



# GraphQL Schema Language Cheat Sheet

The definitive guide to express your GraphQL schema succinctly

Last updated: 28 January 2017  
Prepared by: Hafiz Ismail / @sogko

## What is GraphQL Schema Language?

It is a shorthand notation to succinctly express the basic shape of your GraphQL schema and its type system.

### What does it look like?

Below is an example of a typical GraphQL schema expressed in shorthand.

```
# define Entity interface
interface Entity {
  id: ID!
  name: String
}

# define custom Url scalar
scalar Url

# User type implements Entity interface
type User implements Entity {
  id: ID!
  name: String
  age: Int
  balance: Float
  is_active: Boolean
  friends: [User]!
  homepage: Url
}

# root Query type
type Query {
  me: User
  friends(limit: Int = 10): [User]!
}

# custom complex input type
input ListUsersInput {
  limit: Int
  since_id: ID
}

# root mutation type
type Mutation {
  users(params: ListUsersInput): [User]!
}

# GraphQL root schema type
schema {
  query: Query
  mutation: Mutation
  subscription: ...
}
```

Schema	
schema	GraphQL schema definition
query	A read-only fetch operation
mutation	A write followed by fetch operation
subscription	A subscription operation (experimental)

Built-in Scalar Types	
Int	Int
Float	Float
String	String
Boolean	Boolean
ID	ID

Type Definitions	
scalar	Scalar Type
type	Object Type
interface	Interface Type
union	Union Type
enum	Enum Type
input	Input Object Type

Type Modifiers	
String	Nullable String
String!	Non-null String
[String]	List of nullable Strings
[String]!	Non-null list of nullable Strings
[String!]!	Non-null list of non-null Strings

Input Arguments	
Basic Input	
type Query {	users(limit: Int): [User]
}	

Input with default value	
type Query {	users(limit: Int = 10): [User]
}	

Input with multiple arguments	
type Query {	users(limit: Int, sort: String): [User]
}	

Input with multiple arguments and default values	
type Query {	users(limit: Int = 10, sort: String): [User]
}	
type Query {	users(limit: Int, sort: String = "asc"): [User]
}	
type Query {	users(limit: Int = 10, sort: String = "asc"): [User]
}	

Input Types	
input ListUsersInput {	limit: Int
	since_id: ID
}	
type Mutation {	users(params: ListUsersInput): [User]!
}	

Custom Scalars	
scalar Url	
type User {	name: String
	homepage: Url
}	

Interfaces	
Object implementing one or more Interfaces	
interface Foo {	is_foo: Boolean
}	
interface Goo {	is_goo: Boolean
}	
type Bar implements Foo {	is_foo: Boolean
	is_bar: Boolean
}	
type Baz implements Foo, Goo {	is_foo: Boolean
	is_goo: Boolean
	is_baz: Boolean
}	

Unions	
Union of one or more Objects	
type Foo {	name: String
}	
type Bar {	is_bar: String
}	
union SingleUnion = Foo	
union MultipleUnion = Foo   Bar	
type Root {	single: SingleUnion
	multiple: MultipleUnion
}	

Enums	
enum USER_STATE {	NOT_FOUND
	ACTIVE
	INACTIVE
	SUSPENDED
}	
type Root {	stateForUser(userID: ID!): USER_STATE!
	users(state: USER_STATE, limit: Int = 10): [User]
}	