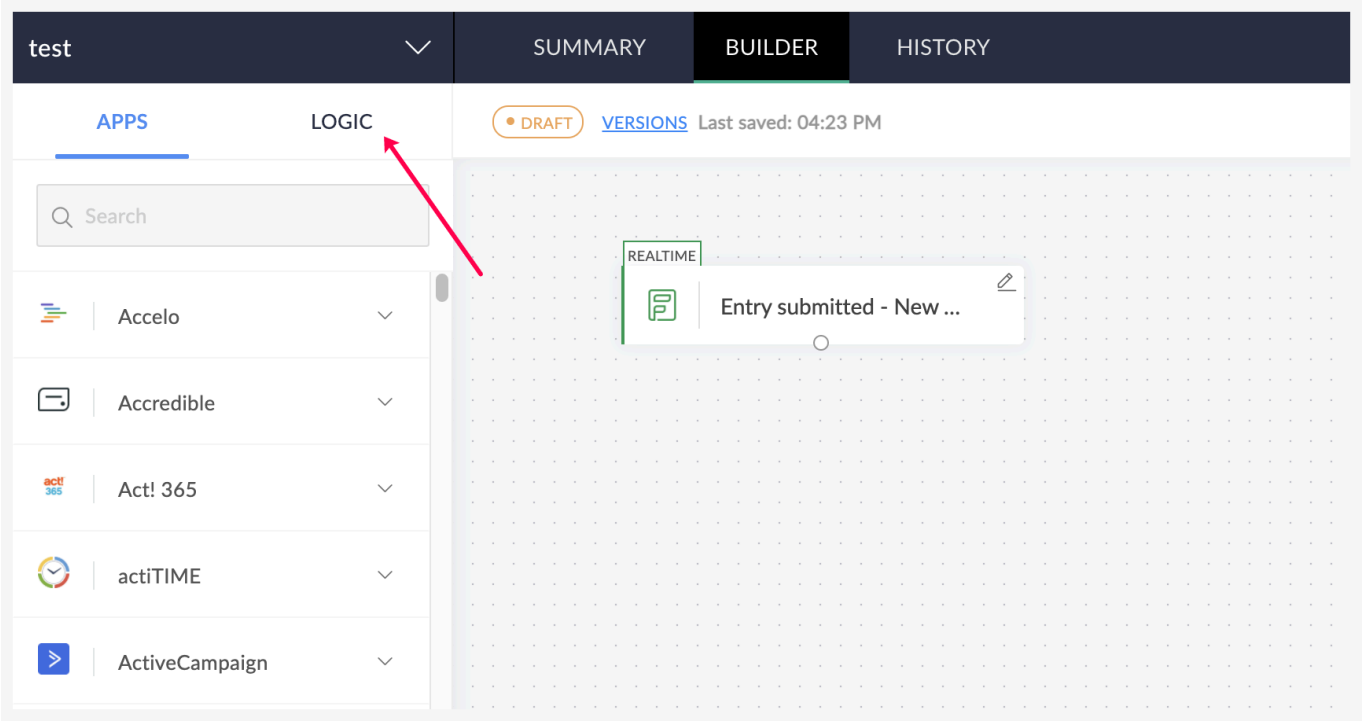
Custom functions

On Zoho Flow, you can write your own Deluge code to create custom functions to make your workflows more powerful. These functions provide flexibility to achieve complex workflows that simplify repetitive tasks. For example, you can create a custom function that calculates the discount rate for your invoice, or a function to analyze your support tickets and alert you if there is a negative statement.

Custom functions can satisfy business-specific requirements by letting you write the entire function from scratch. Once created, they can be used by all members in your organization.

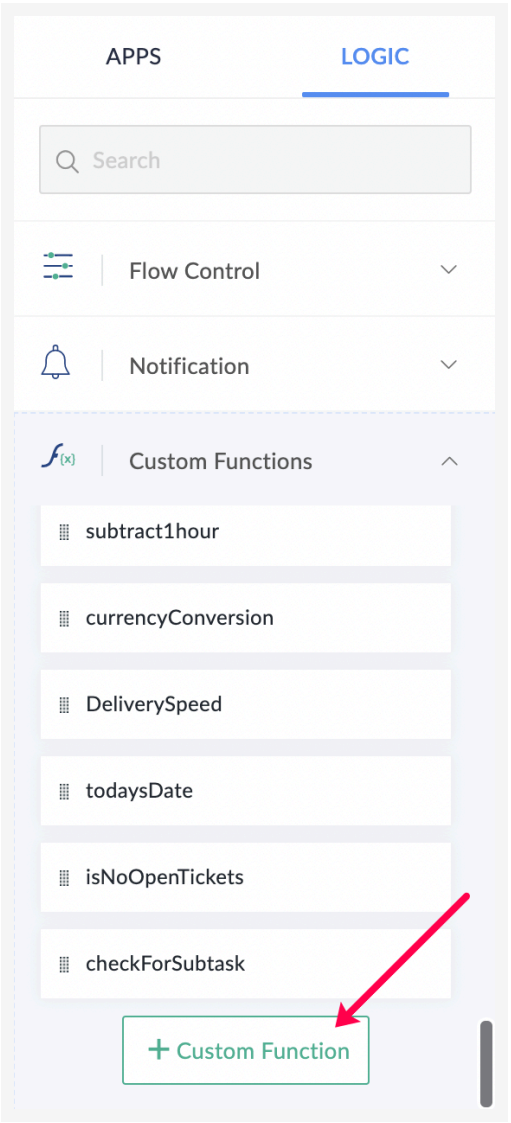
Create a custom function

1. Click the **Logic** tab on the left side of the builder.



2. Click **Custom Functions**.

3. Click **+Custom Function**.



4. Enter a name. Remember that the name must start with a letter or underscore and can only include alphanumeric characters and underscores.
   E.g: _discount_calculation_1, autofill_zipcode

