Training Material:

Building a vertical application using

Developer

www.zoho.com/developer

ZOHO

CONTENTS

Introduction to Training

Welcome to the training guide of the Zoho Developer cloud platform.
In this guide, we'll see how you can build your business applications on our platform without facing technical complexities in coding, hosting, and operating your application.

Zoho Developer Console

Zoho Developer is a cloud platform and development console, which lets you easily, build, brand, and sell robust industry-specific applications on top of Zoho's existing cloud infrastructure. You can also use the console to create functionalities that can be made available to users of Zoho's suite of cloud products.

Getting started with development:

The Zoho Developer platform saves your developers time and effort by taking care of low-level system details while providing them with multiple prebuilt advanced features that can become a part of your cloud application. Any app you build on the console will store and work with data that is particular to your choice of industry. For example, if you build an app for real estate, it will have the power to work with real estate data like property information, documents, handover processes, inventory, etc.

In this guide, we will cover each aspect of developing on the Zoho Developer Console by comparing with a sample CRM for Garages - Garage CRM.

About Garage CRM:

Garage CRM is an application for garage owners to automate all their service processes. It helps them to manage information about customers and vehicles, schedule maintenance jobs, manage their field service operations, handle emergency maintenance jobs, generate and share paperless job cards and bills for the service done, maintain vehicle service histories, and manage inventory of spares.

Defining your data entities

Before you head into the console and start to develop your application, it is essential for you to clearly define the data entities that form a part of your app. Think of your application as a database which works with different types of data present in separate tables. You need to define what each column contains, what type of action can be performed on the data in these tables, and what sort of relationships exist between multiple tables.

For example, here's how data entities are defined in our Garage CRM:
Garage CRM was built to serve the requirements of various types of automotive garages like single authorized, multi-brand authorized and independent garages. Before starting to build the application, a list of features that we needed was compiled.

This list included:

- Digitized job cards
- Inventory management for spare parts
- Generating bills and invoices
- Maintaining a database of customers and their vehicles
- Assigning technicians to each job order
- Reaching customers through SMS and email

Each of these functions require data entities to be handled by the Garage CRM, these entities include:

- Customers
- Appointments
- Repair Bills
- Cases
- Vehicles
- Job Cards
- Technicians

These data entities form the backbone of the application. While creating a Garage CRM on our console, you shouldn't be burdened with database programming to store and retrieve

information about each entity. Any feature or use case for your application can be engineered once you have these entities covered.

For example, here's how you can handle incoming service requests via Garage CRM:

There are three types of service requests usually handled by garages - Scheduled appointments, unscheduled services, and emergency services.

Scheduled appointments:
Garage CRM stores information on all appointments made by customers. Appointment information can either be sourced from calls received by the garage's call center or taken automatically from any request submitted via a webform. Each appointment has information about the type of service required and it is associated to customer and vehicle details present in the application.

Unscheduled services:
When a customer approaches a garage without having set an appointment, a new customer record is created in Garage CRM in case it is a first time visit. A job card is created for the service required and a service technician is assigned to the job.

Emergency services:
In case of emergency services like breakdowns or collisions, a field service agent is assigned to the location of the emergency after getting the exact location from the customer. This assignment can be automated in Garage CRM by using a dispatcher functionality to pick the nearest available service operator and assign them to the point t=of emergency.

Defining your data entities on the console:

You can map each data entity to Zoho Developer's most basic component - Modules.

Modules act as containers and store all the data entities that are part of your application. Though there are underlying databases for every app you create, Zoho Developer lets you handle the front-end requirement of consolidating similar information into actionable datasets. Your application will have a number of tabs lined horizontally at the top of the interface. These tabs are the modules created to input and store data. Modules can by connected to each other by means of creating 'relationships'. A record present inside one module can be related to one in a different module and this relationship can be tracked in your application.

For example, every vehicle that is stored in Garage CRM under the 'Vehicles' module needs to be associated to the respective vehicle owner under the 'Customers' module to understand which customer owns the vehicle being serviced, and the number of vehicles owned by the customer. We created a component known as a 'Lookup Field' under the 'Vehicles' module, to associate a vehicle to a vehicle owner record present in the 'Customers' module.

Every customer record will have a component known as Related List to display all vehicles owned by that particular customer. We'll see more about these components as we go further in this guide.

For example, Garage CRM uses a dedicated 'Customers' module to get all customer information into the app. We customized the prebuilt 'Contacts' module to handle customer data as it matched out requirements. No prebuilt modules could be used for creating a 'Vehicles' module to get customer vehicle data, so we built it from scratch.

We were able to decide that the 'Contacts' module closely matched what we need to be stored in Garage CRM's 'Contacts' module because we had clearly described what we needed from that particular data entity, before heading into app development.

