Inhalt

Ereignisse

Der Handel mit anderen Parteien löst Ereignisse aus. Du kannst im Skript auf diese Ereignisse reagieren. Dazu musst Du einen sogenannten *Event Listener* erstellen.

Ereignis: Waren einkaufen

Einkäufe durch den Spieler lösen das Ereignis `QSB.ScriptEvents.GoodsPurchased` aus. Es ermöglicht Dir an verschiedene Daten heranzukommen.

Zuerst legst Du die Variable an. Das kannst Du an einer beliebigen Stelle tun.

```
function EinkaufVariableAnlegen()
    gvMission.PurchasedFromPlayer2 = 0;
end
```

Danach erstellst Du einen Event Listener.

```
function EinkaufEventListenerAnlegen()
    -- Die ID des Event Listeners wird für später gespeichert
    gvMission.EventListenerPurchasePlayer2 = API.AddScriptEventListener(
        QSB.ScriptEvents.GoodsPurchased,
        function (_Index, _Type, _Good, _Amount, _Price, _PlayerID1, _PlayerID2)
            -- Nur zählen, wenn der Händler Spieler 2 ist
            if _PlayerID2 == 2 then
                gvMission.PurchasedFromPlayer2 = gvMission.PurchasedFromPlayer2 +1;
            end
        end
    );
end
```

Jetzt wird bei jedem Einkauf durch den Spieler die Variable um 1 erhöht.

Den Wert kannst Du z.B. in einem Quest Behavior verwenden.

```
function Goal_HatSpielerGenugGekauft()
    -- Es müssen mindestens 5 Angebote gekauft werden.
    if gvMission.GoodsPurchasedFromPlayer2 >= 5 then
        return true;
    end
end
```

Der Spieler muss mindestens 5 mal (bei Spieler 2) eingekauft haben. Das Behavior ist erfüllt, wenn diese Bedingung erfüllt ist.

Ereignis: Waren verkaufen

Der Verkauf von Waren löst das Ereignis `QSB.ScriptEvents.GoodsSold` aus. Du kannst dadurch ebenfalls an verschiedene Daten heranzukommen.

Wieder brauchst Du eine Variable für die Werte. Diesmal nehmen wir eine Tabelle.

```
function VerkaufVariableAnlegen()
    gvMission.GoodsSold = {};
end
```

Im Event Listener speicherst Du die Anzahlen aller verkauften Waren.

```
function VerkaufEventListenerAnlegen()
    -- Die ID des Event Listeners wird für später gespeichert
    gvMission.EventListenerGoodsSold = API.AddScriptEventListener(
        QSB.ScriptEvents.GoodsSold,
        function (_Type, _GoodAmount, _Price, _PlayerID, _TargetPlayerID)
            gvMission.GoodsSold[_Type] = gvMission.GoodsSold[_Type] or 0;
            gvMission.GoodsSold[_Type] = gvMission.GoodsSold[_Type] + _GoodAmount;
        end
    );
end
```

Jetzt wird für jeden Warentyp die Verkaufsmenge erfasst.

Diese Werte kannst Du z.B. in einem Quest Behavior verwenden.

```
function Goal_HatSpielerGenugVerkauft()
    -- Wenn noch nicht 100 Wildtierkörper verkauft wurden,
    -- ist die Aufgabe nicht erfüllt.
    if not gvMission.GoodsSold[Goods.G_Carcass]
        or gvMission.GoodsSold[Goods.G_Carcass] < 100 then
        return;
    end
    -- Wenn noch nicht 100 Getreide verkauft wurde,
    -- ist die Aufgabe nicht erfüllt.
    if not gvMission.GoodsSold[Goods.G_Grain]
        or gvMission.GoodsSold[Goods.G_Grain] < 100 then
        return;
    end
    -- Es wurden genug Waren verkauft
    return true;
end
```

Der Spieler muss mindestens 100 Tierkörper und 100 Getreide verkaufen. Das Behavior ist erfüllt, wenn diese Bedingung erfüllt ist.

Hooks

Beim Kauf oder Verkauf werden verschiedene Werte berechnet. Du kannst diese Berechnungen ändern. Das geht generell für alle oder für einen bestimmten Spieler.

Die folgenden Beispiele funktionieren nur im lokalen Skript.

