app object that includes some UI features not shown in the image above.

Add a row

If you want to manipulate data using the *InvokeScreenAutomation* API to add rows, then set up the screen and corresponding button in your app.

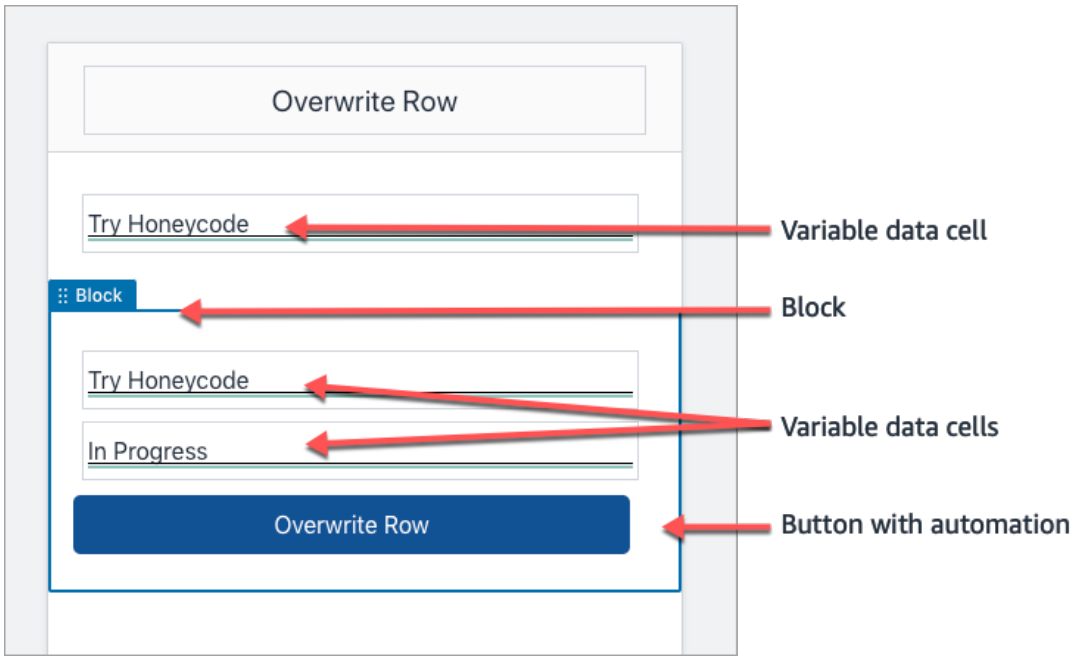


- Add variable data cells and name them Task and Status
- Make the data cells editable
- To add the automation, select the button and click on the Actions tab in the button properties panel
- Configure your automation to take data from the variable data cell *=$[Tasks]* and write it to the Tasks column
- Repeat the automation for the variable data cell *=$[Status]*

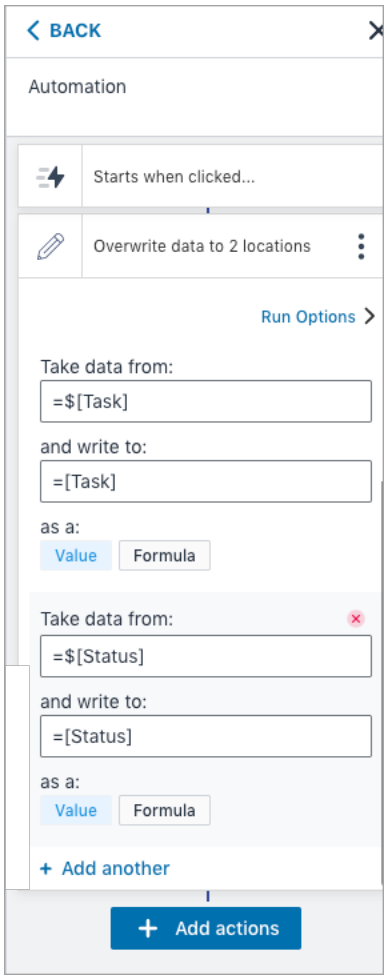


Overwrite a row

If you want to manipulate data using the *InvokeScreenAutomation* API to overwrite existing rows, then set up the screen and corresponding button in your app.

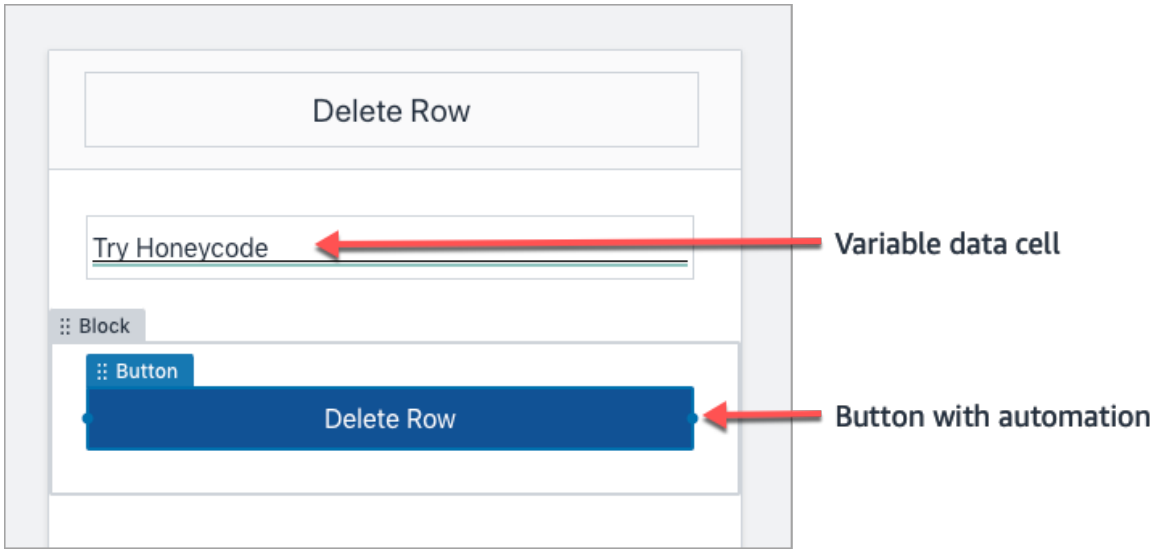


- Add a variable data cell, make it editable, and name it Row
- Under the Display tab, format the data cell as a rowlink and set the source as the Tasks table
- Add a block and set the data source as the variable data cell *=$[Row]*
- Add a variable data cell, name it Task, make it editable, and set the data source to the Task column *=[Task]*
- Repeat the above step, naming it Status and setting the data source to to the Status column *=[Status]*
- To add the automation, select the button and click on the Actions tab in the button properties panel
- Configure your automation to take data from the variable data cell *=$[Tasks]* and write it to the Tasks column
- Repeat the automation for the variable data cell *=$[Status]*



Delete a row

If you want to manipulate data using the *InvokeScreenAutomation* API to delete existing rows, then set up the screen and corresponding button in your app.



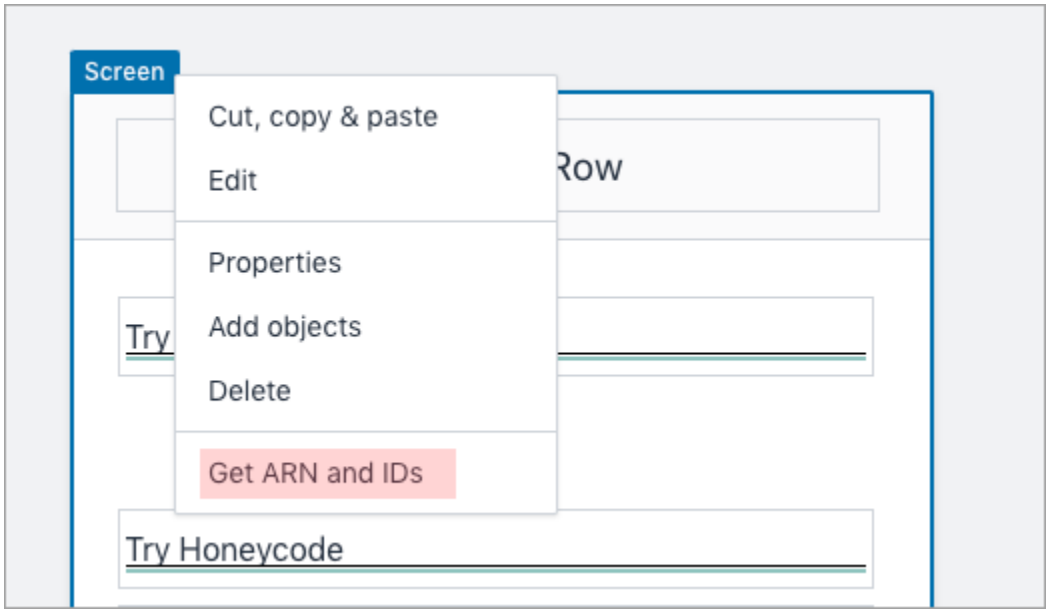
- Add a variable data cell, make it editable, and name it Row
- Under the Display tab, format the data cell as a rowlink and set the data source as the Tasks table
- To add the automation, select the button and click on the Actions tab in the button properties panel
- Configure your automation to delete the context row (aka the triggering row) or a specified row

ARNs and Honeycode IDs

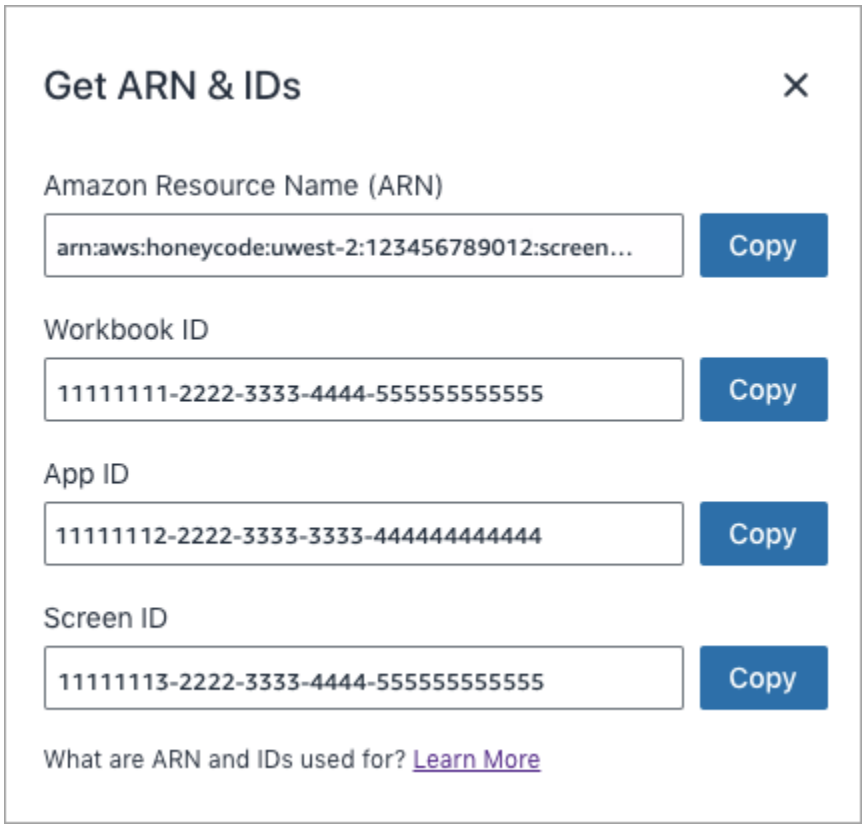
Amazon Resource Names (ARNs) uniquely identify AWS resources and are required for many actions related to IAM policies and API calls.

Accessing ARN and IDs

In any app screen, you can right-click to bring up a menu. Select Get ARN and IDs



From the modal that appears, you can copy the screen ARN and IDs for the workbook, app, and screen.



Similarly, you can right-click on any component that has an automation defined on it and select Get ARN and IDs to get the screen automation ARN and IDs for the workbook, app, screen and automation.

ARNs and authorization

If you are using the managed IAM policies *AmazonHoneycodeWorkbookFullAccess* or *AmazonHoneycodeFullAccess*, you will not need the ARN for authorization. The ARN is only required to to set up authorization using IAM at a granular level.

The Honeycode resources that need to be defined in custom IAM policies for App Screen APIs are given below. Click the links to get more details on how to construct ARNs for those resources.

- *GetScreenData*: screen
- *InvokeScreenAutomation*: screen-automation

GetScreenData

The GetScreenData API allows you to retrieve data from a screen in a Honeycode app. The API allows you to set local variables in the screen to filter, sort or otherwise affect what you display on the screen.

To find more details about using this API check the GetScreenData API Reference page.

InvokeScreenAutomation

The InvokeScreenAutomation API allows you to invoke an action defined in a screen in a Honeycode app. The API allows you to set local variables, which you can then use in the automation you're invoking. This allows you to automate Honeycode app interactions to write, update or delete data in your workbook.

To find more details about using this API check the InvokeScreenAutomation API Reference page.

Sample API calls

At this stage, you're all set to use App Screen APIs using two supported SDKs. This section demonstrates making simple API calls using the *AWS CLI* and *Python SDK*.

