1.0. Interaction Receiver - Function Update

This section describes an enhancement feature for the existing interaction receiver module. The interaction receiver gets activated when a limit order is filled.
The interaction is an attribute of order. It is an unmodifiable data store containing address data in byte form. The field comprises two parts - the receiver's address and the wallet address. This interaction data is used to build the limit order and later used in transfers between the maker and the taker. Refer to the Interaction Receiver section to learn more details on this process.

1.1. Previous Implementation Details

In the initial implementation, there is only one interface called *InteractiveNotificationReceiver*. It contains a method called *notifyFillOrder* which is implemented by the used classes. Refer to the Interaction Receiver guide to study a related use case.

The use of this interface and containing methods has changed with the new changes. The new changes have improved the functionality of the limit orders and added more orderliness and security to the process.

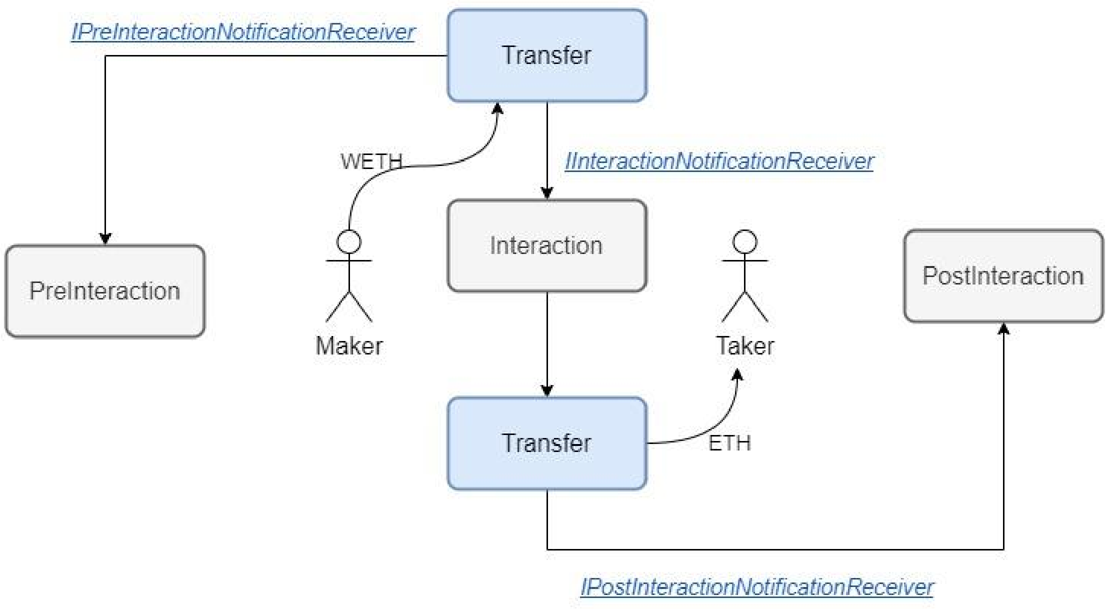
IPreInteractionNotificationReceiver
Transfer
WETH
IInteractionNotificationReceiver
PreInteraction
Maker
Interaction
Taker
PostInteraction
Transfer
ETH
IPostInteractionNotificationReceiver

Figure 1 - Interaction Flow

1.1.1. Interaction Receiver Updated Implementation Details

The new updates are handled by three interaction notification receivers.

The main functionality is included in the OrderMixin file. It handles the fund transfers between maker and taker and related processes.

Interaction Notifications
Interaction data is handled through *InteractiveNotificationReceiver* interfaces. There are three variations of this interface that are used to execute before a fund transfer, in the middle of a transfer, and after a transfer.

1. *IPreInteractionNotificationReceiver* : Interface for interactor which acts before the maker-to-taker transfers. This is the callback method that gets called before fund transfers.
2. *IInteractionNotificationReceiver* : Interface for interactor which acts while maker-to-taker fund transfers.
3. *IPostInteractionNotificationReceiver* : Interface for interactor which acts after taker-to-maker fund transfers. This is the callback method that gets called after all fund transfers.