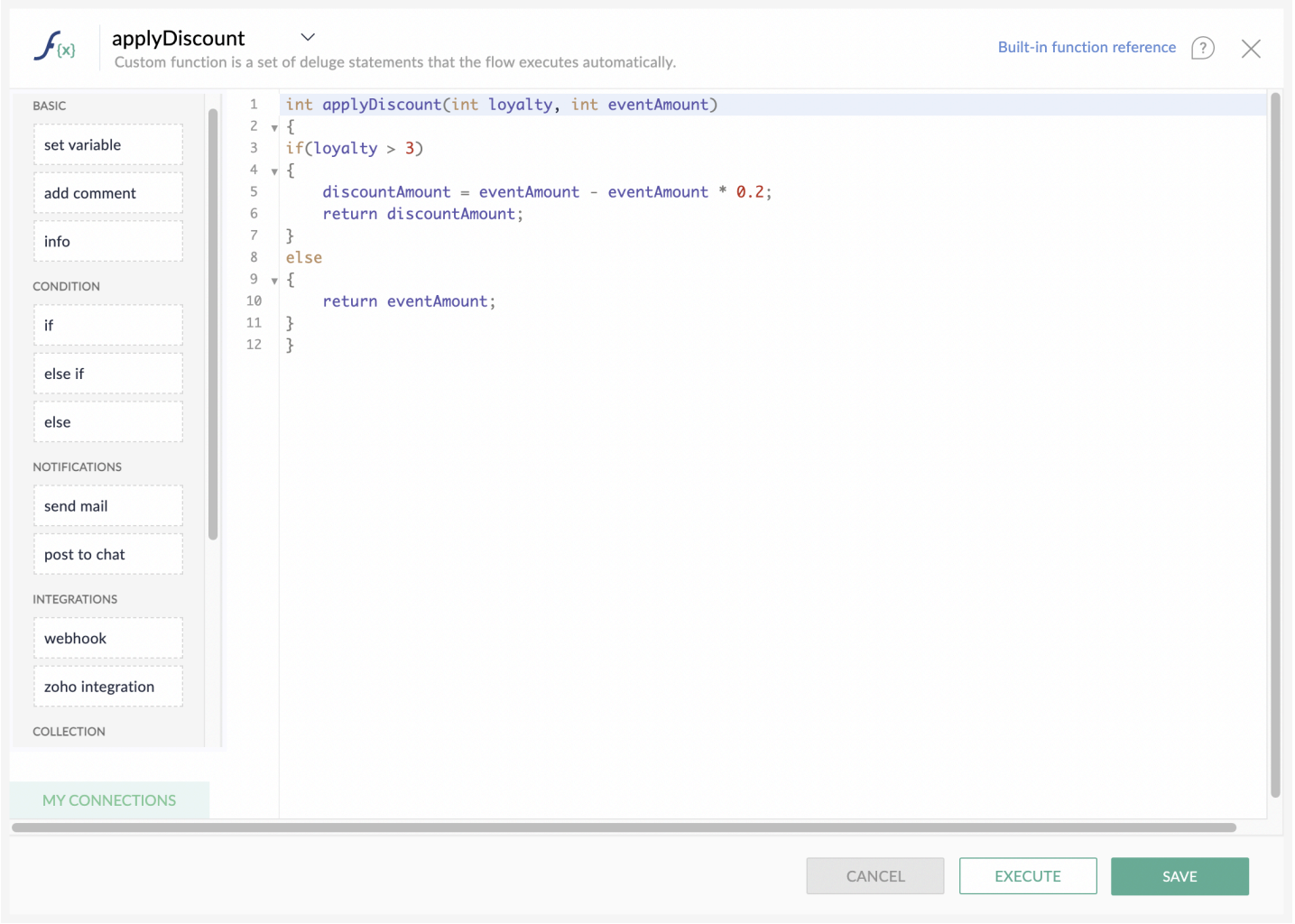
5. Select a return type (output data type) and optionally specify the parameter (input) and its data type.

Data Type	Description
void	No value
int	Integer value
float	Decimal value
string	Text
bool	Either true or false
date	Date value
map	Key-value mapping
list	List of values
file	File object

> **Note:**
> • The return type will be void by default.
> • The input data type cannot be void.
> • FILE is a unique data type that treats all files fetched from the web or cloud services (irrespective of file format) as file objects. Refer to Deluge documentation to learn more.

