**Receive function**

A contract can have at most one receive function, declared using receive() external payable { ... } (without the function keyword). This function cannot have arguments, cannot return anything and must have external visibility and payable state mutability. It can be virtual, can override and can have modifiers.

When Ether is sent directly to a contract (without a function call, i.e. sender uses send or transfer) but the receiving contract does not define a receive Ether function or a payable fallback function, an exception will be thrown, sending back the Ether (this was different before Solidity v0.4.0). If you want your contract to receive Ether, you have to implement a receive Ether function (using payable fallback functions for receiving Ether is not recommended, since the fallback is invoked and would not fail for interface confusions on the part of the sender).

Receive is the function that is executed on plain Ether transfers (e.g. via .send() or .transfer()). If no such function exists, but a payable [fallback function](https://docs.soliditylang.org/en/v0.8.15/contracts.html#fallback-function) exists, the fallback function will be called on a plain Ether transfer. If neither a receive Ether nor a payable fallback function is present, the contract cannot receive Ether through regular transactions and throws an exception.

**Fallback Function**[**ℑ**](https://docs.soliditylang.org/en/v0.8.15/contracts.html#fallback-function)

A contract can have at most one fallback function, declared using either fallback () external [payable] or

fallback (bytes calldata input) external [payable] returns (bytes memory output)

 (both without the function keyword). This function must have external visibility.

A fallback function can be virtual, can override and can have modifiers.

The fallback function is executed on a call to the contract if none of the other functions match the given function signature, or if no data was supplied at all and there is no [receive Ether function](https://docs.soliditylang.org/en/v0.8.15/contracts.html#receive-ether-function).

The fallback function always receives data, but in order to also receive ether it must be marked payable.

**Warning**

A payable fallback function is also executed for plain Ether transfers, if no [receive Ether function](https://docs.soliditylang.org/en/v0.8.15/contracts.html#receive-ether-function) is present.

It is recommended to always define a receive Ether function as well, if you define a payable fallback function to distinguish Ether transfers from interface confusions.

Example

*// SPDX-License-Identifier: GPL-3.0*

**pragma solidity** >=**0.6.2** <**0.9.0**;

**contract** **Test** {

uint x;

*// This function is called for all messages sent to*

*// this contract (there is no other function).*

*// Sending Ether to this contract will cause an exception,*

*// because the fallback function does not have the `payable`*

*// modifier.*

fallback() external { x = 1; }

}

**contract** **TestPayable** {

uint x;

uint y;

*// This function is called for all messages sent to*

*// this contract, except plain Ether transfers*

*// (there is no other function except the receive function).*

*// Any call with non-empty calldata to this contract will execute*

*// the fallback function (even if Ether is sent along with the call).*

fallback() external payable { x = 1; y = **msg.value**; }

*// This function is called for plain Ether transfers, i.e.*

*// for every call with empty calldata.*

receive() external payable { x = 2; y = **msg.value**; }

}

**contract** **Caller** {

function callTest(Test test) public returns (bool) {

(bool success,) = address(test).call(abi.encodeWithSignature("nonExistingFunction()"));

require(success);

*// results in test.x becoming == 1.*

*// address(test) will not allow to call ``send`` directly, since ``test`` has no payable*

*// fallback function.*

*// It has to be converted to the ``address payable`` type to even allow calling ``send`` on it.*

address payable testPayable = payable(address(test));

*// If someone sends Ether to that contract,*

*// the transfer will fail, i.e. this returns false here.*

return testPayable.send(2 ether);

}

function callTestPayable(TestPayable test) public returns (bool) {

(bool success,) = address(test).call(abi.encodeWithSignature("nonExistingFunction()"));

require(success);

*// results in test.x becoming == 1 and test.y becoming 0.*

(success,) = address(test).call{value: 1}(abi.encodeWithSignature("nonExistingFunction()"));

require(success);

*// results in test.x becoming == 1 and test.y becoming 1.*

*// If someone sends Ether to that contract, the receive function in TestPayable will be called.*

*// Since that function writes to storage, it takes more gas than is available with a*

*// simple ``send`` or ``transfer``. Because of that, we have to use a low-level call.*

(success,) = address(test).call{value: 2 ether}("");

require(success);

*// results in test.x becoming == 2 and test.y becoming 2 ether.*

return true;

}

}