Sample API calls using AWS CLI

If you've already set up AWS CLI, this might be the easiest way to verify that everything is working. If you haven't already set up the CLI, follow the guide here.

In the examples below, we assume you're operating on the workbook as defined in Setting up for App Screen APIs

Making a basic query

An example of a basic query to read a list of tasks using the *GetScreenData* API.

Request:

```
aws honeycode get-screen-data \
  --profile <my-auth-profile> \
  --workbook-id <workbook-id> \
  --app-id <app-id> \
  --screen-id <screen-id> \
  --max-results 2
```

It loads the first two tasks from the Tasks table. Since there are more than two rows that match the list formula, *nextToken* is included in the response to continue loading data in subsequent calls.

Response:

```
{
    "nextToken": "<Next Token Value>",
    "results": {
        "Tasks List": {
            "headers": [
                {
                    "format": "Text",
                    "name": "Task"
                },
                {
                    "format": "Text",
                    "name": "Status"
                }
            ],
            "rows": [
                {
                    "dataItems": [
                        {
                            "formattedValue": "Try Honeycode",
                            "rawValue": "Try Honeycode"
                        },
                        {
                            "formattedValue": "In Progress",
                            "rawValue": "In Progress"
                        }
                    ],
                    "tableRowId": "<A Row ID is here>"
                },
                {
```

```
                    "dataItems": [
                        {
                            "formattedValue": "Create a new app in Honeycode",
                            "rawValue": "Create a new app in Honeycode"
                        },
                        {
                            "formattedValue": "Not Started",
                            "rawValue": "Not Started"
                        }
                    ],
                    "tableRowId": "<A Row ID is here>"
                }
            ]
        }
    },
    "workbookCursor": <A number is here>
}
```

Using next token

This *GetScreenData* example takes the pagination token from the previous response and includes it in the input JSON file for the next call. *viewArn* and *maxResults* are the same as in previous request.

Note that *nextToken* is different for different calls, even with the same set of parameters. The token expires after one hour, so if you paste the request below, you'll get an error response that the token is expired. You'll need to run the previous request and copy the token from the output into this request.

Request:

```
aws honeycode get-screen-data \
--profile <my-auth-profile> \
--workbook-id <workbook-id> \
--app-id <app-id> \
--screen-id <screen-id> \
--max-results 2 \
--next-token [Next Token Value]
```

Response:

Includes data for the next two tasks. If there are more results matching the query, a new pagination token is included in the response.

```
{
    "nextToken": "<Next Token Value>",
    "results": {
        "Tasks List": {
            "headers": [
                {
                    "format": "Text",
                    "name": "Task"
                },
                {
                    "format": "Text",
                    "name": "Status"
                }
            ],
            "rows": [
                {
                    "dataItems": [
                        {
                            "formattedValue": "Try API in Honeycode",
                            "rawValue": "Try API in Honeycode"
                        },
```

```
                        {
                            "formattedValue": "Not Started",
                            "rawValue": "Not Started"
                        }
                    ],
                    "tableRowId": "<A Row ID is here>"
                },
                {
                    "dataItems": [
                        {
                            "formattedValue": "Read a book",
                            "rawValue": "Read a book"
                        },
                        {
                            "formattedValue": "Not Started",
                            "rawValue": "Not Started"
                        }
                    ],
                    "tableRowId": "<A Row Id is here>"
                }
            ]
        }
    },
    "workbookCursor": <A number is here>
}
```

Passing named variables

This *GetScreenData* example loads the the list used in the previous example, filtered by the status “Not Started”.

Request:

```
aws honeycode get-screen-data \
--profile <my-auth-profile> \
--workbook-id <workbook-id> \
--app-id <app-id> \
--screen-id <screen-id> \
--max-results 2 \
--variables '{"Status Filter": { "rawValue": "Not Started"}}'
```

Response:

Includes data for the for all the tasks that match the specified status. If there are more results matching the query, a new pagination token is included in the response.

```
{
    "nextToken": "<Next Token Value>",
    "results": {
        "Tasks List": {
            "headers": [
                {
                    "format": "Text",
                    "name": "Task"
                },
                {
                    "format": "Text",
                    "name": "Status"
                }
            ],
            "rows": [
                {
                    "dataItems": [
```

```
                        {
                            "formattedValue": "Create a new app in Honeycode",
                            "rawValue": "Create a new app in Honeycode"
                        },
                        {
                            "formattedValue": "Not Started",
                            "rawValue": "Not Started"
                        }
                    ],
                    "tableRowId": "<A Row ID is here>"
                },
                {
                    "dataItems": [
                        {
                            "formattedValue": "Try API in Honeycode",
                            "rawValue": "Try API in Honeycode"
                        },
                        {
                            "formattedValue": "Not Started",
                            "rawValue": "Not Started"
                        }
                    ],
                    "tableRowId": "<A Row ID is here>"
                },
                {
                    "dataItems": [
                        {
                            "formattedValue": "Read a book",
                            "rawValue": "Read a book"
                        },
                        {
                            "formattedValue": "Not Started",
                            "rawValue": "Not Started"
                        }
                    ],
                    "tableRowId": "<A Row ID is here>"
                }
            ]
        }
    },
    "workbookCursor": <A number is here>
}
```

Adding a row

This *InvokeScreenAutomation* example adds a new row with the values “Test” and “Not Started” in the Tasks table.

The variables field of the *InvokeScreenAutomation* request is an optional field. It is used to set the value of the variable used in the automation. It is a map with the variable name as the map key and the value to set as the map value. In the example below, the variables used in the automation are *$[Task]* and *$[Status]*.

```
{
    "variables": {
        "Task": {
            "rawValue": "Test"
        },
        "Status": {
            "rawValue": "Not Started"
        }
    }
}
```

Request:

```
aws honeycode invoke-screen-automation \
--profile <my-auth-profile> \
--workbook-id <workbook-id> \
--app-id <app-id> \
--screen-id <screen-id> \
--screen-automation-id <screen-automation-id> \
--client-request-token <client-request-token> \
--variables '{"Task": {"rawValue": "Test"}, "Status":{"rawValue": "Not Started"}}'
```

Response:

A new task with the name “Test” is added to the Tasks table.

Overwriting a row

In this *InvokeScreenAutomation* example, the variables used in the automation are *$[Row]*, *$[Task]* and *$[Status]*. The table row ID for the variable *$[Row]* is located in the output of the *GetScreenData* calls.

```
{
    "Row": {
        "rawValue": "<Table row ID goes here>"
    },
    "Task": {
        "rawValue": "Test"
    },
    "Status": {
        "rawValue": "In Progress"
    }
}
```

Request:

```
aws honeycode invoke-screen-automation \
--profile <my-auth-profile> \
--workbook-id <workbook-id> \
--app-id <app-id> \
--screen-id <screen-id> \
--screen-automation-id <screen-automation-id> \
--client-request-token <client-request-token> \
--variables '{"Row": {"rawValue": "<Table row ID goes here>"}, "Task": {"rawValue":
 "Test"}, "Status":{"rawValue": "In Progress"}}'
```

Response:

The status of the “Test” task is updated.

Deleting a row

In this example, the variable used in the automation is *$[Row]*. The table row ID for the variable *$[Row]* is be located in the output of the *GetScreenData calls*. A sample call might look like:

```
{
    "variables": {
        "Row": {
            "rawValue": "<Table row ID goes here>"
        }
    }
}
```

Request:

```
aws honeycode invoke-screen-automation \
--profile <my-auth-profile> \
--workbook-id <workbook-id> \
--app-id <app-id> \
--screen-id <screen-id> \
--screen-automation-id <screen-automation-id> \
--table-row-id <table-row-id> \
--client-request-token <client-request-token> \
--variables '{"Row": {"rawValue": "<Table row ID goes here>"}}'
```

Response:

The “Test” task is deleted.

Sample API calls using Python SDK

If you haven't installed the SDK already, follow the guide here.

You can use the code snippet below to set up a client. Here, we're using a profile to set up the session, but you can choose to do it in other ways.

```
import boto3
import json

session = boto3.Session(profile_name = 'sample-honeycode-profile')
honeycode_client = session.client('honeycode', region_name = 'us-west-2')
```

Making a basic query

An example of a basic query to read a list of tasks using the *GetScreenData* API. The following code snippet can be used to make a call to get rows from the Tasks table.

Request:

```
response = honeycode_client.get_screen_data(
    workbookId = '<workbookId>',
    appId = '<appId>',
    screenId = '<screenId>',
    maxResults = 2
)
print(json.dumps(response, indent = 4))
```

Note that we set *maxResults* to 2 to demonstrate pagination. The default is 100, and its inclusion is optional.

Response:

```
{
    "ResponseMetadata": {
        "RequestId": "<RequestId is here>",
        "HTTPStatusCode": 200,
        "HTTPHeaders": {
            "content-type": "application/json",
            "content-length": "1295",
            "connection": "keep-alive",
            "date": "Tue, 05 May 2020 22:01:41 GMT",
            "x-amzn-requestid": "<RequestId is here>",
            "x-amzn-remapped-x-amzn-requestid": "<RequestId is here>",
            "x-amzn-remapped-content-length": "1295",
```

```
            "x-amz-apigw-id": "MFBK0F3WPHcFcCg=",
            "x-amzn-trace-id": "<TraceId is here>",
            "x-amzn-remapped-date": "<Timestamp here>",
        },
        "RetryAttempts": 0
    },
    "results": {
        "nextToken": "<Next Token Value>",
        "results": {
            "Tasks List": {
                "headers": [
                    {
                        "format": "Text",
                        "name": "Task"
                    },
                    {
                        "format": "Text",
                        "name": "Status"
                    }
                ],
                "rows": [
                    {
                        "dataItems": [
                            {
                                "formattedValue": "Try API in Honeycode",
                                "rawValue": "Try API in Honeycode"
                            },
                            {
                                "formattedValue": "Not Started",
                                "rawValue": "Not Started"
                            }
                        ],
                        "tableRowId": "<A Row ID is here>"
                    },
                    {
                        "dataItems": [
                            {
                                "formattedValue": "Read a book",
                                "rawValue": "Read a book"
                            },
                            {
                                "formattedValue": "Not Started",
                                "rawValue": "Not Started"
                            }
                        ],
                        "tableRowId": "<A Row Id is here>"
                    }
                ]
            }
        },
    }
    "workbookCursor": <A number is here>,
    "nextToken": "<Token string is here>"
}
```


Using next token

This *GetScreenData* example takes the pagination token from the previous response and includes it in the input JSON file for the next call. *viewArn* and *maxResults* are the same as in previous request.

Note that *nextToken* is different for different calls, even with the same set of parameters. The token expires after one hour, so if you save the response somewhere and reuse it later, you'll get an error response that the token is expired. You'll need to run the previous request and use the token from the output into this request.

```
response = honeycode_client.get_screen_data(
    workbookId = '<workbook-id>',
    appId = '<app-id>',
    screenId = '<screen-id>',
    maxResults = 2
)

next_token = response['nextToken']

response = honeycode_client.get_screen_data(
    workbookId = '<workbook-id>',
    appId = '<app-id>',
    screenId = '<screen-id>',
    maxResults = 2,
    nextToken = next_token
)
print(json.dumps(response, indent = 2))
```

Passing named local variables

Request:

This *GetScreenData* example filters the Tasks table to display only tasks that are with the status "Not Started."

```
response = honeycode_client.get_screen_data(
    workbookId = '<workbook-id>',
    appId = '<app-id>',
    screenId = '<screen-id>',
    variables = {"Status Filter": { "rawValue": "Not Started"}}
)
print(json.dumps(response, indent = 2))
```

Adding, overwriting and deleting rows

This *InvokeScreenAutomation* example shows how to add, overwrite, and delete rows in a Tasks table.

```
# Create row automation
honeycode_client.invoke_screen_automation(
    workbookId = '<workbook-id>',
    appId = '<app-id>',
    screenId = '<create-row-screen-id>',
    screenAutomationId = '<screen-automation-id>',
    variables = {"Task": {"rawValue": "Test"}, "Status":{"rawValue": "Not Started"}}
)

# Update row automation
honeycode_client.invoke_screen_automation(
    workbookId = '<workbook-id>',
    appId = '<app-id>',
    screenId = '<update-row-screen-id>',
    screenAutomationId = '<screen-automation-id>',
    variables = {"Row": {"rawValue": "<Table row ID goes here>"}, "Task": {"rawValue":
 "Test"}, "Status":{"rawValue": "In Progress"}}
)

# Delete row automation
honeycode_client.invoke_screen_automation(
    workbookId = '<workbook-id>',
    appId = '<app-id>',
    screenId = '<delete-row-screen-id>',
    screenAutomationId = '<screen-automation-id>',
```

```
    variables = {"Row": {"rawValue": "<Table row ID goes here>"}}
)
```

Table Metadata APIs

Table metadata APIs allow you to retrieve metadata about tables in Honeycode workbooks. The APIs are:

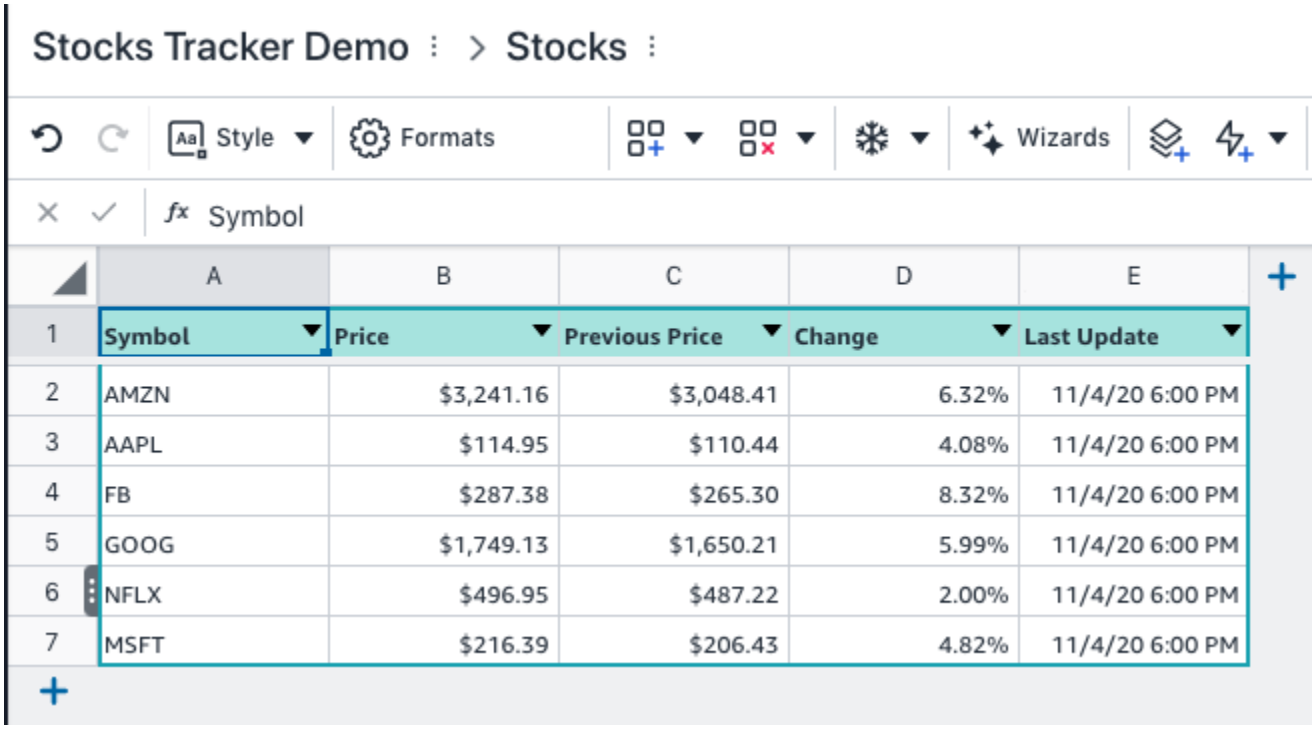
- ListTables
- ListTableColumns

Setting up for Table Metadata APIs

For these APIs, you'll need to first create a table and add some data.

Create table

If you haven't already done so, create a new workbook and add a new table either by importing a CSV file or by adding data manually. In the example below, we start with a simple table that tracks stock prices.



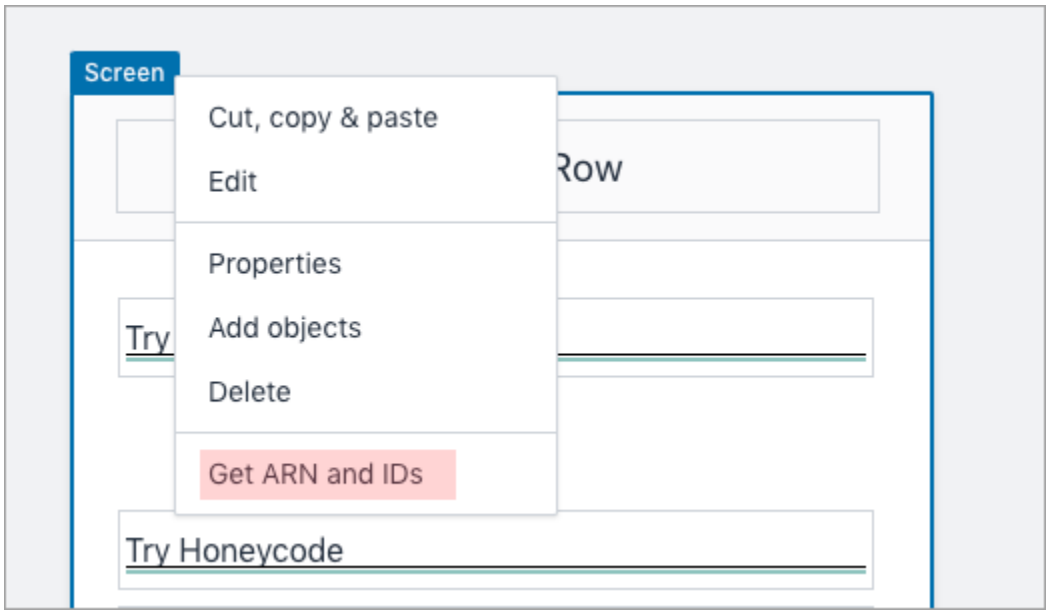
	A	B	C	D	E
1	Symbol	Price	Previous Price	Change	Last Update
2	AMZN	$3,241.16	$3,048.41	6.32%	11/4/20 6:00 PM
3	AAPL	$114.95	$110.44	4.08%	11/4/20 6:00 PM
4	FB	$287.38	$265.30	8.32%	11/4/20 6:00 PM
5	GOOG	$1,749.13	$1,650.21	5.99%	11/4/20 6:00 PM
6	NFLX	$496.95	$487.22	2.00%	11/4/20 6:00 PM
7	MSFT	$216.39	$206.43	4.82%	11/4/20 6:00 PM

ARNs and Honeycode IDs

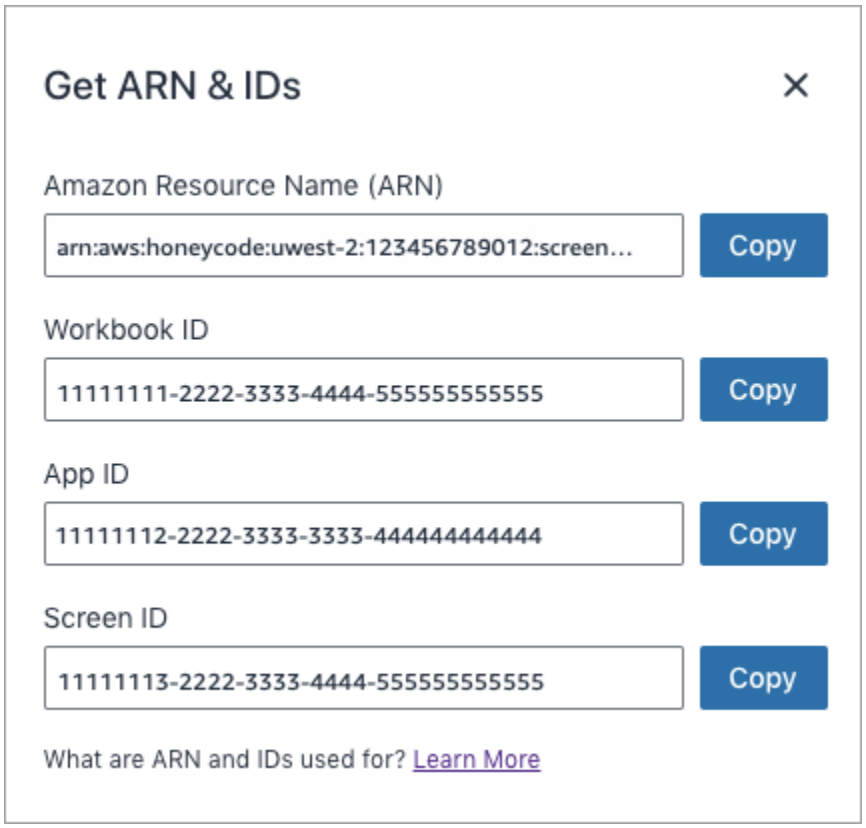
Table metadata APIs require the workbook id as input. You will need an app to get the workbook id. Simply right-click on any app object in builder to access the Get ARN and IDs modal. If you don't have a use case for an app, you can use the App Wizard to quickly create a simple app to grab the Workbook ID and then delete the app as necessary.

Accessing Workbook ID

In any app screen, you can right-click to bring up a menu. Select Get ARN and IDs



From the modal that appears, you can copy the ID for the workbook. You can ignore the other fields on that modal (Amazon Resource Name, App ID, Screen ID) as they are not needed for table metadata APIs.



ARNs and authorization

If you are using the managed IAM policies *AmazonHoneycodeWorkbookFullAccess* or *AmazonHoneycodeFullAccess*, you will not need the ARN for authorization. The ARN is only required to to set up authorization using IAM at a granular level.

The Honeycode resources that need to be defined in custom IAM policies for table metadata APIs are given below. Click the links to get more details on how to construct ARNs for those resources.

- *ListTables*: workbook
- *ListTableColumns*: table

ListTables

The ListTables API allows you to retrieve a list of all the tables in a workbook.

To find more details about using this API check the ListTables API Reference page.

In the examples below, replace `<workbook-id>` with your workbook id.

AWS CLI Example

```
aws honeycode list-tables \
  --workbook-id "<workboook-id>"
```

Python SDK Example

```
response = honeycode_client.list_tables(
    workbookId = '<workbook-id>')
```

Response

```
{
    "tables": [
        {
            "tableId": "<table-id>",
            "tableName": "Stocks"
        }
    ],
    "workbookCursor": 1273158992
}
```

ListTableColumns

The ListTableColumns API allows you to retrieve a list of all the columns in a table in a workbook.

To find more details about using this API check the ListTableColumns API Reference page.

In the examples below, replace `<workbook-id>` with your workbook id and `<table-id>` with the table id from the response of ListTables API call.

AWS CLI Example

```
aws honeycode list-table-columns \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>"
```

Python SDK Example

```
response = honeycode_client.list_table_columns(
```

```
    workbookId = '<workbook-id>',
    tableId = '<table-id>')
```

Response

```
{
    "tableColumns": [
        {
            "format": "AUTO",
            "tableColumnId": "<symbol-column-id>",
            "tableColumnName": "Symbol"
        },
        {
            "format": "CURRENCY",
            "tableColumnId": "<price-column-id>",
            "tableColumnName": "Price"
        },
        {
            "format": "CURRENCY",
            "tableColumnId": "<previous-price-column-id>",
            "tableColumnName": "Previous Price"
        },
        {
            "format": "PERCENTAGE",
            "tableColumnId": "<percentage-change-column-id>",
            "tableColumnName": "Change"
        },
        {
            "format": "DATE_TIME",
            "tableColumnId": "<last-update-column-id>",
            "tableColumnName": "Last Update"
        }
    ],
    "workbookCursor": 1288302476
}
```

Table Row Operation APIs

The table row operation APIs are useful when wanting to connect your Honeycode tables with external sources. There are several things that you can do:

- Create or add new rows to a table
- Read or retrieve rows from a table
- Update or edit rows in a table
- Delete or remove rows from a table

The APIs are:

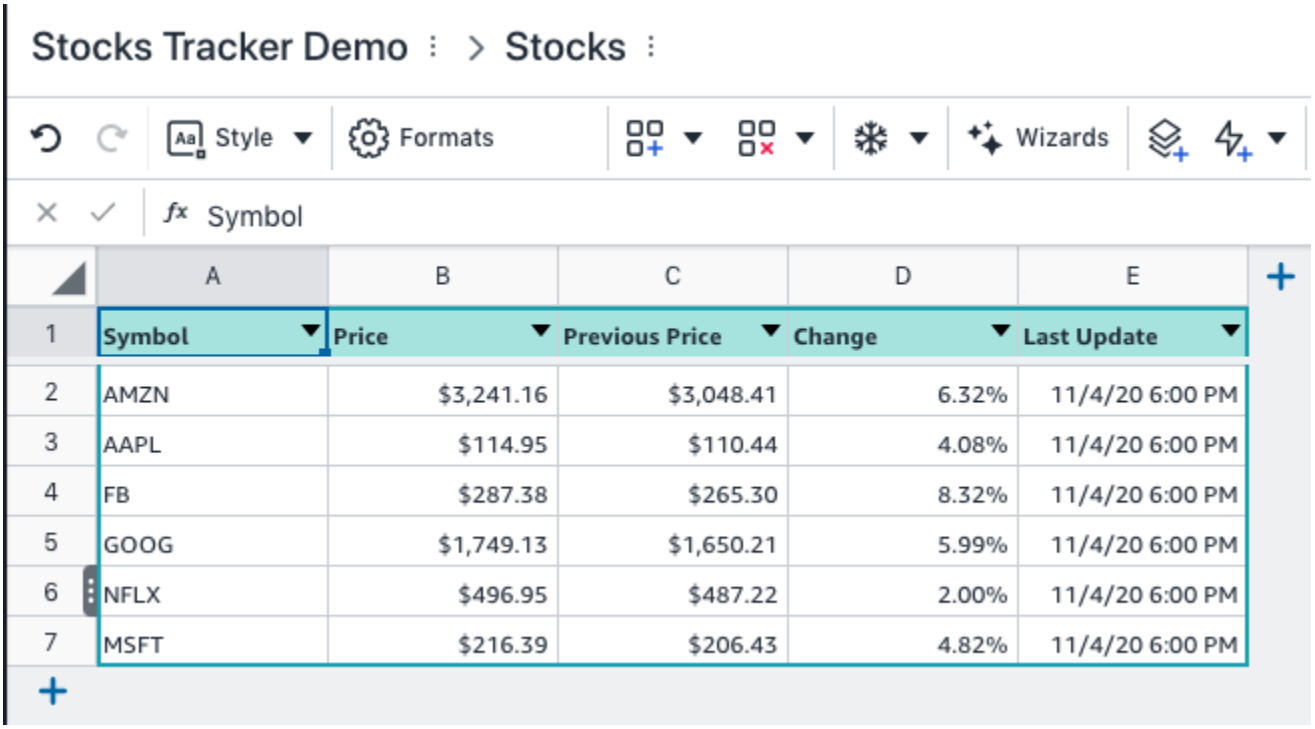
- ListTableRows
- QueryTableRows
- BatchCreateTableRows
- BatchUpdateTableRows
- BatchUpsertTableRows
- BatchDeleteTableRows

Setting up for table row operation APIs

For these APIs, you'll need to first create a table and add some data.