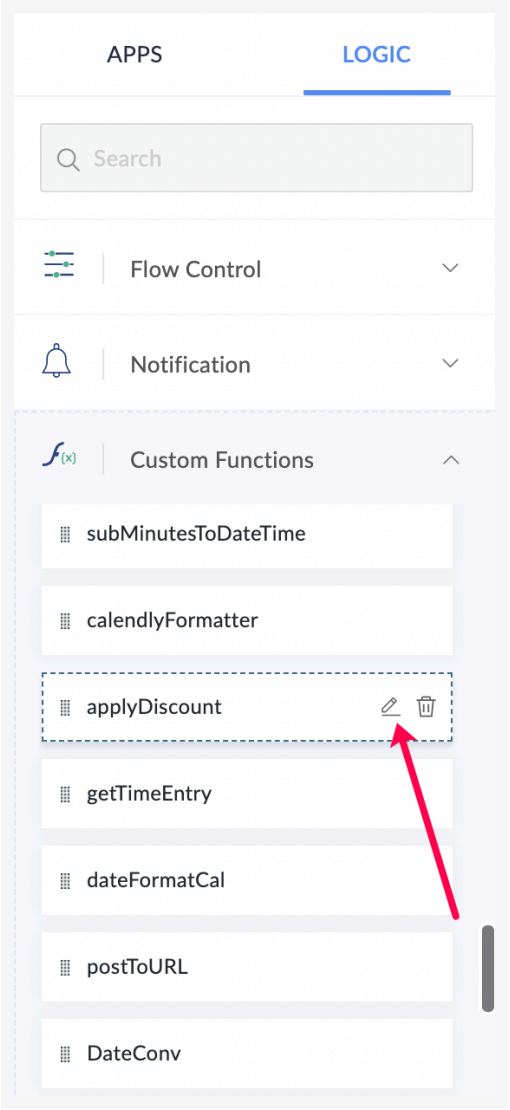
6. Click **Create**.

7. Write the code for your custom function. Refer to Deluge documentation to learn more.



8. Click **Save**.

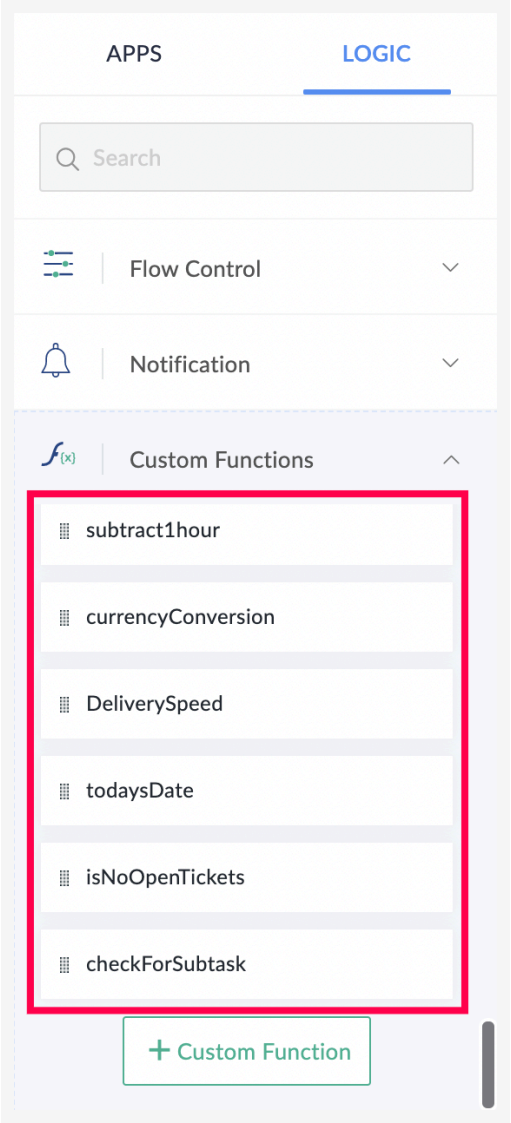
9. To modify the custom function code, click the edit icon on the function. Any member in the organization can edit or delete a custom function in the organization.