The following is a list of all modules we created for Garage CRM; You'll read more about the 'Profile Permissions' section in the sections to come.

MODULE NAME	NATURE OF THE MODULE	PROFILE PERMISSIONS
Customers	Renamed from Contacts	Admin, Service Advisor
Vehicles	Custom module	Admin, Service Advisor, Technician
Job Cards	Custom module	Admin, Service Advisor, Technician
Repair Bills	Renamed from Invoices	Admin, Service Advisor
Items	Renamed from Products	Admin, Service Advisor
Purchase Orders	Prebuilt Module	Admin, Service Advisor
Vendors	Prebuilt Module	Admin, Service Advisor
Reports	Prebuilt Module	Admin, Service Advisor
Dashboards	Prebuilt Module	All profiles
Customer Rating	Custom module	Admin, Service Advisor
OBD Alarm	Custom module	Admin, Service Advisor
Service Check List	Custom module	Admin, Service Advisor
Service Tasks	Custom module	Admin, Service Advisor, Technician
Case	Custom module	Admin, Service Advisor, Technician

Do it yourself - Create your first app

Now that you have been introduced to Zoho Developer's modules, its time for you to create your first app.

1. Login to developer.zoho.com
2. Select the Access Console option under '**CRM for Verticals**' in your console home page.
3. Click on the **Create App button** and enter the following details for your sample application.
   - Application Name: Sample Garage App
   - Application URL: samplegarageapp.zohoplatform.com
   - Category: Automotive
   - Short Description: My first test application.
4. Uncheck every other box apart from Leads, Contacts, and Products. These will be your choice of prebuilt modules to start with, this is not a permanent selection as you can add more modules anytime during development.
5. You have successfully started your first application! Head to the 'Company Details' section in the menu on your left and update fiscal year, business hours, currency, and location information of your fictional organization.

Describing each data entity

While deciding which data entities are going to be included in your application and assigning modules for each one, you need to understand how a module works for your application. Data in a module is stored as a collection of records displayed to users in a list view. Inside a module. each record in the list view consists of a details page which receives and stores information through custom fields. Users can add data to each record by populating custom fields present in the details page.

In order for you to decide how your record page's layout is to look like, and what custom fields are going to be a part of your application, you need to clearly describe your data entities and define what all information is required to constitute one record.

For example, every customer record in Garage CRM will have information on the service advisor assigned to the customer, the customer's contact information, address details, vehicles owned by the customer, and job cards under their name.

All of this information is sourced using components known as custom fields. There are two types of custom fields that can become a part of your app - Input fields and lookup fields.

INPUT FIELDS	LOOKUP FIELDS
Text, integers, check boxes, multi-select, pick-lists, date and time, auto-number, and formula fields.	One-to-one lookup fields, one-to-many lookup fields.

Input Fields:
Input fields serve the purpose of getting various types of information into your records. Zoho Developer comes with a drag-and-drop page builder that you can use to add any sort of input field into your record page's layout. You don't have to create input validations for any field you add as the platform takes care of all validations necessary for your app. You can also define the default input value to be present on any field previous to input by the user. The platform also facilitates creating inter-dependent pick lists that make user input easier.

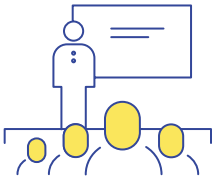
For example, the records page in Garage CRM's Vehicles module has two pick lists named 'Car Brand' and 'Car Model' respectively.

When the user selects a particular car brand, say Ford, from the pick list, the second list is populated with only Ford's car brands for the user to choose from. This inter-dependent pick list method saves users a lot of time in selecting the right value and also offers a clean user experience.

Lookup fields:
Unlike input fields, lookup fields aren't meant to get and store data from your application's users. Instead, they act as fields that track relationships between records present in different modules. Lookup fields are present alongside the input fields in your page layout so that users can 'look up' the relationship that record shares with other records.

There are two types of relationships which can be mapped into lookup fields:

1. **One-to-one lookups:**
These lookups capture the relationship shared by one record to a single other record. This means a record cannot be associated to more than one relationship. For example, a one-to-one lookup is used to associate a job card with one invoice. As it doesn't make sense to have multiple invoices for the same job card.



2. **One-to-many lookups**:
They are used to capture the relationship between one record and multiple other records present in another module. For example, a customer record in Garage CRM is associated to multiple records of vehicles using one-to-many lookups, if that customer is the owner of multiple vehicles that have been serviced in the garage.

One-to-many lookups only associate a record with many records that are present in the same module as we cannot associate a vehicle to an owner, a job card, and an invoice, using the same field.

When multiple vehicles are associated to a customer using lookups, these relationships are displayed inside the customer's record page in Garage CRM using a component known as Related List.