Create table

If you haven't already done so, create a new workbook and add a new table either by importing a CSV file or by adding data manually. In the example below, we start with a simple table that tracks stock prices.



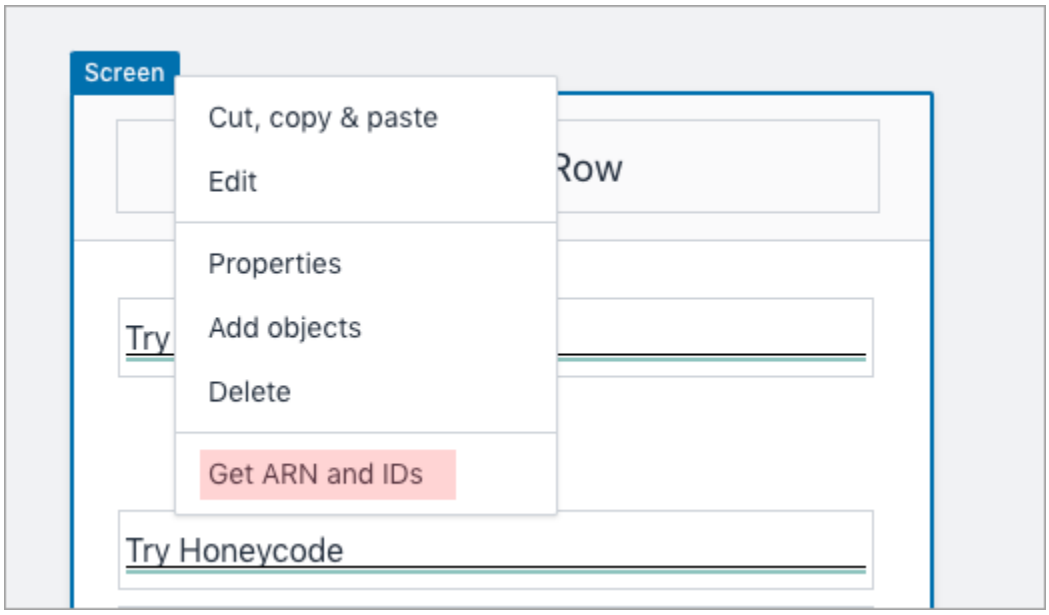
	A	B	C	D	E
1	Symbol	Price	Previous Price	Change	Last Update
2	AMZN	$3,241.16	$3,048.41	6.32%	11/4/20 6:00 PM
3	AAPL	$114.95	$110.44	4.08%	11/4/20 6:00 PM
4	FB	$287.38	$265.30	8.32%	11/4/20 6:00 PM
5	GOOG	$1,749.13	$1,650.21	5.99%	11/4/20 6:00 PM
6	NFLX	$496.95	$487.22	2.00%	11/4/20 6:00 PM
7	MSFT	$216.39	$206.43	4.82%	11/4/20 6:00 PM

ARNs and Honeycode IDs

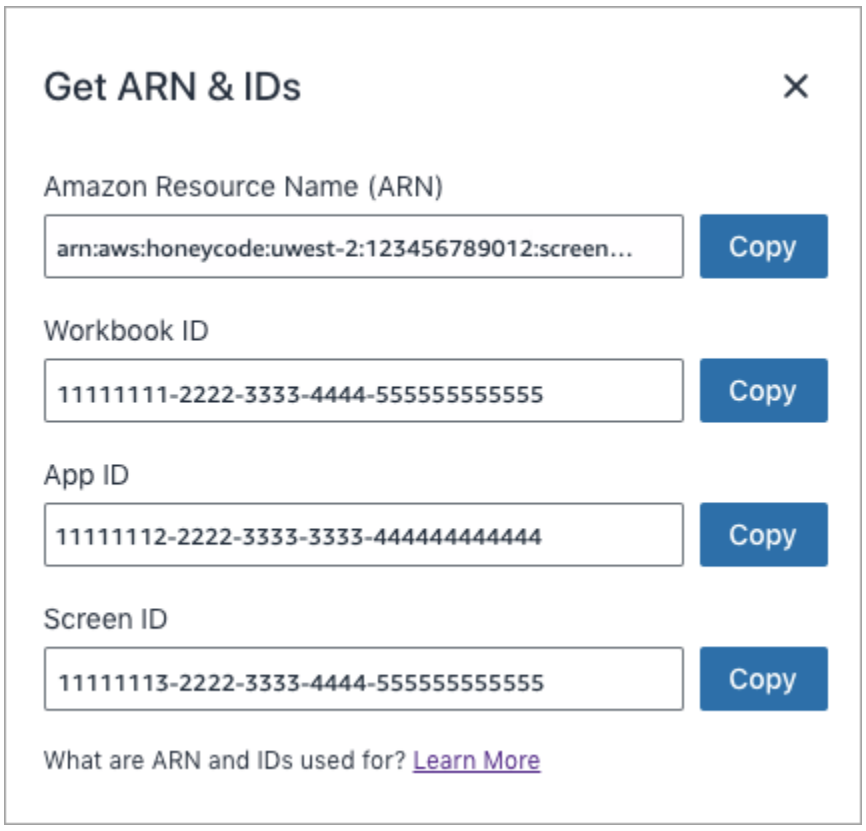
Table row operation APIs require the workbook id as input. You will need an app to get the workbook id. Simply right-click on any app object in builder to access the Get ARN and IDs modal. If you don't have a use case for an app, you can use the App Wizard to quickly create a simple app to grab the Workbook ID and then delete the app as necessary.

Accessing Workbook ID

In any app screen, you can right-click to bring up a menu. Select Get ARN and IDs



From the modal that appears, you can copy the ID for the workbook. You can ignore the other fields on that modal (Amazon Resource Name, App ID, Screen ID) as they are not needed for table row operation APIs.



ARNs and authorization

If you are using the managed IAM policies *AmazonHoneycodeWorkbookFullAccess* or *AmazonHoneycodeFullAccess*, you will not need the ARN for authorization. The ARN is only required to to set up authorization using IAM at a granular level.

The Honeycode resources that need to be defined in custom IAM policies for table row operation APIs are given below. Click the links to get more details on how to construct ARNs for those resources.

- *ListTableRows*: table
- *QueryTableRows*: table
- *BatchCreateTableRows*: table
- *BatchUpdateTableRows*: table
- *BatchUpsertTableRows*: table
- *BatchDeleteTableRows*: table

ListTableRows

The ListTableRows API allows you to retrieve a list of rows in a table in a workbook.

To find more details about using this API check the ListTableRows API Reference page.

In the examples below, replace `<workbook-id>` with your workbook id and `<table-id>` with the table id returned by the ListTables API call. Note that maxResults is set to 3 to show how pagination works.

AWS CLI Example

```
aws honeycode list-table-rows \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --max-results 3
```

Python SDK Example

```
response = honeycode_client.list_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    maxResults = 3)
```

Response

```
{
    "columnIds": [
        "<symbol-column-id>",
        "<price-column-id>",
        "<previous-price-column-id>",
        "<percentage-change-column-id>",
        "<last-update-column-id>"
    ],
    "nextToken": "<token-for-page-2>",
    "rows": [
        {
            "cells": [
                {
                    "format": "AUTO",
                    "formattedValue": "AMZN",
                    "rawValue": "AMZN"
```

```
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$3,241.16",
                "rawValue": "3241.16"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$3,048.41",
                "rawValue": "3048.41"
            },
            {
                "format": "PERCENTAGE",
                "formattedValue": "6.32%",
                "formula": "=[Price]/[Previous Price]-1",
                "rawValue": "0.06322968367116"
            },
            {
                "format": "DATE_TIME",
                "formattedValue": "11/4/20 6:00 PM",
                "rawValue": "44139.75"
            }
        ],
        "rowId": "<amzn-row-id>"
    },
    {
        "cells": [
            {
                "format": "AUTO",
                "formattedValue": "AAPL",
                "rawValue": "AAPL"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$114.95",
                "rawValue": "114.95"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$110.44",
                "rawValue": "110.44"
            },
            {
                "format": "PERCENTAGE",
                "formattedValue": "4.08%",
                "formula": "=[Price]/[Previous Price]-1",
                "rawValue": "0.040836653386454"
            },
            {
                "format": "DATE_TIME",
                "formattedValue": "11/4/20 6:00 PM",
                "rawValue": "44139.75"
            }
        ],
        "rowId": "<aapl-row-id>"
    },
    {
        "cells": [
            {
                "format": "AUTO",
                "formattedValue": "FB",
                "rawValue": "FB"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$287.38",
```

```
                    "rawValue": "287.38"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$265.30",
                    "rawValue": "265.3"
                },
                {
                    "format": "PERCENTAGE",
                    "formattedValue": "8.32%",
                    "formula": "=[Price]/[Previous Price]-1",
                    "rawValue": "0.083226535996985"
                },
                {
                    "format": "DATE_TIME",
                    "formattedValue": "11/4/20 6:00 PM",
                    "rawValue": "44139.75"
                }
            ],
            "rowId": "<fb-row-id>"
        }
    ],
    "workbookCursor": 1288302476
}
```

Note that the `nextToken` in the response indicates that there is more data to load. You can pass that value as the `nextToken` parameter in a subsequent request to retrieve page 2. The new response will have no `nextToken` since all 6 rows in the table have been loaded.

AWS CLI Example (nextToken)

```
aws honeycode list-table-rows \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --max-results 3 \
  --next-token "<token-for-page-2>"
```

Python SDK Example (nextToken)

```
response = honeycode_client.list_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    maxResults = 3,
    nextToken = '<token-for-page-2>')
```

Response (nextToken)

```
{
    "columnIds": [
        "<symbol-column-id>",
        "<price-column-id>",
        "<previous-price-column-id>",
        "<percentage-change-column-id>",
        "<last-update-column-id>"
    ],
    "rows": [
        {
            "cells": [
                {
                    "format": "AUTO",
```

```
                "formattedValue": "GOOG",
                "rawValue": "GOOG"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$1,749.13",
                "rawValue": "1749.13"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$1,650.21",
                "rawValue": "1650.21"
            },
            {
                "format": "PERCENTAGE",
                "formattedValue": "5.99%",
                "formula": "=[Price]/[Previous Price]-1",
                "rawValue": "0.05994388592967"
            },
            {
                "format": "DATE_TIME",
                "formattedValue": "11/4/20 6:00 PM",
                "rawValue": "44139.75"
            }
        ],
        "rowId": "<goog-row-id>"
    },
    {
        "cells": [
            {
                "format": "AUTO",
                "formattedValue": "NFLX",
                "rawValue": "NFLX"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$496.95",
                "rawValue": "496.95"
            },
            {
                "format": "CURRENCY",
                "formattedValue": "$487.22",
                "rawValue": "487.22"
            },
            {
                "format": "PERCENTAGE",
                "formattedValue": "2.00%",
                "formula": "=[Price]/[Previous Price]-1",
                "rawValue": "0.019970444563031"
            },
            {
                "format": "DATE_TIME",
                "formattedValue": "11/4/20 6:00 PM",
                "rawValue": "44139.75"
            }
        ],
        "rowId": "<nflx-row-id>"
    },
    {
        "cells": [
            {
                "format": "AUTO",
                "formattedValue": "MSFT",
                "rawValue": "MSFT"
            },
            {
```

```
                    "format": "CURRENCY",
                    "formattedValue": "$216.39",
                    "rawValue": "216.39"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$206.43",
                    "rawValue": "206.43"
                },
                {
                    "format": "PERCENTAGE",
                    "formattedValue": "4.82%",
                    "formula": "=[Price]/[Previous Price]-1",
                    "rawValue": "0.04824880104636"
                },
                {
                    "format": "DATE_TIME",
                    "formattedValue": "11/4/20 6:00 PM",
                    "rawValue": "44139.75"
                }
            ],
            "rowId": "<msft-row-id>"
        }
    ],
    "workbookCursor": 1288302476
}
```

ListTableRows API can also be called with specific row ids in the input. In this case, the API returns only the requested row ids.