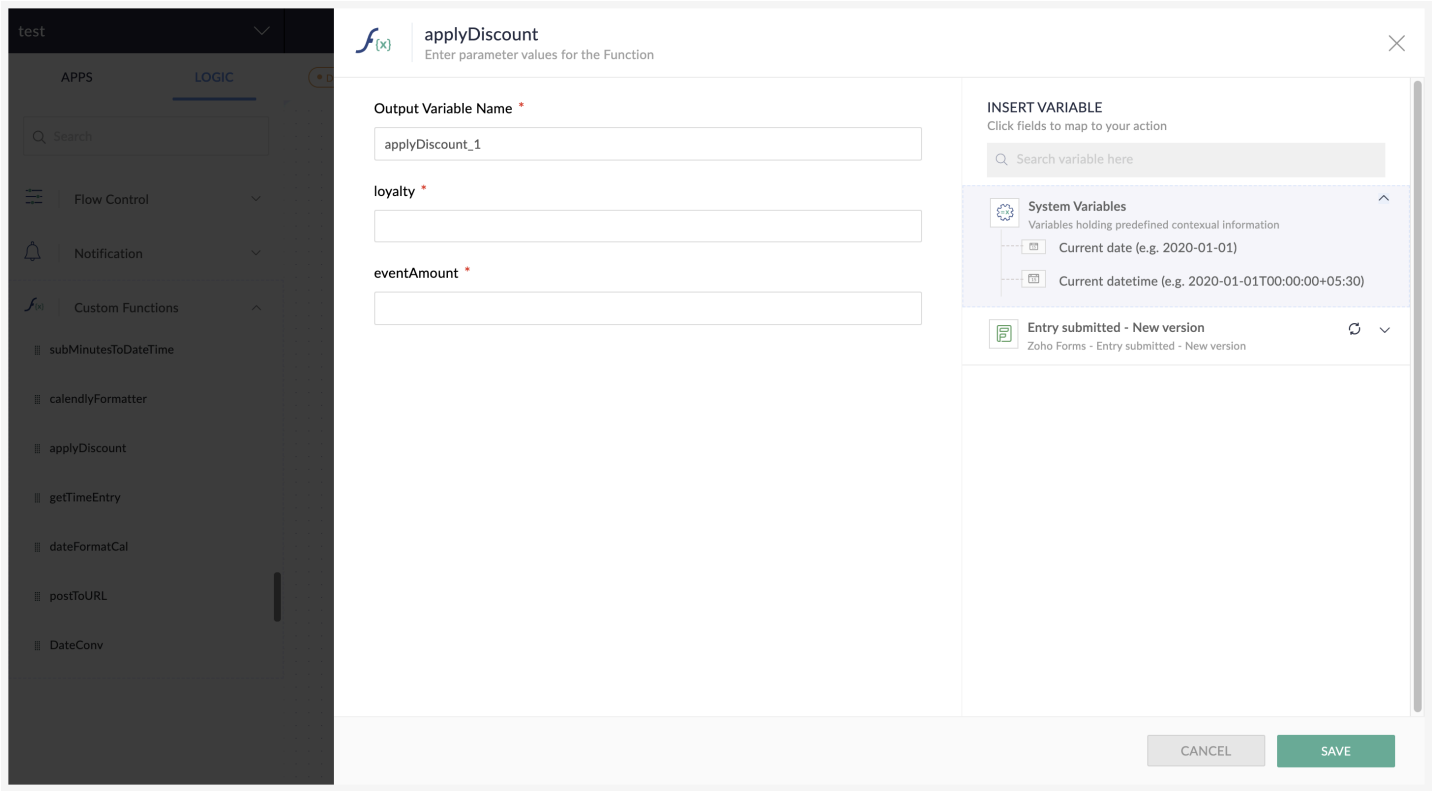
The created function will now be available under **Custom Functions** in the **Logic** tab.

Use an existing custom function

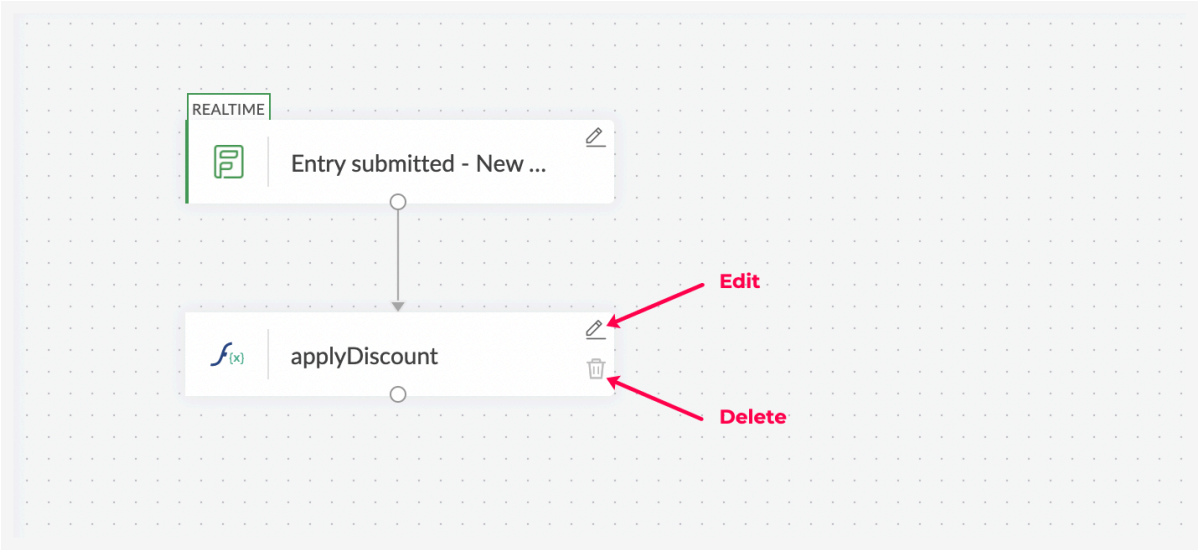
1. Click the **Logic** tab on the left side of the builder.
2. Click **Custom Functions**. You can view the list of existing custom functions in your organization.