The table below lists all the custom fields that have been created in each module of Garage CRM:

MODULES	CUSTOM FIELDS	TYPE
Customers - general section	Person's Name	Single Line
	Customer GSTN	Single Line
	Country of Birth	Single Line
	Company	Single Line
	Job Title	Single Line
FCustomers - Address Information	Country	Dropdown Menu
	State	Dropdown Menu
	City	Dropdown Menu
	Street Address	Single Line
Vehicles - general section	Vehicle Contact	Look up
	Color	Dropdown Menu
	Registration Number	Single Line
	Manufacturer	Dropdown Menu
	Engine Number	Single Line
	Car Model	Dropdown Menu
	Chassis Number	Single Line

	Fuel Type	Dropdown Menu
	Transmission	Dropdown Menu
	Fuel Capacity	Single Line
	Last Service Date	Date
	Vehicle Capacity	Single Line
	Next Service Date	Date
	Engine Displacement	Single Line
	Distance Travelled	Single Line
	Total Distance Travelled (till day)	Single Line
	Last Service Done (odometer reading)	Single Line
	Next Service Due (odometer reading)	Single Line
	Kilometres Remaining Till Next Service	Single Line

Vehicles - OBD Section	Device Identification	Single Line
	Device Type	Dropdown Menu
	SIM Mobile Number	Single Line
	Work Mode	Dropdown Menu

Vehicles - Insurance & Pollution Details	Insurance Company	Dropdown Menu
	Pollution Expiration Date	Date
	Insurance Expiration Date	Date
	Policy Number	Single Line
	IDV value	Currency
	Place of Issue	Single Line
Job Cards - general section	Customer Name	Lookup
	Service IN Date	Date
	Service Type	Dropdown Menu
	Vehicle	Lookup
	Due Date Kilometres Travelled	Date Number
	Service Status	Dropdown Menu
	Registration Number	Single Line
	Pick Up Date & Time	Date/Time
	Rough Estimate	Single Line
	Driver Name	Single Line
	Total Amount	Single Line
	Total Time Spent On Service (in hrs)	Single Line
	Arrived Service Type	Single Line
Job Cards - Complaints and Damage Details	Existing Damages	Multi Line
	Complaints	Multi Line

Job Cards - Additional Services	Alignment and Balancing	Checkbox
	Polish	Dropdown Menu
	Under Chassis Coating	Checkbox
	Upgrade Head Lamps	Checkbox
Job Cards - Insurance	Insurance Claim	Checkbox
	Surveyor Name	Single Line
	Insurance Company	Dropdown Menu
	Surveyor Phone Number	Phone
	Insurance Approval Status	Dropdown Menu
	Date of Survey	Date
	Insurance Amount	Currency
Repair Bills - general section	Repair Invoice Number	Auto Number
	Job Card	Lookup
	Payment Type	Dropdown Menu
	Vehicle Registration Number	Single Line
	Credit Card Last 4 Digits	Number
	Vehicle	Lookup
Repair Bills - Spares and Labor Calculation	Spares Total	Currency
	Labor Total	Currency
	CGST and SGST Tax System	Checkbox
	IGST Tax System	Checkbox

Repair Bills - CGST & SGST Tax Computation	Spare CGST (%)	Percent
	Labor CGST (%)	Percent
	Spare SGST (%)	Percent
	Labor SGST (%)	Percent
	Spares CGST Amount	Currency
	Spares SGST Amount	Currency

Repair Bills - IGST Computation	Spare IGST (%)	Percent
	Labor IGST (%)	Percent
	Spare IGST Amount	Currency
	Labor IGST Amount	Currency

Repair Bills - Total Calculation	Spares Tax Amount	Currency
	Spares Total After Tax	Currency
	Labor Tax Amount	Currency
	Labor Total After Tax	Currency

Repair Bills - Insurance Primary Inspection	Insurance Estimation	Currency
	Surveyor Name	Single Line
	Insurance Amount Approved	Currency
	Surveyor Email Address	Email
	Primary Inspection Date/Time	Date/Time
	Surveyor Phone Number	Phone

Repair Bills - Insurance Primary Inspection	Insurance Estimation	Currency
	Surveyor Name	Single Line
	Insurance Amount Approved	Currency
	Surveyor Email Address	Email
	Primary Inspection Date/Time	Date/Time
	Surveyor Phone Number	Phone

Repair Bills - Insurance Supplementary Inspection	Supplementary Estimation	
	Currency	
	Supplementary Surveyor Name	Single Line
	Supplementary Amount Approved	Currency
	Supplementary Surveyor Email Address	Email
	Supplementary Inspection Date/Time	Date/Time
	Supplementary Surveyor Phone Number	Phone
	Supplementary Surveyor Phone Number	Phone

Items - Item Information	HSN/SAC Code	Single Line
Items - Price Information	Cost Price	Currency
Items - Stock Information	Initial Stock Count	Decimal
	Reorder Stock Calculation	Formula
	Units Sold	Formula

Do it yourself: Hands - On Exercises

You had added three modules - Leads, Contacts, and Products to the Sample Garage App that you had earlier created. It is now time to configure the page layout and custom fields for these modules. This exercise requires you to edit the Leads module of your sample app.