AWS CLI Example (row ids)

```
aws honeycode list-table-rows \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --row-ids '["<amzn-row-id>", "<msft-row-id>"]'
```

Python SDK Example (row ids)

```
response = honeycode_client.list_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    rowIds = ['<amzn-row-id>', '<msft-row-id>'])
```

Response

```
{
    "columnIds": [
        "<symbol-column-id>",
        "<price-column-id>",
        "<previous-price-column-id>",
        "<percentage-change-column-id>",
        "<last-update-column-id>"
    ],
    "rows": [
        {
            "cells": [
                {
                    "format": "AUTO",
                    "formattedValue": "AMZN",
```

```
                    "rawValue": "AMZN"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$3,241.16",
                    "rawValue": "3241.16"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$3,048.41",
                    "rawValue": "3048.41"
                },
                {
                    "format": "PERCENTAGE",
                    "formattedValue": "6.32%",
                    "formula": "=[Price]/[Previous Price]-1",
                    "rawValue": "0.06322968367116"
                },
                {
                    "format": "DATE_TIME",
                    "formattedValue": "11/4/20 6:00 PM",
                    "rawValue": "44139.75"
                }
            ],
            "rowId": "<amzn-row-id>"
        },
        {
            "cells": [
                {
                    "format": "AUTO",
                    "formattedValue": "MSFT",
                    "rawValue": "MSFT"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$216.39",
                    "rawValue": "216.39"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$206.43",
                    "rawValue": "206.43"
                },
                {
                    "format": "PERCENTAGE",
                    "formattedValue": "4.82%",
                    "formula": "=[Price]/[Previous Price]-1",
                    "rawValue": "0.04824880104636"
                },
                {
                    "format": "DATE_TIME",
                    "formattedValue": "11/4/20 6:00 PM",
                    "rawValue": "44139.75"
                }
            ],
            "rowId": "<msft-row-id>"
        }
    ],
    "workbookCursor": 1288302476
}
```


QueryTableRows

The QueryTableRows API can be used to query for specific rows in the table using a filter function.

To find more details about using this API check the QueryTableRows API Reference page.

The following example finds all stocks that had more than 6.00% change from the previous price. Replace <workbook-id> with your workbook id and <table-id> with the table id from the response of ListTables API call.

AWS CLI Example

```
aws honeycode query-table-rows \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --filter-formula '{"formula": "=Filter(Stocks,\"Stocks[Change]>0.06\")"}'
```

Python SDK Example

```
response = honeycode_client.query_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    filterFormula = { "formula": "=Filter(Stocks,\"Stocks[Change]>0.06\")"} )
```

Response

```
{
    "columnIds": [
        "<symbol-column-id>",
        "<price-column-id>",
        "<previous-price-column-id>",
        "<percentage-change-column-id>",
        "<last-update-column-id>"
    ],
    "rows": [
        {
            "cells": [
                {
                    "format": "AUTO",
                    "formattedValue": "AMZN",
                    "rawValue": "AMZN"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$3,241.16",
                    "rawValue": "3241.16"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$3,048.41",
                    "rawValue": "3048.41"
                },
                {
                    "format": "PERCENTAGE",
                    "formattedValue": "6.32%",
                    "formula": "=[Price]/[Previous Price]-1",
                    "rawValue": "0.06322968367116"
                },
                {
                    "format": "DATE_TIME",
                    "formattedValue": "11/4/20 6:00 PM",
                    "rawValue": "44139.75"
                }
            ],
```

```
            "rowId": "<amzn-row-id>"
        },
        {
            "cells": [
                {
                    "format": "AUTO",
                    "formattedValue": "FB",
                    "rawValue": "FB"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$287.38",
                    "rawValue": "287.38"
                },
                {
                    "format": "CURRENCY",
                    "formattedValue": "$265.30",
                    "rawValue": "265.3"
                },
                {
                    "format": "PERCENTAGE",
                    "formattedValue": "8.32%",
                    "formula": "=[Price]/[Previous Price]-1",
                    "rawValue": "0.083226535996985"
                },
                {
                    "format": "DATE_TIME",
                    "formattedValue": "11/4/20 6:00 PM",
                    "rawValue": "44139.75"
                }
            ],
            "rowId": "<fb-row-id>"
        }
    ],
    "workbookCursor": 1288302476
}
```

BatchCreateTableRows

The BatchCreateTableRows API can be used to append a batch of rows to the end of a table.

To find more details about using this API check the BatchCreateTableRows API Reference page.

The following example demonstrates using BatchCreateTableRows API to add two rows at the end of a table. Replace `<workbook-id>` with your workbook id and `<table-id>` with the table id from the response of ListTables API call.

Note:

- This API takes `batchItemId` in the input. This is an identifier that you can assign to that particular row so that you can link the row id in the response with the item in the request.
- The `cellsToCreate` map needs the ids of the columns in the table. You can get these ids from the ListTableColumns API response.
- The column `Change` has a column level formula. So it does not need to be included in the input as the column formula will be automatically applied to the new rows. You can include the column in the input if you want to override the column formula for the new rows with a different value.

AWS CLI Example

```
aws honeycode batch-create-table-rows \
```

```
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --rows-to-create '[
    {
      "batchItemId": "item-001",
      "cellsToCreate": {
        "<symbol-column-id>": { "fact": "AAA" },
        "<price-column-id>": { "fact": "23.47" },
        "<previous-price-column-id>": { "fact": "27.27" },
        "<last-updated-column-id>": { "fact": "11/5/20 6:00 PM" }
      }
    },
    {
      "batchItemId": "item-002",
      "cellsToCreate": {
        "<symbol-column-id>": { "fact": "BBB" },
        "<price-column-id>": { "fact": "108" },
        "<previous-price-column-id>": { "fact": "127.2" },
        "<last-updated-column-id>": { "fact": "11/5/20 6:00 PM" }
      }
    }
  ]'
```

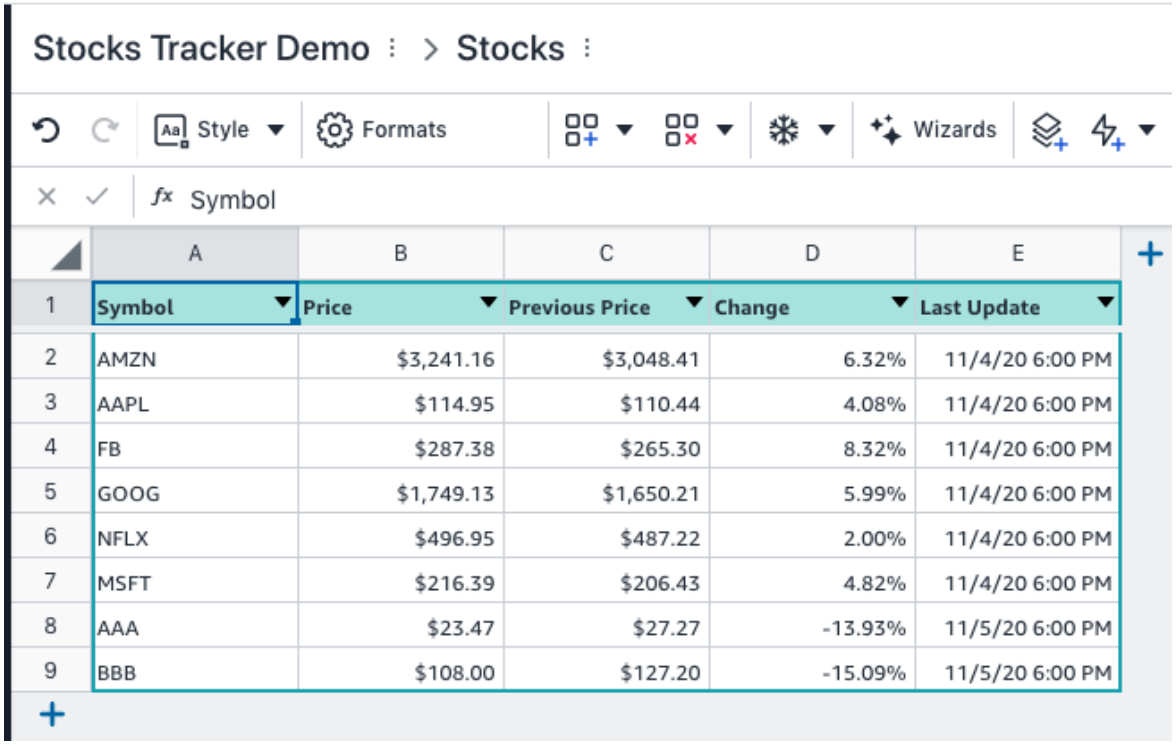
Python SDK Example

```
def create_row_data(batch_item_id, symbol, price, previous_price, last_updated):
    return {
        "batchItemId": batch_item_id,
        "cellsToCreate": {
            "<symbol-column-id>": { "fact": symbol },
            "<price-column-id>": { "fact": price },
            "<previous-price-column-id>": { "fact": previous_price},
            "<last-updated-column-id>": { "fact": last_updated }
        }
    }

response = honeycode_client.batch_create_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    rowsToCreate = [
        create_row_data("item-001", "AAA", "23.47", "27.27", "11/5/20 6:00 PM"),
        create_row_data("item-002", "BBB", "108", "127.2", "11/5/20 6:00 PM")
    ])
```

Response

```
{
    "createdRows": {
        "item-001": "<aaa-row-id>",
        "item-002": "<bbb-row-id>"
    },
    "workbookCursor": 1288497196
}
```

After the rows are inserted, this is how the table looks:



	A	B	C	D	E
1	Symbol	Price	Previous Price	Change	Last Update
2	AMZN	$3,241.16	$3,048.41	6.32%	11/4/20 6:00 PM
3	AAPL	$114.95	$110.44	4.08%	11/4/20 6:00 PM
4	FB	$287.38	$265.30	8.32%	11/4/20 6:00 PM
5	GOOG	$1,749.13	$1,650.21	5.99%	11/4/20 6:00 PM
6	NFLX	$496.95	$487.22	2.00%	11/4/20 6:00 PM
7	MSFT	$216.39	$206.43	4.82%	11/4/20 6:00 PM
8	AAA	$23.47	$27.27	-13.93%	11/5/20 6:00 PM
9	BBB	$108.00	$127.20	-15.09%	11/5/20 6:00 PM

BatchUpdateTableRows

The BatchUpdateTableRows API can be used to update the data in one or more columns of specific rows in a table.

To find more details about using this API check the BatchUpdateTableRows API Reference page.

The following example demonstrates using BatchUpdateTableRows API to update the price column of one row and the date column of another row in the same request. Replace `<workbook-id>` with your workbook id and `<table-id>` with the table id from the response of ListTables API call. The row ids `<aaa-row-id>` and `<bbb-row-id>` are from the output of the BatchCreateTableRows API call.

AWS CLI Example

```
aws honeycode batch-update-table-rows \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --rows-to-update '[
    {
      "rowId": "<aaa-row-id>",
      "cellsToUpdate": {
        "<price-column-id>": { "fact": "24.74" }
      }
    },
    {
      "rowId": "<bbb-row-id>",
      "cellsToUpdate": {
        "<last-updated-column-id>": { "fact": "11/7 5:59 PM" }
      }
    }
  ]'
```

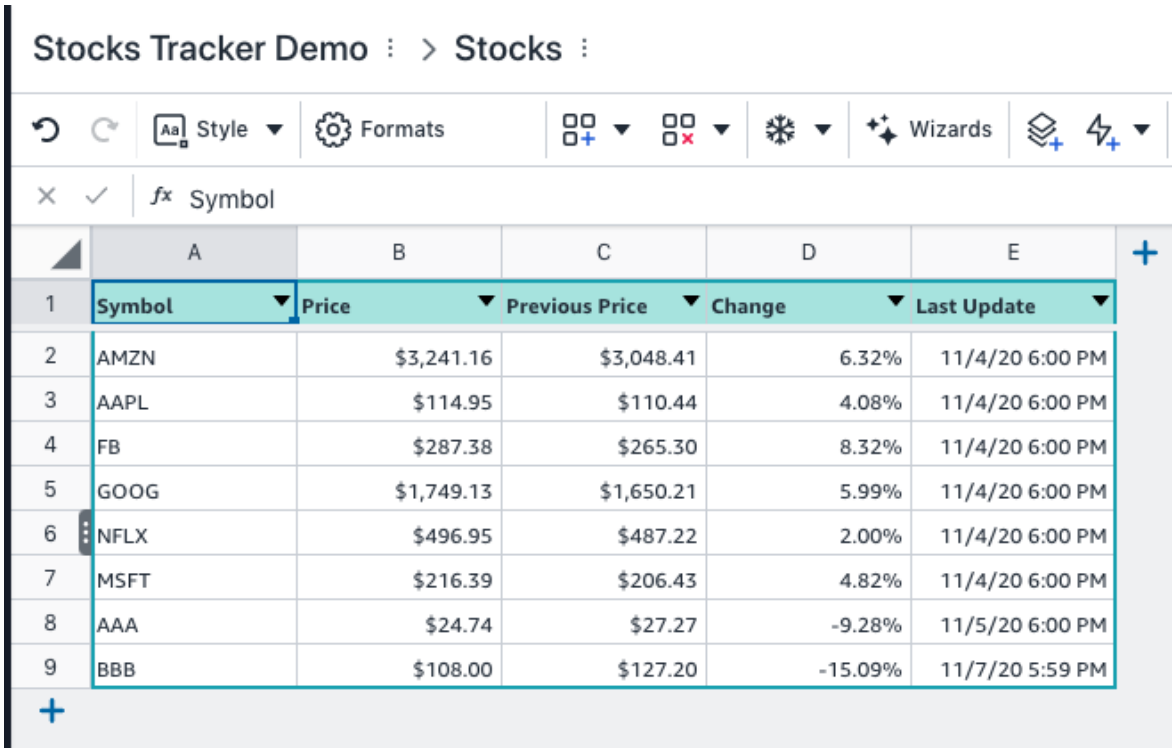
Python SDK Example

```
response = honeycode_client.batch_update_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    rowsToUpdate = [
        {
          "rowId": "<aaa-row-id>",
          "cellsToUpdate": {
            "<price-column-id>": { "fact": "24.74" }
          }
        },
        {
          "rowId": "<bbb-row-id>",
          "cellsToUpdate": {
            "<last-updated-column-id>": { "fact": "11/7 5:59 PM" }
          }
        }
    ])
```