3. Drag and drop the function you want to use to the builder screen. A configuration window will open.



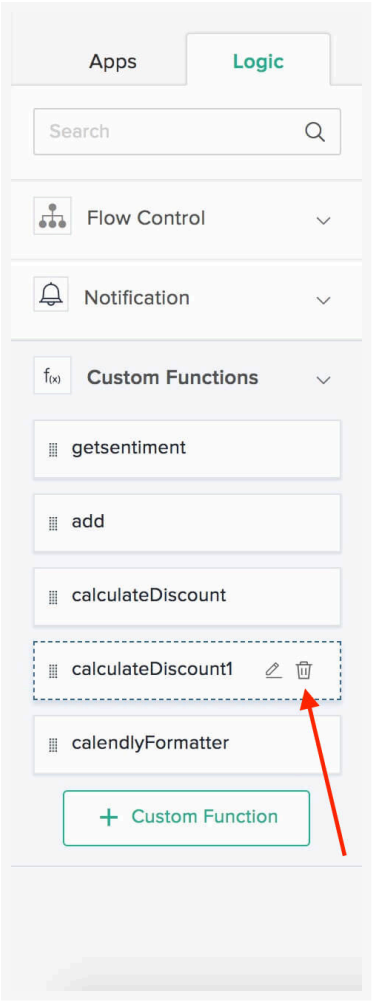
4. Configure the function by entering the input parameter(s) of the function.
5. Click **Save**.
6. You can edit the custom function data you configure by clicking the edit icon on the right side of the custom function. You can also delete it like other actions.



Delete a custom function

1. Click the **Logic** tab on the left side of the builder.
2. Click **Custom Functions**. You can view the list of existing custom functions in your organization.

3. On mouse over, the delete icon will appear. Click on it to delete the function.



If a custom function is deleted, all the flows using the function will be affected.

Map a custom function's output variable

The output of a custom function will be available as a variable in the **Insert Variables** section. You can just map this variable to an action's fields the same way you map other variables.

> If you are using a custom function that returns key-value pairs (map datatype), you can map individual keys separately by executing the custom function with sample data. Learn more

Using app connections in your functions

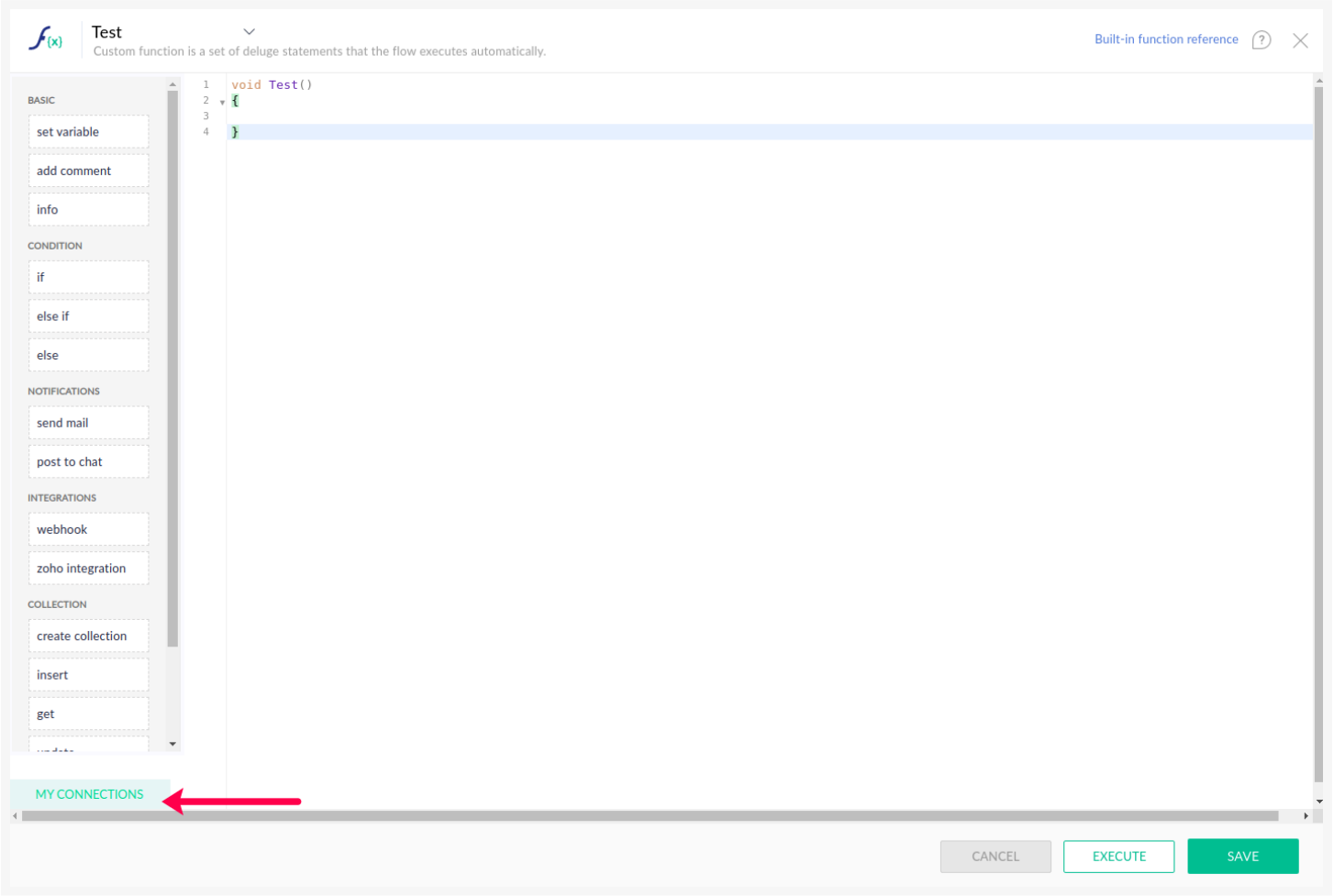
Zoho Flow allows you to use your app connections in custom functions by using the **invokeURL** task. This helps you to establish authentication with Zoho or third-party services to access data and integrate with them when

using a function.