We are now to customize the Leads and Contacts modules for mapping the details of trainers and franchises - Imagine that your garage staff are trained by multiple automobile franchises on service best practices. The sample garage app now has to store the details of all trainers and the franchises they are associated to.

1. Inside your sample app's setup page, click on the Modules section from the left menu and then click on the Leads module.
2. You will now enter the layout builder view. Hover over the fields to get the 'edit/delete' option.
3. Remove the following fields from the layout:
    - Fax
    - Title
    - Skype ID
    - Phone
4. Once you have removed the fields from the view, make the Title, Mobile, Email and First Name fields mandatory.

5. Add the following fields to the page layout :

Field Name	Type of Field
Lead Auto ID	Auto Number
Product Evaluated	Lookup (Products Module)
Lead Rating	Number

Setting access permissions in your app:

Now that you have multiple records, modules, and relationships set up inside your application, you now need to focus on your application's data security and user experience. A crucial part of building a business application is to decide which type of user gets to access certain data in the app.

For example, Garage CRM's users range from top level administrators to field service agents. An administrator will need access to the entire app setup and customization, whereas field agents are interested in only the emergency jobs assigned to them.

In this case, setting up access permissions not only makes the app data safe from accidental editing by field agents; it also provides agents with a clean view of only the information they need. We can give Garage CRM users different profiles – Administrator and Field Agent, to achieve this.

When building your app on Zoho Developer, you can create multiple user profiles for your application and customize the app for each profile. Each profile can be assigned to multiple users but a user can be associated to just one profile.

Do it yourself: Create a new profile

Add two new profiles to your sample garage app and name the profiles Service Technician and Service Adviser.

- In your app's setup page, select 'Permissions' from your left selection menu.
- Click on the New Profile button in the Permissions page.
- Create two new profiles with the required names.
- You can set permissions for each profile. Play around with the permissions ang give these profiles access to any module or field!

Listing and filtering data in modules:

All records stored in each module of your app will have their own List View. List views in general are a collection of ungrouped, unclassified records that users can sort through, bulk edit, bulk update, or delete. Your app also contains Zoho's prebuilt Advanced Filter, which gives users a more efficient way of searching for data in the app. With a regular filter, your search is based on field values - for example, all customers whose vehicle is a Ford.

This is pretty straightforward and ordinary. Whereas with advanced filters, you can run more advanced searches based on the activities associated to the records, such as "Fords that have been serviced in the past 5 weeks". Or "Fords without any job cards for the last 6 months".

Do it yourself: Create a new list view:

Creating a new list view requires you to move away from your apps setup page and into territory - your sandbox.

The sandbox is a front end view of your app that you can enter by clicking on the Test Your Application button on your app's setup page. The sandbox is where you can test any feature or component that you add to an application. To do that, you need to set up the right test data inside your app's sandbox.

Create a CSV spreadsheet of multiple leads (The more, the better) and import them into the leads module of your app. Make sure that your CSV contains information for all the mandatory fields of your leads page layout. You can also add information for all the non-mandatory fields in order to get proper list views and filters.

- Once you're done importing your test leads, head to the Leads module and you will find a prebuilt Advanced Filter on the right of your standard list view.
- Create a custom list view by filtering your leads based on any combination of parameters. All parameters you see on the Advanced Filter are information that is brought in from your custom fields.

Listing all related data - Related Lists:

Getting data into modules and displaying them is a static way of using an app, but your application acts as a dynamic collection of interrelated datasets when it tracks relationships between the data in each module. You can use Zoho Developer's Related List component to map relationships without having to manage any relational databases for your app. Once created, Related Lists become a part of your record's page layout.

For example, Garage CRM has a related list named 'Jobs' under each record in the Vehicles module. The related list is used to display details of all open job cards associated to that particular vehicle.

Do it yourself: Create a related list

In this exercise, we'll see how you can create a related list between the Leads and Products module in your sample garage app to see which garage services your leads are interested in.

- Select the Customize option from the left pane of your application's setup page.
- Select Related List from the list of available options at the top and select Customize.
- In the Related List Columns page, select the Products module from the Modules List.
- Select the fields from the Available Columns list that you want to display in your related list.
- Move the selected options to the Selected Columns list and Save.

