### How to Create a Use Case Diagram

Consider the various processes using a banking system as an example.

##### **Identifying Actors**

Actors are external entities that interact with your system. It can be a person, another system or an organization. In a banking system, the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role you’re trying to show in the use case.

An example of an external organization can be the tax authority or the central bank. The loan processor is a good example of an external system associated as an actor.

##### **Identifying Use Cases**

Now it’s time to identify the use cases. A good way to do this is to identify what the actors need from the system. In a banking system, a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So all of these can be considered as use cases.

Top level use cases should always provide a complete function required by an actor. You can extend or include use cases depending on the complexity of the system.

Once you identify the actors and the top level use case you have a basic idea of the system. Now you can fine tune it and add extra layers of detail to it.

##### **Look for Common Functionality to use Include**

Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that it’s always called when the original use case is executed. ( see the diagram for an example ).

##### **Is it Possible to Generalize Actors and Use Cases**

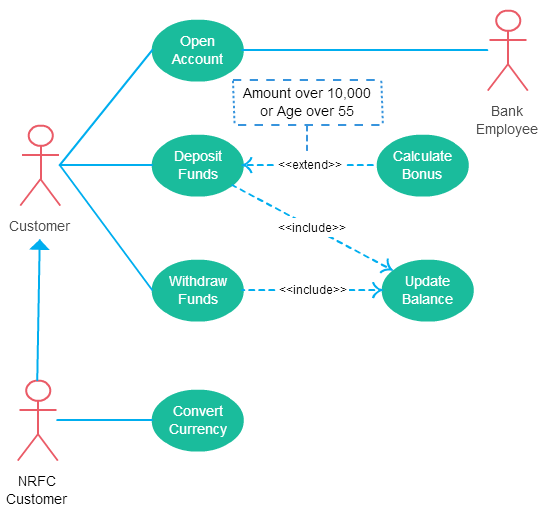
There may be instances where actors are associated with similar use cases while triggering a few use cases unique only to them. In such instances, you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.

One of the best examples of this is “Make Payment” use case in a payment system. You can further generalize it to “Pay by Credit Card”, “Pay by Cash”, “Pay by Check” etc. All of them have the attributes and the functionality of payment with special scenarios unique to them.

##### **Optional Functions or Additional Functions**

There are some functions that are triggered optionally. In such cases, you can use the extend relationship and attach an extension rule to it. In the below banking system example “Calculate Bonus” is optional and only triggers when a certain condition is matched.

Extend doesn’t always mean it’s optional. Sometimes the use case connected by extending can supplement the base use case. The thing to remember is that the base use case should be able to perform a function on its own even if the extending use case is not called.

[](https://d3n817fwly711g.cloudfront.net/blog/wp-content/uploads/2015/02/use-case-diagram-relationships-include.png)

*A use case with most of the scenarios found in use case diagrams*

Use Case Diagram Relationships with Examples

There can be 5 relationship types in a use case diagram.

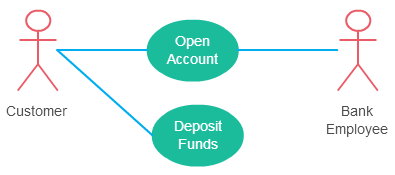
* Association between actor and use case
* Generalization of an actor
* Extend between two use cases
* Include between two use cases
* Generalization of a use case

Let’s take a look at these relationships in detail.

Association Between Actor and Use Case

This one is straightforward and present in every use case diagram. Few things to note.

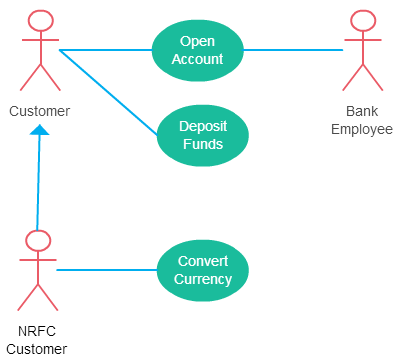
* An actor must be associated with at least one use case.
* An actor can be associated with multiple use cases.
* Multiple actors can be associated with a single use case.

[](https://d3n817fwly711g.cloudfront.net/blog/wp-content/uploads/2015/02/use-case-relationship-actor-use-case.png)

*Different ways association relationship appears in use case diagrams*

Generalization of an Actor

Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor. The descendant has one or more use cases that are specific to that role. Let’s expand the previous use case diagram to show the generalization of an actor.