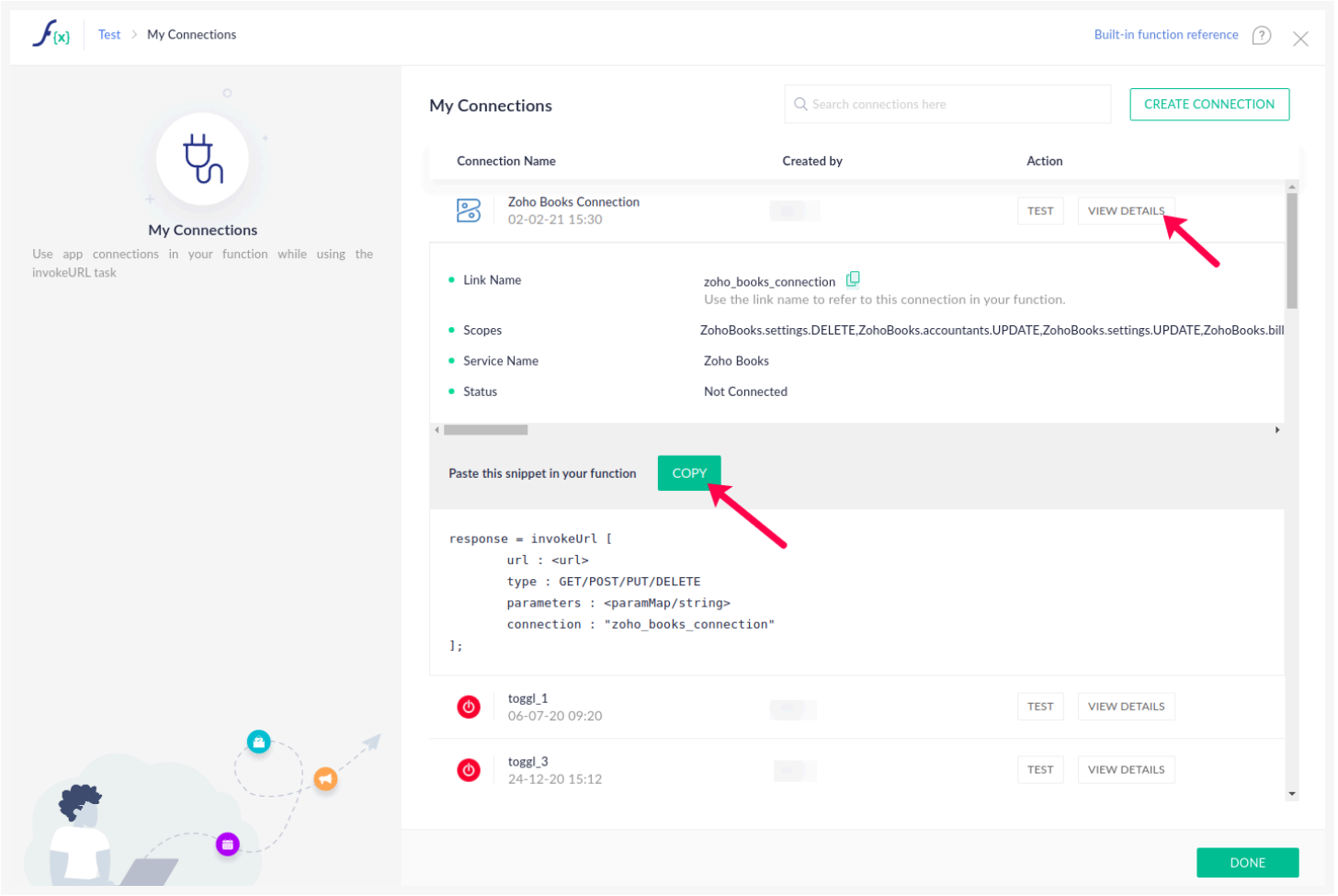
Learn more about the invokeURL task.

How to embed your app connections in an invokeURL task

1. Click **My Connections** on the left of the custom function console.



2. Find the app connection that you require from the list, then click **View Details**.
   If the connection doesn't already exist, you can create a new app connection by clicking **Create Connection**.



3. Copy the code snippet and paste it into your function to embed this connection.

4. You can also copy the *Connection link name* from the same box.

Note:

• **Connection link names** are unique names given to an app connection that can be used in invokeURL scripts to refer to a connection and authorize the connection with that app.

• Connection support is only currently available for the invokeURL task. We are working on supporting predefined integration tasks.

Important note for Zoho app connections

If you are already using any authtoken-based Zoho app connections in your invokeURL tasks, it is important that you modify them to OAuth-based connections immediately. All Zoho apps are migrating from authtoken-based authorization to OAuth-based authorization and existing authtoken-based connections will no longer work.

To update the connections:

1. Identify the Zoho app connection that you have embedded in your custom function using the invokeURL task.
2. Create a new OAuth-based connection for your Zoho app.
3. In the required invokeURL task, remove the authtoken parameter and use the new connection. Follow the steps in the previous section to embed the new connection in your existing invokeURL task.