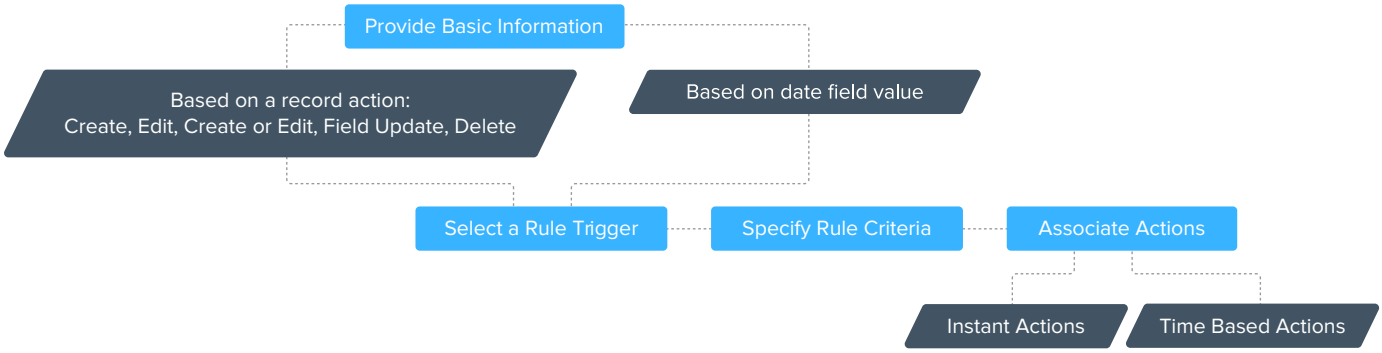
You've successfully created your first related list.

Advanced customization - Setting up automated processes

If you've created all the functionalities we've discussed till now in your application, you will have all the modules, custom fields, list views, and lookups needed to store all data and their relationships. After you have completed these data management steps, setting up workflows inside your application ensures that you can capture all business processes inside your application.

Each business has its own unique set of requirements. Workflow rules can be used to implement customizations suited to the needs of businesses who you target using your application. When writing a workflow, your workflow rule remains specific to a single module. You can set conditions and rule criteria to define how you work with data on that particular module and it's fields. However, the actions caused by the workflow rule are reflected as data changes in other modules as well.

Here's how you can configure a workflow for your app:



Example:
We used a workflow rule in Garage CRM to calculate the distance remaining for a vehicle to reach its next service checkpoint. The records present in the Vehicles module have fields named 'Distance Travelled Till Day and 'Remaining Kilometers for Next Service' to get values for performing this workflow.

The input fields that are to be filled in the Vehicles module are

- Distance Travelled Last Day
- Total Distance Travelled (till day)
- Last Service Done (in Kilometers)
- Remaining Kilometers for Next Service

Using these values, the value for the 'Remaining Kilometres for Next Service' is obtained through an automated workflow.

Going further, if you were to create a new workflow named 'Distance Update' in Garage CRM to mimic the same functionality mentioned earlier, here's how you can do it:

1. Click Automate in the left pane of your Zoho Developer account's Vertical CRM page and select Workflow
2. In the Rules tab, create a new Rule.
3. Give details for the following sections:

Basic Information

Module	:	Vehicles
Rule Name	:	Distance Update
Status	:	Check the box as 'Active' (to enable your workflow in the app)

Add a short description (To make sure you know why you created this workflow)

Rule Trigger

Execute based on	:	Select a Record Action

Click the Field Update

Select

1. "Distance Travelled Last Day" Field
2. None
3. None

Rule Criteria

Click “This workflow will only be executed the first time the record is edited”

Actions

Click “+” in Call Custom Functions

Copy and paste the following code in the Deluge editor page, click on the ‘Save to’ Button and save the workflow rule.

```
vehId = input.vehicle.get(“Vehicles.ID”);
crmResp = zoho.crm.getRecordById(“CustomModule2”, vehId.
toLong());
lastDay = (crmResp.get(“Distance Travelled Last Day”)).
toLong();
totalDistance = (crmResp.get(“Total Distance traveled(till
day)”)).toLong();
totalDistance = (totalDistance + lastDay);
lastService = (crmResp.get(“Last service done(KMS)”)).toLong();
nextService = (crmResp.get(“Next service due in KMS”)).
toLong();
temp = (totalDistance -  lastService);
remainingKms = (nextService - temp);
mapp = map();
// info totalDistance;
// info remainingKms;
mapp.put(“Total Distance traveled(till day)”, totalDistance);
mapp.put(“Remaining KMS for next service”, remainingKms);
crmRes = zoho.crm.updateRecord(“CustomModule2”, vehId.to-
String(), mapp);
info crmRes;
```

The above code is a custom function that you can add to your application. We’ll see custom functions in more detail in a dedicated section.