Hook: Basispreis (Einkauf)

Der Basispreis ermittelt sich aus der Tabelle `MerchantSystem.BasePrices`. Es reicht oft aus, den Wert direkt in den Basispreisen zu verändern. Manchmal möchte man den Wert aber nur bedingt verändern. Beispielsweise nur für einen bestimmten Spieler.

```
function BasispreisBeiSpieler2Anedern()
    API.PurchaseSetBasePriceForPlayer(
        2,
        function(_Type, _Good, _PlayerID1, _PlayerID2)
            local BasePrice = MerchantSystem.BasePrices[_Good];
            -- Nur für Spieler 1 wird der Basispreis verdoppelt
            if _PlayerID2 == 1 then
                BasePrice = BasePrice * 2;
            end
           return (BasePrice == nil and 3) or BasePrice;
        end
    );
end
```

In diesem Beispiel wird der Basispreis nur bei Spieler 2 verdoppelt.

Hook: Basispreis (Verkauf)

Den Basispreis für den Verkauf kannst Du auf die gleiche Weise anpassen.
Siehe dazu: `API.SaleSetBasePriceForPlayer(_PlayerID, _Function)`

Hook: Inflation (Einkauf)

Die Inflation wird durch die gekauften Menge ermittelt. Der Preis steigert sich um 25%. Der Preis steigt nur bis zur achten gekauften Wagenladung.

```
function InflationBeiSpieler2Aendern()
    API.PurchaseSetInflationForPlayer(
        2,
        function(_Type, _Good, _OfferCount, _Price, _PlayerID1, _PlayerID2)
            -- Bei Speiler 2 ist die Inflation nicht auf ein
            -- Maximum von 8 beschränkt.
            if _PlayerID1 ~= 1 and _OfferCount > 8 then
                OfferCount = 8;
            end
            -- Die Berechnung bleibt unverändert.
            local Result = _Price + (math.ceil(_Price / 4) * _OfferCount);
            return (Result < _Price and _Price) or Result;
        end
    );
end
```

In diesem Beispiel steigt die Inflation beim Einkauf bei Spieler 2 ins Unermessliche.

Hook: Deflation (Verkauf)

Die Deflation des Verkaufs kannst Du auf die gleiche Weise anpassen.
Siehe dazu: `API.SaleSetDeflationForPlayer(_PlayerID, _Function)`

Hook: Handelsbedingung (Einkauf)

Normalerweise sind alle Angebote eines Handelspartners verfügbar. Du kannst zusätzliche Bedingungen vorgegeben. Sind sie nicht erfüllt, kann der Spieler die Ware nicht kaufen.

```
function EinkaufBedingungVorgeben()
    API.PurchaseSetConditionForPlayer(
        2,
        function(_Type, _Good, _Amount, _PlayerID1, _PlayerID2)
            -- Spieler 1 kann nur mit Spieler 2 handeln, wenn
            -- Handelsbeziehungen zu Spieler 3 bestehen
            if _PlayerID1 == 1 then
                return Diplomacy_GetRelationBetween(_PlayerID1, 3) >= 1;
            end
            return true;
        end
    );
end
```

In diesem Beispiel kann bei Spieler 2 nicht sofort gekauft werden. Spieler 2 verkauft die Waren erst, wenn mit Spieler 3 Handelsbeziehungen aufgenommen wurden.

Hook: Handelsbedingung (Einkauf)

Du kannst Bedingungen für den Verkauf auf die gleiche Weise einstellen.
Siehe dazu: `API.SaleSetConditionForPlayer(_PlayerID, _Function)`

Hook: Heldenfähigkeit (Einkauf)

Der Spieler kommt normalerweise einen Discount von 20%. Natürlich nur mit Elias als Helden. Du kannst in die Berechnung eingreifen. Beispielsweise, um den Discount zu erhöhen.

```
function HeldenfaehigkeitAnpassen()
    API.PurchaseSetTraderAbilityForPlayer(
        2,
        function(_Type, _Good, _BasePrice, _PlayerID1, _PlayerID2)
            local Modifier = 1.0;
            -- Nur für Elias wird ein Bonus gewährt
            local KnightType =  Logic.GetEntityType(Logic.GetKnightID(_PlayerID1));
            if KnightType == Entities.U_KnightTrading then
                Modifier = 1.4;
           end
           return math.ceil(_BasePrice / Modifier);
        end
    );
end
```

In diesem Beispiel bekommt der Spieler weiterhin durch Elias einen Discount. Der Discount wurde jedoch bei Spieler 2 von 20% auf 40% erhöht.

Hook: Handelsbedingung (Verkauf)

Du kannst den Discount auch für den Verkauf einstellen.
Siehe dazu: `API.SaleSetTraderAbilityForPlayer(_PlayerID, _Function)`