For example, in the following function, an authtoken is used to establish a connection to Zoho Books:

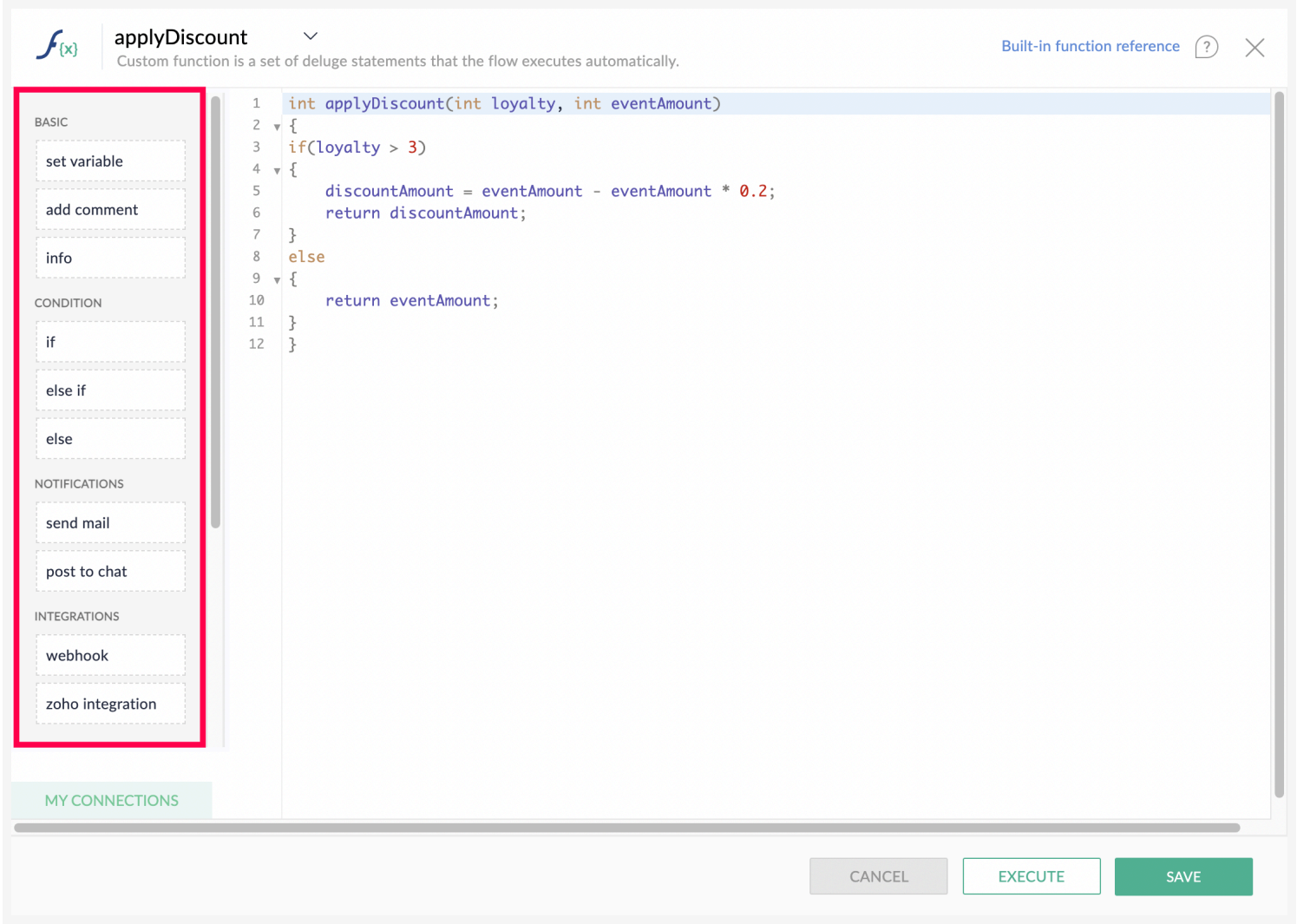
```
map getItemFromZohoBooks(string Authtoken, string BooksOrgId, string itemName)
{
url = "https://books.zoho.com/api/v3/items";
params = Map();
params.put("authtoken",Authtoken);
params.put("organization_id",BooksOrgId);
params.put("name",itemName);
item = invokeurl
[
    url :url
    type :GET
    parameters:params
];
return item;
}
```

After creating an OAuth connection for Zoho Books, the script can be modified to embed the newly created connection.

```
map getItemFromZohoBooks(string BooksOrgId, string itemName)
{
url = "https://books.zoho.com/api/v3/items";
params = Map();
params.put("organization_id",BooksOrgId);
params.put("name",itemName);
item = invokeurl
[
    url :url
    type :GET
    parameters:params
    connection: "zoho_books_connection"
];
return item;
}
```

Deluge statements and tasks

These are blocks of code available on the left of the custom function coding screen. Drag and drop them onto the coding area and enter the required data.