Response

```
{
    "workbookCursor": 1288538679
}
```

After the rows are updated, this is how the table looks:



	A	B	C	D	E
1	Symbol	Price	Previous Price	Change	Last Update
2	AMZN	$3,241.16	$3,048.41	6.32%	11/4/20 6:00 PM
3	AAPL	$114.95	$110.44	4.08%	11/4/20 6:00 PM
4	FB	$287.38	$265.30	8.32%	11/4/20 6:00 PM
5	GOOG	$1,749.13	$1,650.21	5.99%	11/4/20 6:00 PM
6	NFLX	$496.95	$487.22	2.00%	11/4/20 6:00 PM
7	MSFT	$216.39	$206.43	4.82%	11/4/20 6:00 PM
8	AAA	$24.74	$27.27	-9.28%	11/5/20 6:00 PM
9	BBB	$108.00	$127.20	-15.09%	11/7/20 5:59 PM

BatchUpsertTableRows

The BatchUpsertTableRows API can be used to upsert one or more rows in a table. The upsert operation takes a filter formula as input and uses it to find matching rows in the table. If matching rows are found, cells in those rows are updated as specified in the request. If a matching row is not found, a new row is created and cells in the new row are set as specified in the request.

To find more details about using this API check the BatchUpsertTableRows API Reference page.

The following example shows this API being called with two batch items. One batch item finds an existing row and updates it and the second batch item creates a new row. Replace `<workbook-id>` with your workbook id and `<table-id>` with the table id from the response of ListTables API call.

AWS CLI Example

```
aws honeycode batch-upsert-table-rows \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --rows-to-upsert '[
    {
      "batchItemId": "item-001",
      "filter": { "formula": "=Filter(Stocks,\"Stocks[Symbol]=%\",\"BBB\")" },
      "cellsToUpdate": {
        "<symbol-column-id>": { "fact": "BBB" },
        "<price-column-id>": { "fact": "25.32" },
        "<previous-price-column-id>": { "fact": "28.76" },
        "<last-updated-column-id>": { "fact": "11/8 6:13 PM" }
      }
    },
    {
      "batchItemId": "item-002",
      "filter": { "formula": "=Filter(Stocks,\"Stocks[Symbol]=%\",\"CCC\")" },
      "cellsToUpdate": {
        "<symbol-column-id>": { "fact": "CCC" },
        "<price-column-id>": { "fact": "110.8" },
        "<previous-price-column-id>": { "fact": "108.10" },
        "<last-updated-column-id>": { "fact": "11/8 6:13 PM" }
      }
    }
  ]'
```

Python SDK Example

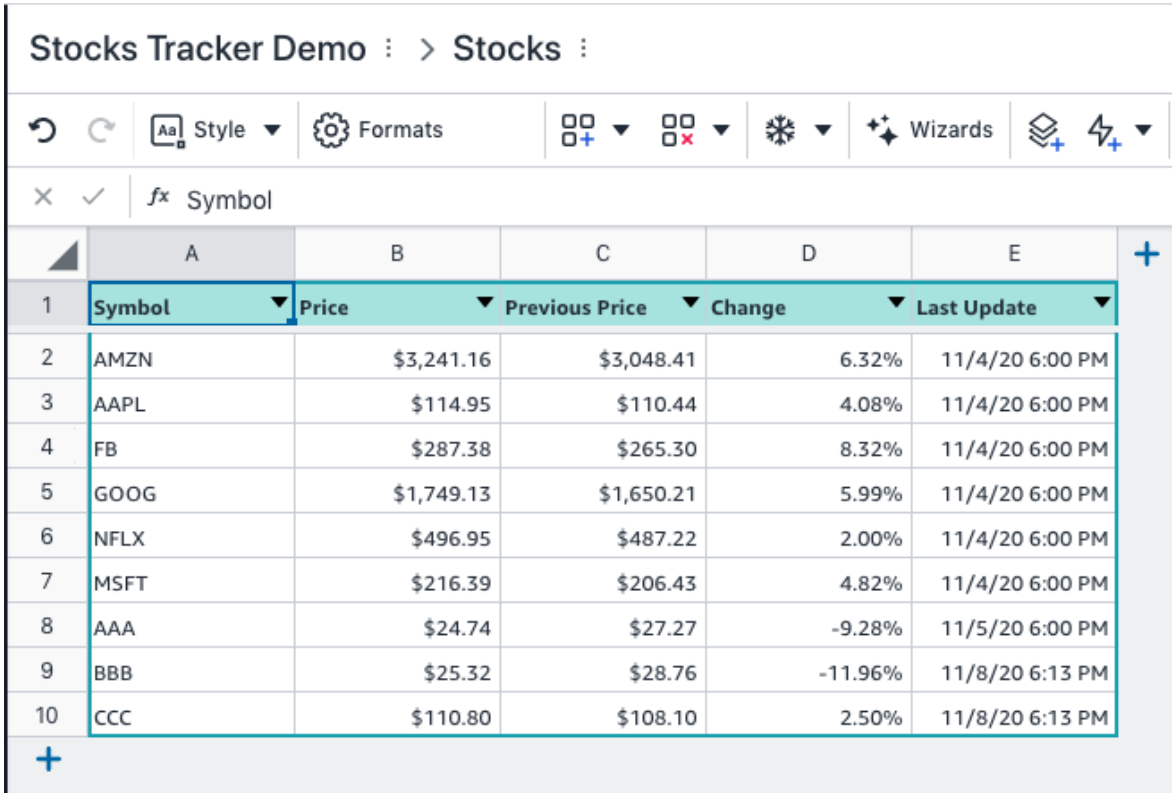
```
def upsert_row_data(batch_item_id, filter_formula, symbol, price, previous_price,
 last_updated):
    return {
        "batchItemId": batch_item_id,
        "filter": { "formula": filter_formula },
        "cellsToUpdate": {
            "<symbol-column-id>": { "fact": symbol },
            "<price-column-id>": { "fact": price },
            "<previous-price-column-id>": { "fact": previous_price},
            "<last-updated-column-id>": { "fact": last_updated }
        }
    }

response = honeycode_client.batch_upsert_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    rowsToUpsert = [
```

```
        upsert_row_data("item-001", "=Filter(Stocks,\"Stocks[Symbol]=%\",\"BBB\")", "BBB",
 "25.32", "28.76", "11/8 6:13 PM"),
        upsert_row_data("item-002", "=Filter(Stocks,\"Stocks[Symbol]=%\",\"CCC\")", "CCC",
 "110.8", "108.10", "11/8 6:13 PM")
    ])
```

Response

```
{
    "rows": {
        "item-001": {
            "rowIds": [ "<bbb-row-id>" ],
            "upsertAction": "UPDATED"
        },
        "item-002": {
            "rowIds": [ "<ccc-row-id>" ],
            "upsertAction": "APPENDED"
        }
    },
    "workbookCursor": 1288566784
}
```

After the rows are upserted, this is how the table looks:



	A	B	C	D	E
1	Symbol	Price	Previous Price	Change	Last Update
2	AMZN	$3,241.16	$3,048.41	6.32%	11/4/20 6:00 PM
3	AAPL	$114.95	$110.44	4.08%	11/4/20 6:00 PM
4	FB	$287.38	$265.30	8.32%	11/4/20 6:00 PM
5	GOOG	$1,749.13	$1,650.21	5.99%	11/4/20 6:00 PM
6	NFLX	$496.95	$487.22	2.00%	11/4/20 6:00 PM
7	MSFT	$216.39	$206.43	4.82%	11/4/20 6:00 PM
8	AAA	$24.74	$27.27	-9.28%	11/5/20 6:00 PM
9	BBB	$25.32	$28.76	-11.96%	11/8/20 6:13 PM
10	CCC	$110.80	$108.10	2.50%	11/8/20 6:13 PM

BatchDeleteTableRows

The BatchDeleteTableRows API will delete the list of rows passed in the input.

To find more details about using this API check the BatchDeleteTableRows API Reference page.

The following example illustrates this by deleting the rows with stock symbols AAA , BBB and CCC using their row ids. Replace `<workbook-id>` with your workbook id and `<table-id>` with the table id from the response of ListTables API call.

AWS CLI Example

```
aws honeycode batch-delete-table-rows \
  --workbook-id "<workboook-id>" \
  --table-id "<table-id>" \
  --rowIds '["<aaa-row-id>", "<bbb-row-id">, "<ccc-row-id>"]'
```

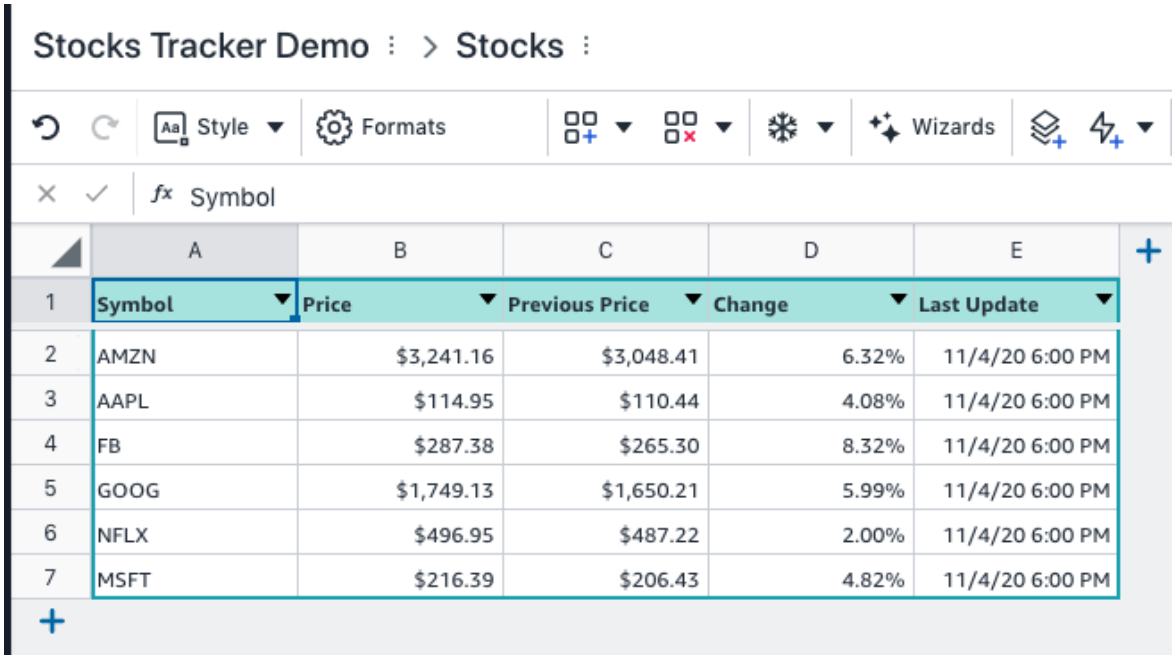
Python SDK Example

```
response = honeycode_client.batch_delete_table_rows(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    rowIds = [ "<aaa-row-id>", "<bbb-row-id">, "<ccc-row-id>" ])
```

Response

```
{
   "workbookCursor": 1288604696
}
```

After the rows are deleted, this is how the table looks:



Import APIs

The import APIs are useful when you want to bulk load data into tables in a workbook. The APIs are:

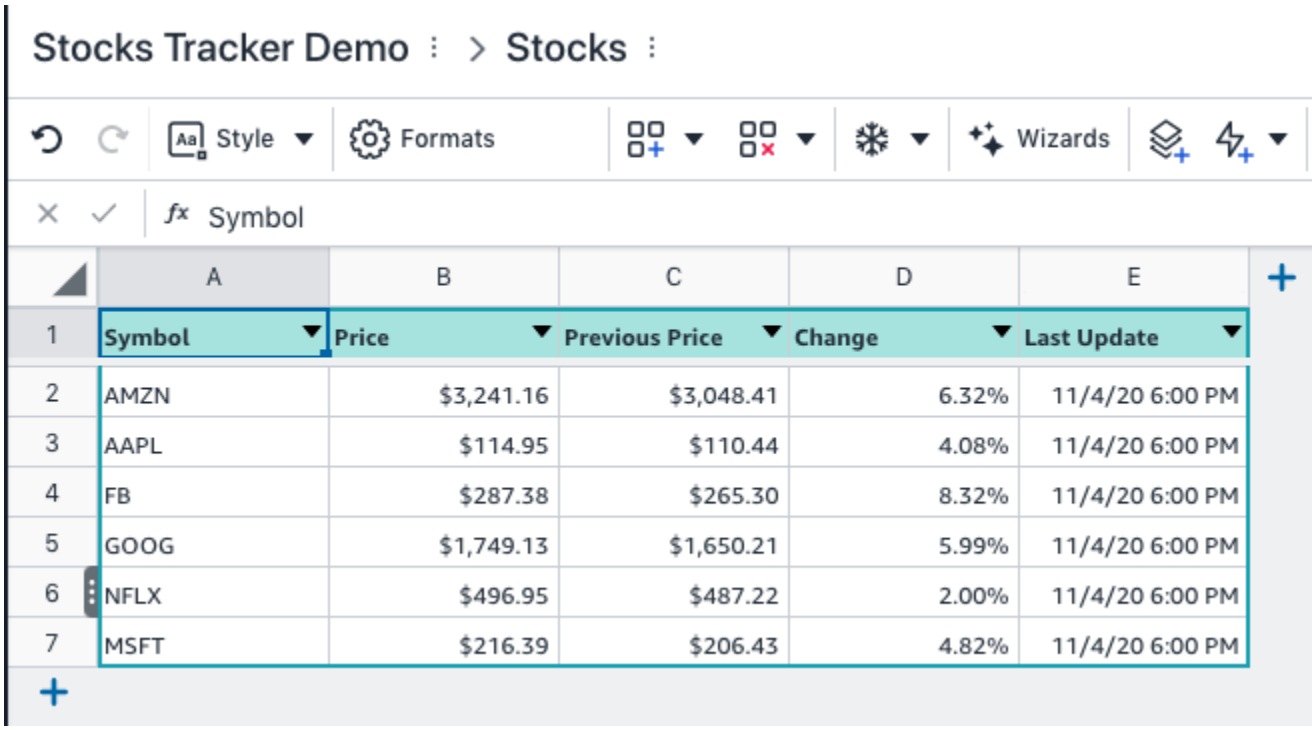
- StartTableDataImportJob
- DescribeTableDataImportJob

Setting up for import APIs

For the import APIs, you'll need to first create a table.

Create table

If you haven't already done so, create a new workbook and add a new table. In the examples below, we use a simple table that tracks stock prices.



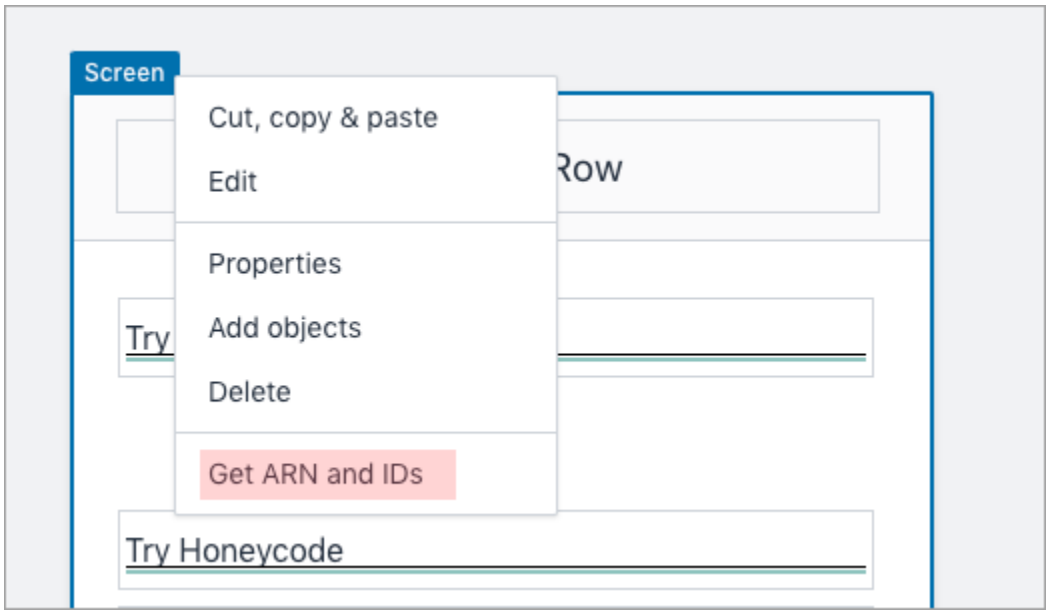
	A	B	C	D	E
1	Symbol	Price	Previous Price	Change	Last Update
2	AMZN	$3,241.16	$3,048.41	6.32%	11/4/20 6:00 PM
3	AAPL	$114.95	$110.44	4.08%	11/4/20 6:00 PM
4	FB	$287.38	$265.30	8.32%	11/4/20 6:00 PM
5	GOOG	$1,749.13	$1,650.21	5.99%	11/4/20 6:00 PM
6	NFLX	$496.95	$487.22	2.00%	11/4/20 6:00 PM
7	MSFT	$216.39	$206.43	4.82%	11/4/20 6:00 PM

ARNs and Honeycode IDs

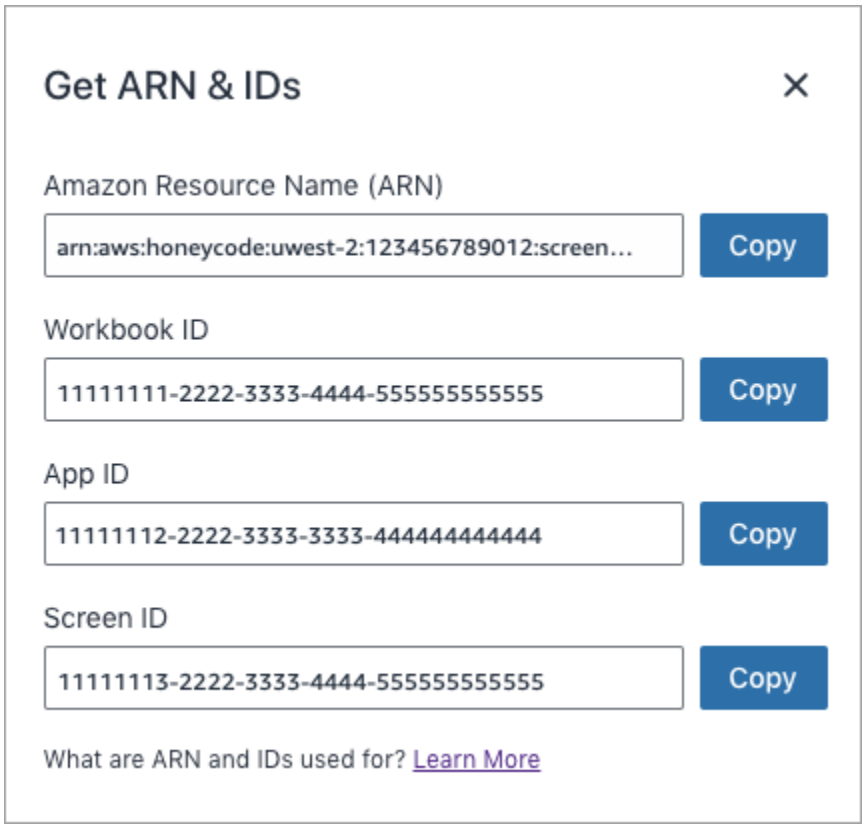
Import APIs require the workbook id as input. You will need an app to get the workbook id. Simply right-click on any app object in builder to access the Get ARN and IDs modal. If you don't have a use case for an app, you can use the App Wizard to quickly create a simple app to grab the Workbook ID and then delete the app as necessary.

Accessing Workbook ID

In any app screen, you can right-click to bring up a menu. Select Get ARN and IDs



From the modal that appears, you can copy the ID for the workbook. You can ignore the other fields on that modal (Amazon Resource Name, App ID, Screen ID) as they are not needed for import APIs.



ARNs and authorization

If you are using the managed IAM policies *AmazonHoneycodeWorkbookFullAccess* or *AmazonHoneycodeFullAccess*, you will not need the ARN for authorization. The ARN is only required to to set up authorization using IAM at a granular level.

The Honeycode resources that need to be defined in custom IAM policies for import APIs are given below. Click the links to get more details on how to construct ARNs for those resources.

- *StartTableDataImportJob*: table
- *DescribeTableDataImportJob*: table

StartTableDataImportJob

The StartTableDataImportJob API starts a table data import job that runs in the background. Once the job is started, you can use the DescribeTableDataImportJob API to find the status of the import.

To find more details about using this API check the StartTableDataImportJob API Reference page.

The following example shows how to import a CSV file to the Stocks table. First, create a CSV file with contents as follows:

```
Symbol,Price,Previous Price,Last Update
AAA,123.17,182.21,11/9/20 6:23 PM
BBB,127.27,128.13,11/9/20 6:23 PM
```

Upload this CSV to an S3 bucket. Use the following AWS CLI command to generate a presigned URL for the CSV. Be sure to replace with the S3 bucket name and with the name of the file you uploaded.

```
aws s3 presign s3://<bucket-name>/<file-name>
```

Copy the presigned URL from the output. This URL will need to be passed in as a parameter in the call to StartTableDataImportJob API. Now the file is ready for import. In the examples below, replace `<workbook-id>` with your workbook id.

AWS CLI Example

```
aws honeycode start-table-data-import-job \
  --workbook-id '<workbook-id>' \
  --table-id '<table-id>' \
  --dataSource '{ "dataSourceConfig": {"dataSourceUrl": "<presigned-url>" } }' \
  --dataFormat 'DELIMITED_TEXT' \
  --import-options '{
    "destinationOptions": {
      "columnMap": {
        "<symbol-column-id>": { "columnIndex": 1 },
        "<price-column-id>": { "columnIndex": 2 },
        "<previous-price-column-id>": { "columnIndex": 3 },
        "<last-update-column-id>": { "columnIndex": 4 }
      }
    },
    "delimitedTextOptions": {
      "delimiter": ",",
      "hasHeaderRow": true,
      "ignoreEmptyRow": true,
      "dataCharacterEncoding": "UTF-8"
    }
  }' \
  --client-request-token '<request-token>'
```

Python SDK Example

```
response = honeycode_client.start_table_data_import_job(
```

```
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    dataSource = { "dataSourceConfig": {"dataSourceUrl": "<presigned-url>" } },
    dataFormat = 'DELIMITED_TEXT',
    importOptions = {
      "destinationOptions": {
        "columnMap": {
          "<symbol-column-id>": { "columnIndex": 1 },
          "<price-column-id>": { "columnIndex": 2 },
          "<previous-price-column-id>": { "columnIndex": 3 },
          "<last-update-column-id>": { "columnIndex": 4 }
        }
      },
      "delimitedTextOptions": {
        "delimiter": ",",
        "hasHeaderRow": true,
        "ignoreEmptyRow": true,
        "dataCharacterEncoding": "UTF-8"
      }
    },
    clientRequestToken = '<request-token>')
```

Response

```
{
  "jobId": "<job-id>",
  "jobStatus": "SUBMITTED"
}
```

DescribeTableDataImportJob

The DescribeTableDataImportJob API describes a table data import job that was started previously.

To find more details about using this API check the DescribeTableDataImportJob API Reference page.

