```
function atomicMatch(Order memory buy, Sig memory buySig, Order memory sell, Sig memory sellSig, bytes32 metadata){
    /* Core business logic non-related checks */
    ...
    /* Must be matchable. */
    require(ordersCanMatch(buy, sell));
    ...
    /* Execute funds transfer and pay fees. */
    uint price = executeFundsTransfer(buy, sell);
    ...
}

function ordersCanMatch(Order memory buy, Order memory sell){
    return (
        ...
        SaleKindInterface.canSettleOrder(buy.listingTime, buy.expirationTime) &&
        SaleKindInterface.canSettleOrder(sell.listingTime, sell.expirationTime)
    );
}

function canSettleOrder(uint listingTime, uint expirationTime){
    return (listingTime < now) && (expirationTime == 0 || now < expirationTime);
}

function executeFundsTransfer(Order memory buy, Order memory sell){
    ...
    /* Calculate match price. */
    uint price = calculateMatchPrice(buy, sell);

    if (price > 0 && sell.paymentToken != address(0))
        transferTokens(sell.paymentToken, buy.maker, sell.maker, price);

    ...
    return price;
}

function calculateMatchPrice(Order memory buy, Order memory sell){
    /* Calculate sell price. */
    uint sellPrice = SaleKindInterface.calculateFinalPrice(sell.side, sell.saleKind, sell.basePrice, sell.extra, sell.listingTime, sell.expirationTime);
    ...
    /* Require price cross. */
    require(buyPrice >= sellPrice);
    return sell.feeRecipient != address(0) ? sellPrice : buyPrice;
}

function calculateFinalPrice(Side side, SaleKind saleKind, uint basePrice, uint extra, uint listingTime, uint expirationTime) {
    ...
    uint diff = SafeMath.div(SafeMath.mul(extra, SafeMath.sub(now, listingTime)), SafeMath.sub(expirationTime, listingTime));

    return SafeMath.sub(basePrice, diff);
}

function transferFrom(address token, address from, address to, uint amount){
    return ERC20(token).transferFrom(from, to, amount);
}
```