[Basic]

[Condition]

[Notifications]

[Integrations]

[Collection]

Basic

Set Variable

Creates a variable with the given value that can be accessed within the action

For example:

```
price = quantity*20
```

Every time the quantity value is changed, the value of the price will be modified

Add Comment

Adds comment to make your code understandable by others

For example:

```
price = quantity*20 //Multiplies the quantity of the product order by 20 to calculate the total price
```

Any data that is after '//' will be considered a comment

info

Prints the value of specified parameters as the function output in the history log

For example:

```
info customer_names;
```

This prints the customer names in the history log

Condition

if

Checks for a condition. If the condition is true, it performs the specified action.

For example:

```
if (client_title == "CEO")
{
client_type = "premium";
}
```

This checks if the client title is CEO. If it is true, the client type is set to premium.

else if

Executes when the previous *if* statement is false and this statement is true

For example:

```
if (client_title == "CEO")
{
client_type = "premium";
}
else if (client_title == "Admin")
{
client_type = "standard";
}
```

When the client title is not CEO, but is Admin, the client type is set to standard

else

Executes when both *if* and *else if* statements fail

For example:

```
if (client_title == "CEO")
{
client_type = "premium";
}
else if (client_title == "Admin")
{
client_type = "standard";
}
else
{
client_type = "regular";
```

```
}
```

When neither *if* nor *else if* conditions hold true, the client type is set to regular.

Notifications

Send mail

Sends an email to the specified recipients

For example:

```
sendmail
[
from: zoho.adminuserid
to: "bruce@zylker.com"
subject: "Your request has been approved"
message:
"Hello Bruce,
Your request for a new laptop has been approved. Please contact your IT administrator to collect it.
Regards,
Frank Wilson"
]
```

Integrations

webhook

Creates a webhook subscription for another application

```
param = Collection("TestParam":"TestValue");
header = Collection("Content-type":"application/json");
testWebhook = invokeurl
[
url :"https://requestb.in/1ckt5a31%22"
```

```
6. type: POST
7. parameters: param
8. headers: header
9. ];
```

Learn more

Collection

create collection

Creates a map or a list depending on the input elements specified

For example:

```
1. students = Collection ();
```

insert

Adds elements to the specified collection

For example:

```
1. students.insert ("Kevin", "Jessica", "William", "Emma");
```

Adds the given names to the students list

For example:

```
1. students. insert ("name":"Matt" , "grade":"8" , "subject":"English");
```

Adds the given key-value pairs to the map

get

Fetches a particular element from the specified collection

For example:

```
1. students.get (6);
```

Fetches the value at index 6 from the students list

For example:

```
1. students.get ("name");
```

Fetches the value mapped with the key 'name' in the students map

update

Updates the specified collection

For example:

```
1. students.update(2,"Micheal");
```

Updates the value in the second element of the list to 'Micheal'

For example:

```
1. students.update("grade","9");
```

Updates the value in the key 'grade' to 9

for each element

Performs the task for each element in the specified collection

For example:

```
1. for each student_mail in students
2. {
3. myMessage = "Welcome to the students sports club. Please assemble at 9:00 am in the basketball court tomorrow.";
4. sendmail
5. [
```

```
from :zoho.loginuserid
to :student_mail
subject :"Invitation to Sports club"
message :myMessage
]
}
```

For every student email address available in the list, the mail will be sent

For example:

```
name = collection();
name.insert("First Name":"Emma", "Middle Name":"Marley", "Last Name":"Becker");
Fullname = "";
for each element in name
{
Fullname += (element+" ");
}
```

This function concatenates every value in the map. The output value of this function will be 'Emma Marley Becker'.

Note: Date-time fields must be converted into string functions before being used in custom functions or data mapping.

Sample code:

This helps transform one date format into another and converts it into a string function.

```
string formatDate(string myDate, string fromFormat, string toFormat)
{
fromDate = myDate.toTime(fromFormat);
dateStr = fromDate.toString(toFormat);
return dateStr;
}
```

Create a custom function with the above code. Specify the following input parameters:

- myDate - Date field from the previous step
- fromFormat - Date format of the previous step
- toFormat - Date format of the next step

For example, if you are creating this custom function between a MailChimp trigger and a Zoho Creator action, map myDate to the date field from MailChimp. fromFormat will be MailChimp's date format and toFormat will be Zoho Creator's date format.

Useful applications of Deluge tasks and built-in functions

Deluge tasks and built-in functions can help you customize your functions to serve specific requirements in your workflows. The following list provides you with some possibilities to explore:

HTTP or API calls

To access and modify web resources, you will need to make HTTP or API calls. In such cases, you can use the invokeURL task which is an HTTP client.

Files and attachments

File functions allow you to operate on file objects (FILE datatype). They can help you get the file's properties, zip or unzip files, and more.

Predefined integrations with Zoho services

Integration tasks help you access, transfer, and synchronize data across various Zoho services. In case you want to include these integration tasks in your function, you can use this list of integration tasks.

Date and time

You can work with date and time formats, perform calculations, and more using datetime and time functions.

Numbers and calculations

If you need to work with advanced calculations or mathematical operations, you can use number functions in your code.

Map and List operations

Map functions help you manipulate key-value data in a variable. For example, it lets you show the keys in a variable, remove key-value pairs, and more.

For each record

The for each record task iterates through a form's records, which can be based on specific criteria.

Conversions

Conversion functions help you convert data into your required data type or structure.

For more information related to Deluge functions and tasks, you can browse through Deluge's help documentation.