AWS CLI Example

```
aws honeycode describe-table-data-import-job \
  --workbook-id '<workbook-id>'
  --table-id '<table-id>'
  --job-id '<job-id>'
```

Python SDK Example

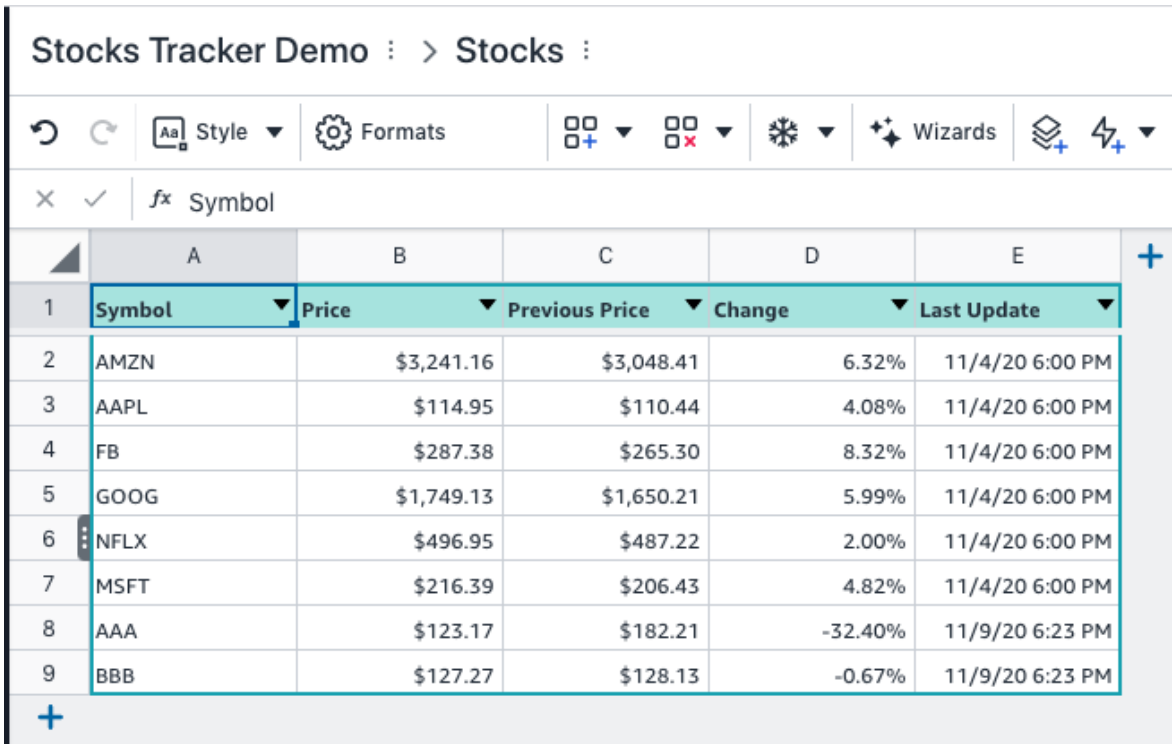
```
response = honeycode_client.describe_table_data_import_job(
    workbookId = '<workbook-id>',
    tableId = '<table-id>',
    jobId = '<job-id>')
```

Response

```
{
    "jobMetadata": {
        "dataSource": {
            "dataSourceConfig": {
                "dataSourceUrl": "<presigned-url>"
```

```
            }
        },
        "importOptions": {
            "delimitedTextOptions": {
                "dataCharacterEncoding": "UTF-8",
                "delimiter": ",",
                "hasHeaderRow": true,
                "ignoreEmptyRows": false
            },
            "destinationOptions": {
                "columnMap": {
                    "<price-column-id>": {
                        "columnIndex": 2
                    },
                    "<previous-price-column-id>": {
                        "columnIndex": 3
                    },
                    "<symbol-column-id>": {
                        "columnIndex": 1
                    },
                    "<last-update-column-id>": {
                        "columnIndex": 4
                    }
                }
            }
        },
        "submitTime": 1.60456803175E9,
        "submitter": {
            "userArn": "<submitter-user-arn>"
        }
    },
    "jobStatus": "COMPLETED"
}
```


After the import job is completed, this is how the table looks.



	A	B	C	D	E
1	Symbol	Price	Previous Price	Change	Last Update
2	AMZN	$3,241.16	$3,048.41	6.32%	11/4/20 6:00 PM
3	AAPL	$114.95	$110.44	4.08%	11/4/20 6:00 PM
4	FB	$287.38	$265.30	8.32%	11/4/20 6:00 PM
5	GOOG	$1,749.13	$1,650.21	5.99%	11/4/20 6:00 PM
6	NFLX	$496.95	$487.22	2.00%	11/4/20 6:00 PM
7	MSFT	$216.39	$206.43	4.82%	11/4/20 6:00 PM
8	AAA	$123.17	$182.21	-32.40%	11/9/20 6:23 PM
9	BBB	$127.27	$128.13	-0.67%	11/9/20 6:23 PM

Logging Amazon Honeycode API Calls with AWS CloudTrail

Amazon Honeycode APIs are integrated with AWS CloudTrail, a service that records all API calls and events for AWS accounts. CloudTrail is enabled when you create an AWS account.

Using the information recorded by CloudTrail, you can identify trends and further isolate activity by attributes, such as what API call was made, when, who made the request, and the IP address.

To learn more about CloudTrail, see the AWS CloudTrail User Guide

Honeycode activity in CloudTrail

When API activity occurs in Amazon Honeycode apps, the activity is recorded in a CloudTrail event. You can view, search, and download recent events in your AWS account.

For an ongoing record of events in Amazon Honeycode, as well as your other AWS accounts, you can create a trail. A trail enables CloudTrail to continuously deliver events as log files to an Amazon S3 bucket.

All Amazon Honeycode API actions are logged by CloudTrail. For example, any calls to the *GetScreenData* or *InvokeScreenAutomation* actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine:

- Whether the request was made with root or AWS IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity Element .

If you don't configure a trail, you can still view the most recent events in the CloudTrail console's event history. For more information, see Viewing Events with CloudTrail Event History .

Honeycode log files on CloudTrail

CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the Amazon Honeycode API calls, so they won't appear in any specific order.

The following example shows a CloudTrail log entry of the *GetScreenData* action.

```
{
    "awsRegion": "us-west-2",
    "eventID": "3b61e597-4bf1-4c17-aac5-70440468f7d9",
```

```
    "eventName": "GetScreenData",
    "eventSource": "honeycode.amazonaws.com",
    "eventTime": "2020-05-21T07:07:39Z",
    "eventType": "AwsApiCall",
    "eventVersion": "1.05",
    "readOnly": true,
    "recipientAccountId": "123456789012",
    "requestID": "73ae2ce0-214b-4dc8-9378-a7a2e2d7aa4e",
    "requestParameters": {
        "appId": "9507a45a-8e7c-4b9b-bdc7-80c29b5ee3e2",
        "maxResults": 10,
        "screenId": "44e50421-8b7c-4074-a6bd-ba5d581ab020",
        "variables": "***",
        "workbookId": "cf8aff9e-3aa3-45e4-b60e-1512a2fa462c"
    },
    "responseElements": null,
    "sourceIPAddress": "12.345.67.890",
    "userAgent": "Jersey/${project.version} (HttpUrlConnection 1.8.0_201)",
    "userIdentity": {
        "accessKeyId": "ACESSKEYIDEXAMPLE12",
        "accountId": "123456789012",
        "arn": "arn:aws:sts::123456789012:assumed-role/honeycode-full-access/
HoneycodeTests-cf9c31ee-dcfd-439c-9ba0-8bb68766bcfe",
        "principalId": "PRINCIPALIDEXAMPLE1234:HoneycodeTests-cf9c31ee-
dcfd-439c-9ba0-8bb68766bcfe",
        "sessionContext": {
            "attributes": {
                "creationDate": "2020-05-21T07:07:39Z",
                "mfaAuthenticated": "false"
            },
            "sessionIssuer": {
                "accountId": "123456789012",
                "arn": "arn:aws:iam::123456789012:role/honeycode-full-access",
                "principalId": "PRINCIPALIDEXAMPLE1234",
                "type": "Role",
                "userName": "honeycode-full-access"
            },
            "webIdFederationData": {}
        },
        "type": "AssumedRole"
    }
}
```


Similarly, the following example shows a CloudTrail log entry of the *InvokeScreenAutomation* action.


```
{
    "awsRegion": "us-west-2",
    "eventID": "30c82beb-4d38-41ef-9dd2-961ed827412a",
    "eventName": "InvokeScreenAutomation",
    "eventSource": "honeycode.amazonaws.com",
    "eventTime": "2020-05-21T07:07:29Z",
    "eventType": "AwsApiCall",
    "eventVersion": "1.05",
    "readOnly": false,
    "recipientAccountId": "123456789012",
    "requestID": "18e22c8a-495c-4c0f-b3d3-e308541baef5",
    "requestParameters": {
        "appId": "5c132f99-d482-45be-b4f5-6deaf8067d0a",
        "automationId": "124bb3c9-8ab3-4d39-b380-6a43b63dc666",
        "clientRequestToken": "c5f201b9-76ed-4329-bb46-d4a6cc4fc638",
        "rowId": "row:6655a2f2-1e70-45a9-86ec-4d3c63d443b6/f9b70edb-486a-36b4-
b72b-89df6f92be44",
        "screenAutomationId": "124bb3c9-8ab3-4d39-b380-6a43b63dc666",
```

```
        "screenId": "d2d4b6c6-c5e4-45fc-b342-019132ffb4f8",
        "variables": "***",
        "workbookId": "aa34dd68-2077-440e-abca-470deef13e9b"
    },
    "responseElements": {
        "workbookCursor": 815985817
    },
    "sourceIPAddress": "54.244.61.237",
    "userAgent": "Jersey/${project.version} (HttpUrlConnection 1.8.0_201)",
    "userIdentity": {
        "accessKeyId": "ACESSKEYIDEXAMPLE12",
        "accountId": "123456789012",
        "arn": "arn:aws:sts::123456789012:assumed-role/honeycode-full-access/
HoneycodeTests-3941fe61-25ee-4df1-ba85-411bb7e01472",
        "principalId": "PRINCIPALIDEXAMPLE1234:HoneycodeTests-3941fe61-25ee-4df1-
ba85-411bb7e01472",
        "sessionContext": {
            "attributes": {
                "creationDate": "2020-05-21T07:07:26Z",
                "mfaAuthenticated": "false"
            },
            "sessionIssuer": {
                "accountId": "123456789012",
                "arn": "arn:aws:iam::123456789012:role/honeycode-full-access",
                "principalId": "PRINCIPALIDEXAMPLE1234",
                "type": "Role",
                "userName": "honeycode-full-access"
            },
            "webIdFederationData": {}
        },
        "type": "AssumedRole"
    }
}
```


More AWS CloudTrail resources

Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

FAQs

Topics

How many transactions can the APIs handle per second?

Throttling will be per AWS Account. Currently, these are the limits (subject to change):

- GetScreenData: 25 tps rate/250 tps burst
- InvokeScreenAutomation: 5 tps rate/50 tps burst
- StartTableDataImportJob: 1 tps rate/5 tps burst
- DescribeTableDataImportJob: 5 tps rate/25 tps burst
- ListWorkbooks, ListTables, ListTableColumns, QueryTableRows and ListTableRows: 25 tps rate/50 tps burst
- BatchCreateTableRows, BatchDeleteTableRows, BatchUpdateTableRows and BatchUpsertTableRows: 5 tps rate/25 tps burst

Can I trigger code elsewhere based on an event in Honeycode (e.g. a button being clicked on in a Honeycode app or a row being added to a Honeycode table)?

Yes, you can use webhooks. Please see this article.

Can I use system variables like SYS_USER, conditional visibility, or personalization in screens I intend to use with Honeycode APIs?

No. These concepts are meant to control the app experience for users. However, since there is no concept of logged-in user when interacting with apps programmatically, they will not be expressed when screens are read from the APIs.

Can I get a history of Honeycode API calls made on my account for security analysis and operational troubleshooting purposes?

Yes. To receive a history of Honeycode API calls made on your account, simply turn on CloudTrail in the AWS Management Console. The AWS API call history produced by CloudTrail enables security analysis, resource change tracking, and compliance auditing. Learn more about CloudTrail here.

What is the size limit of the file that can be imported?

The file size should be less than 100 MB. The file that you want to import has to be a delimited text file containing one table, where each line represents a row, and fields separated by a delimiter such as a comma. It can contain up to 1000 rows, including headers. The file can have any number of columns; however, a maximum of 99 columns will be imported by the API. If there are more than 99 columns, it will import the first 99 columns. Alternatively, a column mapping can also be specified to import particular columns.

What file types can I import into Honeycode?

The file that you want to import has to be a delimited text file containing one table, where each line represents a row, and fields separated by a delimiter such as comma, tab or a pipe. The delimiter can be specified at the time of import. Support for other file types is not present at the moment. You could try converting your data to a comma separated (CSV) or tab separated (TSV) format to import into Honeycode.

Would existing automations work on the new data?

Yes. If the table you are importing data into has some automations associated with it, they will continue to trigger for the new data.

Can I import data with emails and rowlink selections?

Yes. As a precursor, the target Honeycode table column must be formatted as Contact/Rowlink before importing data. Emails of team members who belong to the org in which the workbook is created will get auto-formatted as a contact. For emails that don't, the data will still be imported. However, the email will show as plain text and not contact formatted. Rowlinks will be detected from the input data, as long as the values match the display value of corresponding rowlink.

Can I control the column mapping from my data to the Honeycode table?

Yes. You can specify column mapping in `importOptions`. For details, check the API Reference page for StartTableDataImportJob here .

Can I import from more than one file for a table?

You can import data from only one file into one table in a single API call. However, you can make multiple calls to the import API sequentially for each file. Please note that your billing tier limits will still apply. For instance, if you are in the free tier, you can only have 2500 rows in the table, and import will honor the same limits.

What if I want to import more than 1000 rows?

With one operation, you can only import 1000 rows. However, you can break the file into multiple files, each containing 1000 rows and call import sequentially for each part. Please note that your billing tier limits will still apply. For instance, if you are in the free tier, you can only have 2500 rows in the table, and import will honor the same limits.

Document History for User Guide

The following table describes the documentation for this release of Amazon Honeycode.

- **API version: latest**
- **Latest documentation update:** Dec 1, 2020

update-history-change	update-history-description	update-history-date
New APIs to interact with Honeycode tables (p. 55)	Honeycode introduced APIs to read and write directly from tables. Use APIs to pull table and column metadata, then use the read and write APIs to programmatically read and write from the tables. For more information, see Interacting with Honeycode workbooks via SDK	December 1, 2020
Initial Version (p. 55)	Initial version of the documentation for Amazon Honeycode	June 24, 2020

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.