Adding custom buttons to your pages

If you have all workflows configured for your app, it can now handle business processes and automatically update records in all modules. But if someone were to create a new record in a module and associate it with a record in another module, the user has to create the record first and then add a lookup to the other record.

The process is time consuming and is even prolonged when someone is creating multiple records.

Using Custom Buttons can easily solve this particular use-case. They are a component available in the Zoho Developer Console that can be used to:

1. Connect different modules present in your app.
2. Connect your app to third party applications.

Example:
An 'Add Vehicle' Custom Button in Garage CRM's customers module's list view page lets users add a new vehicle to the Vehicles module and associate it with a customer at the click of a button. To do this,

1. Click Customize in the Left pane of your application.
2. Click Links and Buttons and select create new button.
3. Fill up the following details:
    - The module in which you'd like to create the button - Select the Customers module.
    - What would you like to name the button - Name the button as 'Add Vehicle'
    - Where would you like to place the button - Select the 'List View - Button for each record' from the drop down.
    - What action would you like the button to perform - Select 'Writing Custom Function' from the options present.

Add the following function:

- Include the following code in the editor and name the function as 'Adding Ve hicles'. Save the button.

```
constLookUpDelimiter = "||-||-||"; customerId = input.
customer.get("Contacts.ID"); respo = zoho.crm.getRecordById
("Contacts", customerId.toLong()); firstName = respo.get
("First Name"); if (firstName == null) { firstName = ""; }
LastName = respo.get("Last Name"); customerName = firstName +
" " + LastName; info respo; URLhash = "#Crm_CustomModule1_
COBJ2CF111=" + customerName + constLookUpDelimiter +
customerId; openUrl("https://myauto.zohosandbox.com/crm/
CreateEntity.do?module=CustomModule1" + URLhash, "New Tab");
return "";
```

You can also use Zoho Developer's Deluge Script Editor to invoke a URL or a widget and have it as a button action. Zoho supports both static and dynamic URLs, giving you a high degree of flexibilty. We'll see more of widgets in the coming sections.

Custom Functions

Custom functions are pieces of code that you can write to automate adding, updating, or transferring of the records in your application. You can use custom functions to either update records across multiple modules or use it for or automatically updating data in third-party applications.

The platform has its own script editor where you can write custom functions using Deluge Script. Deluge is Zoho's proprietary scripting language that was designed to help you perform complex actions with fewer lines of code than with traditional programming languages.

Writing Custom Functions using Deluge doesn't need much of a learning curve as it was designed keeping non-programmers in mind. It's also easy to get started right away if you're slightly familiar with Java programming.

You can include custom functions inside your app through the following components:

- Workflows
- Custom Buttons
- Related Lists
- Schedulers
- Formula Fields

How a custom function works:

While creating a custom function, you should be able to call the value of any custom field inside your apps records, to be able to update the field values. Custom functions make this possible by using implicit variables to hold values of all fields inside a record. Each module has a predefined variable name that you can use inside your function. The names of the implicit variables are same as the module names, but entirely in lower case.

For example, the variable for a Customer module would be ‘customer’.

The following example illustrates retrieving values from a record in the Customer module using the implicit variable customer.

```
customerId = input.customer.get (“Contacts.ID”);
name = input.customer.get(“First Name”);
phone = input.customer.get(“Mobile”);
```

This function would retrieve information from the Customer ID, First Name, and Mobile fields present in the records inside the Customer module of Garage CRM. While writing custom functions, you can use the script editor’s ‘Refer Fields’ feature to refer to the list of module names and fields. Move your mouse pointer to the center-right edge of the Deluge Script Editor to view the refer fields.

Integrating your app with other services

Zoho Developer provides the infrastructure to integrate your app with third party services like cloud telephony, document management, payment gateways, or customer support desks. You can also integrate any of your own custom applications, or proprietary apps that your team has built already.

You can either

- Push and sync data between your app and other services.
- Build contextual integrations between applications using widgets.
- Integrating your app with other services
- Creating custom variables for your app
- Embedding a weather widget in your app
- Learning mobile SDK
- Steps to build a hello world Android app

If you want your app to interact with other applications by pushing data and triggering events, say by getting customer support ticket information of a record in your app or by syncing online campaign data from Google Adwords, you can use a component known as a 'Connector' to do this.

Connectors facilitate integration between multiple applications using two forms of secure authorization - OAuth and token based authorization protocols. OAuth is an industry-standard, open-source authorization protocol, which is used to gain partial access to a user's account on an HTTP service.

For example, you use OAuth services to sign in to multiple web services using your Google, Facebook, or LinkedIn account.

Building connectors with OAuth authentication involves covering some complex topics that aren't present in this beginner level course. Instead, we'll focus on integrating using token based authorization protocols.