1.2. Order Fund Transfer Steps

1.2.1. Prepare PreInteractionNotificationReceiver

In this step, the maker handles the funds interactively before transferring those from maker to taker. The execution will proceed only if the *interaction* length is enough to store the address. Then the *preInteraction* data from the order is assigned and return the values for the *interactionTraget* address and the *interactionData* fields.

fillOrderPreInteraction is the callback method that gets called before any fund transfers.

```
// Maker can handle funds interactively
if (order.preInteraction().length >= 20) {

// proceed only if interaction length is enough to store address
(address interactionTarget, bytes calldata interactionData) =
order.preInteraction().decodeTargetAndCalldata();

IPreInteractionNotificationReceiver(interactionTarget).fillOrderPr
eInteraction(
      orderHash, order.maker, msg.sender, actualMakingAmount,
actualTakingAmount, remainingMakingAmount, interactionData
   );
}
```

1.2.1.1. Handle Order Data via InteractionNotificationReceiver

This step handles the order data in the middle of the fund transfer. At this step, maker-to-taker transfer occurs first and then taker-to-maker transfer happens with a deposit to the relevant address. Interaction data from the order is taken to produce *InteractionTraget* and *interactionData* fields. A new variable *offeredTakingAmount* is derived from *fillOrderInteraction* method from *IInteractionNotificationReceiver* interface. If all the conditions are satisfied, assign the *actualTakingAmount* to *offeredTakingAmount.* Finally, the maker receives the ETH with the help of *interactionReceiver.*

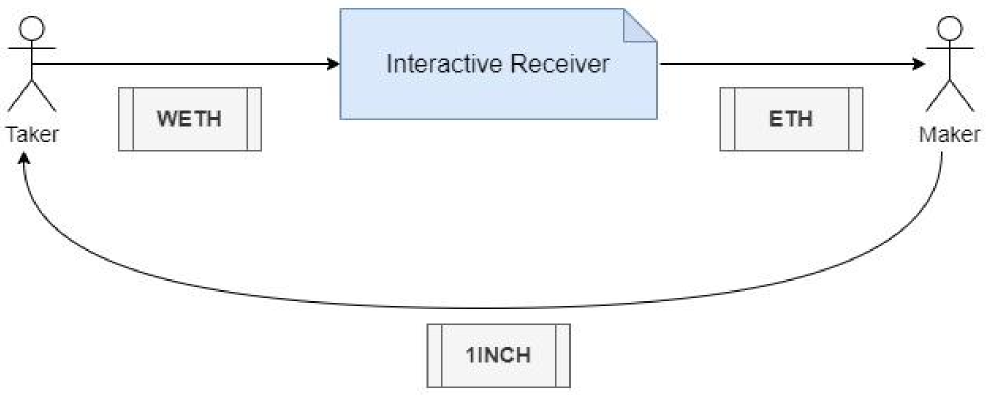

Figure 2 - Token Transfer

```
if (interaction.length >= 20) {

// proceed only if interaction length is enough to store address
(address interactionTarget, bytes calldata interactionData) =
interaction.decodeTargetAndCalldata();
   uint256 offeredTakingAmount =
IInteractionNotificationReceiver(interactionTarget).fillOrderInter
action(msg.sender, actualMakingAmount, actualTakingAmount,
interactionData);

if (offeredTakingAmount > actualTakingAmount &&
!OrderLib.getterIsFrozen(order.getMakingAmount()) &&
!OrderLib.getterIsFrozen(order.getTakingAmount()))
            {
                actualTakingAmount = offeredTakingAmount;
            }
}
```

1.2.1.2. Handle Funds via PostInteractionNotificationReceiver

In this step, the maker handles the funds interactively after completing the fund transfer. The execution will proceed only if the interaction length is enough to store the address. Then the postInteraction data from the order is assigned and the values for the *interactionTraget* address and the *interactionData* fields are received to support the post transfer execution.

fillOrderPostInteraction is the callback method that gets called after any funds transfers.

```
// Maker can handle funds interactively
if (order.postInteraction().length >= 20) {

// proceed only if interaction length is enough to store address
(address interactionTarget, bytes calldata interactionData) =
order.postInteraction().decodeTargetAndCalldata();

IPostInteractionNotificationReceiver(interactionTarget).fillOrderP
ostInteraction(orderHash, order.maker, msg.sender,
actualMakingAmount, actualTakingAmount, remainingMakingAmount,
interactionData);
        }
}
```

1.3. Order Handling Methods

Interaction data is used in three order methods - *fillOrder*, *fillOrderToWithPermit*, and *fillOrderTo*. In the previous implementation, there was only one method called *notifyFillOrder*. The three methods in the updated version are:

- *fillOrder* - Fills an order. If one doesn't exist (first fill) it will be created using order.makerAssetData
- *fillOrderToWithPermit* - Same as `fillOrderTo` but calls permit first, allowing to approve token spending and make a swap in one transaction. Also allows specifying funds destination instead of `msg.sender`
- *fillOrderTo* - Same as `fillOrder` but allows to specify funds destination instead of `msg.sender`

End of the Task Document

Notes:

- *Please note that some technical points need further clarifications to include in the document.*
- *In some occurrences, I was not sure whether to include specific details in the documentation or not. So, maybe certain details may have to remove.*