[](https://d3n817fwly711g.cloudfront.net/blog/wp-content/uploads/2015/02/use-case-relationship-actor-generalization.png)

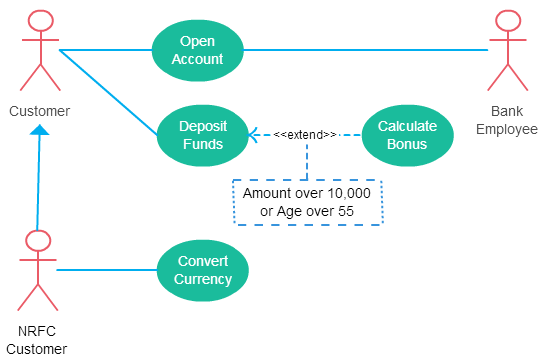
*A generalized actor in an use case diagram*

Extend Relationship Between Two Use Cases

Many people confuse the extend relationship in use cases. As the name implies it extends the base use case and adds more functionality to the system. Here are a few things to consider when using the <<**extend**>> relationship.

* **The extending use case is dependent on the extended (base) use case**. In the below diagram the “Calculate Bonus” use case doesn’t make much sense without the “Deposit Funds” use case.
* **The extending use case is usually optional** and can be triggered conditionally. In the diagram, you can see that the extending use case is triggered only for deposits over 10,000 or when the age is over 55.
* **The extended (base) use case must be meaningful on its own**. This means it should be independent and must not rely on the behavior of the extending use case.

Lets expand our current example to show the <<extend>> relationship.

[](https://d3n817fwly711g.cloudfront.net/blog/wp-content/uploads/2015/02/use-case-diagram-relationships-extend.png)

*Extend relationship in use case diagrams*

Although extending use case is optional most of the time it is not a must. An extending use case can have non-optional behavior as well. This mostly happens when your modeling complex behaviors.

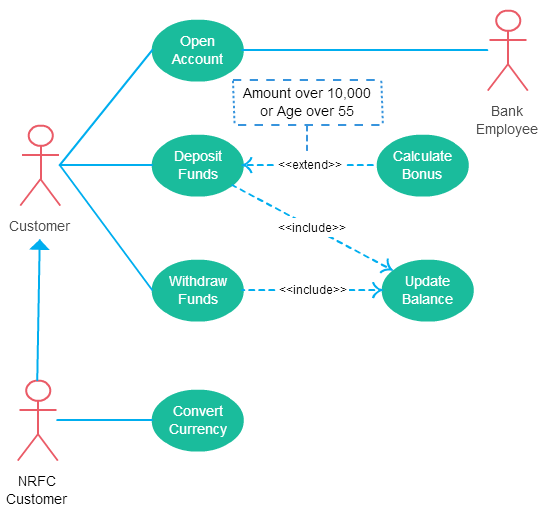
For example, in an accounting system, one use case might be “Add Account Ledger Entry”. This might have extending use cases “Add Tax Ledger Entry” and “Add Payment Ledger Entry”. These are not optional but depend on the account ledger entry. Also, they have their own specific behavior to be modeled as a separate use case.

Include Relationship Between Two Use Cases

Include relationship show that the behavior of the included use case is part of the including (base) use case. The main reason for this is to reuse common actions across multiple use cases. In some situations, this is done to simplify complex behaviors. Few things to consider when using the <<include>> relationship.

* The base use case is incomplete without the included use case.
* The included use case is mandatory and not optional.

Lest expand our banking system use case diagram to show include relationships as well.

[](https://d3n817fwly711g.cloudfront.net/blog/wp-content/uploads/2015/02/use-case-diagram-relationships-include.png)

*Includes is usually used to model common behavior*

For some further reading regarding the difference between extend and include relationships in use case diagrams check this [StackOverflow link](http://stackoverflow.com/questions/1696927/whats-is-the-difference-between-include-and-extend-in-use-case-diagram" \o "Difference between include and extend in use case diagrams" \t "_blank).

Generalization of a Use Case

This is similar to the generalization of an actor. The behavior of the ancestor is inherited by the descendant. This is used when there is common behavior between two use cases and also specialized behavior specific to each use case.