To integrate with applications that support token based authentication you will have to make use of custom variables.

Custom variables are components that store static information in your application. Think of them as a global variable that replaces certain key values in your app - such as any value stored in your custom field, allowing users to easily make system-wide changes and push data to other apps by calling the global variable.

To connect your application with a third party service, an auth token has to be created for that service and made into a custom variable. This process cannot be done by you during the app development process as a unique auth token is generated by the third party service for every user. Your app's users have to get the auth token from the third party service and update it using custom variables.

Creating custom variables for your app

Here's how you can create custom variables for your application:

- Click on the Access Console link under CRM for Verticals, in your Zoho Developer dashboard.
- Select the vertical app that you want to create custom variables for.
- From the left selection pane, select the Company Settings option.
- Select the Org Settings option and click on the Create button to create a custom variable.
- Click Save after entering the details for the variable.

There are two types of services which use token based authentication, apps that have a common token for all account and apps that generate a unique authorization token for every single account. For apps with a common authorization token, you can add the token under the Value section of your custom variable and save it. Hiding the field ensures that your users aren't displayed information that they don't have use for.

Your users will now have the integration available once they sign up for the application. For applications which generate a unique authorization code, you need to leave the Value field empty for your customers to fetch the auth token and enter the value.

Do it yourself: Building a Zillow integration

You can integrate your app with Zillow's real estate database using a token based integration model. You need to write connectors and custom functions in order to get the integration fully working, but for now we'll focus on the first step of getting it done - creating the custom variable.

The first step would be to create a Zillow account and generate a unique Zillow key known as the ZWSID. This ID is used to retrieve and display dynamic content from Zillow using Zillow API.

You can get your ZWSID from this link:
https://www.zillow.com/webservice/Registration.htm

Now create a custom variable with the following values:

Field	:	Zillow Account ID
API Name	:	ZWSID
Value	:	**<Enter your ZWSID>**

You have successfully created a Zillow custom variable.

Building more contextual integrations - widgets

Based on the understanding of Zoho Developer Console you have gained so far, you know that the console supports building of industry-specific applications without making you develop from scratch. You can use Zoho Developer's components to store and handle your data, handle business processes, and to perform advanced automations.

But what if you have propreitary apps, or third party apps that you want to make an integral part of your app's functionality?

Zoho Developer's Widget SDK is the answer to this challenge. Widget SDK ensures that you have the resources to bring in third party apps, or any of your own applications and make

them reside in your apps components. At times, you might want to include features that are available in third party apps .The ability to contextually integrate your application with these external services using custom user interfaces is achieved using widgets.

What are widgets

Widgets are a component of Zoho Developer that lets you surface any external application into your application's interface. Instead of just syncing and displaying data, you can embed the apps into your user interface and let your users work with them from inside your application's interface. You can use the widget framework to not only display other apps inside your apps enviroment, the embedded application also acts on your app's data in real time.

Regular integrations built on the console support pushing data and syncing between multiple applications. If you want a more feature-rich integration for your app, widgets give you the following advantages:

- Build a widget using any language of your choice and embed it inside your app.
- Widgets have access to all of your app data. You can work on a third party app using data from your vertical application.
- Your widget can pull data from any third party service using APIs and display it inside your app UI in real time.
- Beyond just contextually displaying data, widgets let you manipulate and combine data from different endpoints from inside the same application.
- WIdgets support any custom UI of your choice. You can design your oown application using third party app API and make it into a widget.
- Widgets support both server-side and client-side applications. You can make a wid get out of an app hosted in any external server and you can also host your widget in Zoho's servers.

Examples

Examples include the Google Drive widget which can be used to drag and drop any file into your app's record page to add that file into a Google Drive account. If there are no folders in Google Drive for that particular record in your application, the widget automatically creates a new folder for that record name.

If you're building an app for real estate agents or brokers, your users generally spend a lot of their time switching between Zillow and your app. Now imagine having all the property listings and Zestimates from Zillow embedded into records in your app. If users have a client interested in a commercial or residential property in a particular area, they can search Zillow's databases to find matching properties, based on your clients requirements.

How to create a widget

To configure a widget in your vertical, you need to create a connected app. The following steps will guide you through the process in detail.

- Head to your Zoho Developer account at developer.zoho.com and click on 'Access Console' under the CRM for Verticals section.
- Choose a vertical app that you've done already, or start with a sample app from scratch by using the 'Create App' button on the page. Fill all descriptions and select a couple of modules if you're starting with a new app.
- From the left pane menu, select the Connected Apps option to build your first widget. You'll be prompted to visit our help documentation, open the documentation in a sepa rate tab and use it for your reference.
- Provide basic details about your application - like the name of the app, short description, type of hosting etc.
- If your app exists already, enter the base URL and sandbox URL details into the console. During hands-on widget development, you will receive a file containing a sample app for you to get started right away.
- If you are to create an app from scratch to use it as a widget, refer to the help documen tation that you were prompted with while entering the Connected Apps page. The docu ment will provide all SAML protocol and OAuth related steps for you to follow while building the widget, but that goes beyond the reach of this particular guide.

Once you have built the widget, you can embed it into any app component that was listed in the previous section. You can either choose to host this app in a remote server as a server-side application, or use Zoho Developer's Command Line Interface (CLI) to host it as a client-side application in Zoho's servers. The CLI provides packaged commands that creates a .zip file from all of the files present in your application's folder.

Do it yourself: Embedding a weather widget in your app

Now that we know what widgets are and what steps are needed for to build one, its time to build your first widget. We'll build a sample weather widget that has the ability to provide real-time weather information based on the PIN Code or ZIP Code available inside a record. This widget is embedded as a separate Related List named 'WeatherWidget'.

Follow the steps below to get this widget into your sample app:

In the Connected Apps page, add the following details:

Name of the Connected App	:	Tech Mahindra
Description	:	Test weather widget.
Hosting	:	Server side application.
Sandbox URL	:	https://localhost:4443/
Production URL	:	https://localhost:4443/

You have successfully added the widget into your Developer account!
We can now make it a part of a custom Related List.

- Select Customize from the left menu pane in your console.
- Click on Related List from the top horizontal selection menu, and select 'Custom' and click on the Add Widgets button.
- In the page that follows, enter the details of the weather widget and select the modules in which you want it to appear in.
- The widget is now a part of your application. If you had selected the Leads module to embed this widget into, all records in that module will have a Related List named WeatherWidget that gives information on real-time weather based on the address information present in the record.

Building mobile apps for your application

Zoho Developer has an open source mobile SDK kit for Android that provide a bunch of native libraries that you can use to make secure connections to your application built on the platform, and quickly create mobile apps. The SDK abstracts all API calls and OAuth 2.0 calls that are involved in connecting your mobile app with the browser based application built using Zoho Developer.

To create Android apps using the SDK, you can download the latest version and get stared right away on your Android Studio account. There's no customization or setup change that you need to do to your Zoho Developer account for c7reating an app. Once built, your mobile apps act independent of the core browser-based app that they interact with, which means users can create or update any record information using their mobile applications even if they are not subscribed as users of the core application.

Steps to build a hello world Android app

Before you begin, get the .aar package using the URL below:

https://goo.gl/uvcgNZ

Get the latest version of our android SDK and follow the following steps in your Android Studio to build your hello world app.

1. Create a new Android Project

2. Add ZCRMAndroid.aar to your library in "app/libs"

To include the Android Application Record (AAR), add the following line in build.gradle (Project)

```
allprojects {
    repositories {
        jcenter()
        flatDir { dirs "libs"} // this line
    }
}
in build.gradle(App)
android {
packagingOptions {
     exclude "META-INF/LICENSE.txt"
          exclude "META-INF/LICENSE"
     }
}
dependencies {
     compile(name: 'ZCRMAndroid', ext: 'aar')
     compile 'org.apache.httpcomponents:httpclient-android:4.3.5'
     //    File upload
     compile 'org.apache.httpcomponents:httpmime:4.3'
}
```

3. Create a new folder named ‘assets’ under “app/src/main/”
4. Add a new file named as ‘app_configuration.properties’ with the code as given below:

```
appType=verticalcrm
#apiBaseUrl=https://crm.zoho.com
appDomain=garagecrm // your app domain
mainActivityClass=com.zoho.platform.vcrmtest.MainActivity
authScopes=ZohoCRM.modules.ALL,ZohoCRM.settings.READ,
ZohoCRM.users.READ,ZohoCRM.org.READ
```

In the line ‘appDomai=garagecrm’ as present in the above code, the appDomain property represents the domain of the vertical app that you have created in the Zoho Developer Console. Each app is mapped to a certain domain in the pattern domainID.zohoplatform.com

5. In the AndroidManifest.xml, add internet permissions to your application:

```
<uses-permission android:name="android.permission.
INTERNET" />
```

6. Remove the intent filter from your base activity since the applications base activity will be the login screen.

7. Now that you have the base activity as login, add the parent activity meta.

   Example:

```
<activity android:name=".MainActivity">
 </activity>
 <meta-data
      android:name="android.support.PARENT_ACTIVITY"
android:value=".MainActivity" />
```

8. Add the logout action by extending your main activity with ZCRMBaseActivity. By doing this, you get an override method named logout( ). You can now add your logout action by calling the super.logout( ) function.