**Who Calls Payment Commands in SAP Commerce Cloud in (SOP)?**

In SAP Commerce Cloud, the payment-related commands (like **authorize**, **capture**, and **refund**) are triggered by the **eCommerce platform**, typically within **checkout, order processing, or business logic** layers. The **PSP** (like Adyen, Stripe) does **not directly invoke these commands**. Instead, SAP Commerce sends **API requests** to the PSP by calling these commands in specific steps of the order and payment workflows.

**1. Authorization Request (Triggered during Checkout)**

* **When?**  
  After the user confirms the order (typically on the **checkout page**).
* **Who Triggers the Command?**  
  The **eCommerce platform** (via PaymentService) sends an **authorization request** to the PSP.

**Flow:**

1. **User confirms the order**.
2. The **CheckoutService** finalizes the cart and triggers **authorize**.
3. **PaymentService** calls the **authorize command** to place a hold on the payment amount.

**Command Execution:**

|  |
| --- |
| PaymentTransactionEntryModel authorizationEntry =  paymentService.authorize("orderCode", paymentInfoModel, amount); |

* This command sends an **API request** to the PSP (like Adyen) to authorize the payment.
* The PSP responds with an **authorization token** or **status** (e.g., "AUTHORIZED" or "REJECTED").

**2. Payment Capture (Triggered Post-Authorization)**

* **When?**  
  After successful **authorization**. Capture can be triggered:
  + **Immediately:** If you want payment captured as soon as the order is placed (e.g., for digital products).
  + **After Shipment:** For physical products, payment capture might be delayed until shipment confirmation.
* **Who Triggers the Command?**  
  SAP Commerce triggers the **capture command**—this is often handled automatically by the **OrderProcess** or **Fulfillment Process**.

**Command Execution:**

|  |
| --- |
| PaymentTransactionEntryModel captureEntry =  paymentService.capture("orderCode", paymentTransaction, amount); |

* This command sends a **capture request** to the PSP, which transfers the money from the customer’s account to the merchant’s account.
* If successful, the **PaymentTransactionModel** is updated to **"CAPTURED"**.

**3. Refund (Triggered during Returns or Order Cancellations)**

* **When?**  
  If an order is canceled, or the customer returns a product, a **refund** must be initiated.
* **Who Triggers the Command?**  
  The **RefundProcess** or **CustomerService** in SAP Commerce triggers the **refund command**.

**Command Execution:**

|  |
| --- |
| PaymentTransactionEntryModel refundEntry =  paymentService.refund("orderCode", paymentTransaction, refundAmount); |

* This command sends a **refund request** to the PSP. The PSP returns the funds to the customer and updates the transaction status to **"REFUNDED"**.

**How the PSP is Involved in the Payment Flow**

1. **API Calls from SAP Commerce to PSP:**
   * When commands like **authorize, capture, or refund** are executed, SAP Commerce sends **API requests** to the PSP.
2. **PSP Responses:**
   * The PSP responds with the status of each operation (e.g., **authorized**, **captured**, **rejected**).
   * In some cases, PSPs send **webhooks** to notify the system asynchronously about changes (e.g., capture or refund confirmation).
3. **Handling PSP Notifications:**
   * SAP Commerce listens to these webhooks and updates the **PaymentTransactionModel** accordingly.
   * Example: A webhook may trigger an **event listener** to update the order status when the payment is captured.

**How Request Flow Works with HOP Integration**

1. **Customer Checkout & Redirection to PSP (Hosted Page)**
2. **Customer Completes Payment on PSP's Page**
3. **PSP Sends Response to SAP Commerce Cloud (via Callback/Notification & Customer Redirect)**
4. **SAP Commerce Validates the Response and Finalizes the Order**

**Detailed Step-by-Step Flow with HOP Integration**

**1. Customer Checkout & Redirection to PSP**

* **User adds items to the cart** and proceeds to checkout.
* After collecting delivery and billing details, the eCommerce platform redirects the user to the **PSP’s Hosted Order Page (HOP)** to complete the payment.

**Code Example – Redirect to PSP’s HOP:**

|  |
| --- |
| String redirectUrl = paymentService.getRedirectUrlForHOP(cartModel, paymentInfoModel, returnUrl);  response.sendRedirect(redirectUrl); |

* **redirectUrl**: The PSP-provided URL where the customer completes the payment.
* **returnUrl**: The URL to which the customer will be redirected after completing the payment.

**2. Customer Completes Payment on PSP’s HOP**

* The customer enters **payment details** (e.g., card info) directly on the PSP’s secure payment page.
* Once the payment is processed, the PSP performs two actions:
  1. **Redirects the customer back to SAP Commerce** using the **return URL**.
  2. **Sends a server-to-server notification** (e.g., a webhook) to SAP Commerce to confirm the payment status.

**3. PSP Sends Payment Confirmation via Redirect and Callback (Webhook)**

**A. Customer Redirect to Return URL**

* After the payment is completed, the customer is redirected back to a **return URL** provided by SAP Commerce during the checkout process.
* The **return URL** typically includes **query parameters** with the payment status (e.g., success/failure) and a **payment transaction reference**.

**Example Return URL:**

|  |
| --- |
| https://your-commerce-store/payment/return?status=SUCCESS&transactionRef=123456789 |

* SAP Commerce captures this response and updates the **PaymentTransactionModel** accordingly.

**Code Example – Handling Return:**

|  |
| --- |
| String transactionRef = request.getParameter("transactionRef");  String paymentStatus = request.getParameter("status");  if ("SUCCESS".equals(paymentStatus)) {  paymentService.updateTransactionStatus(transactionRef, TransactionStatus.ACCEPTED);  } else {  paymentService.updateTransactionStatus(transactionRef, TransactionStatus.REJECTED);  } |

**B. PSP Sends Server-to-Server Notification (Webhook)**

* In addition to redirecting the customer, the PSP sends a **webhook notification** to a specific **callback URL** configured in SAP Commerce.
* This ensures that even if the **customer does not return** to the website (e.g., they close the browser), the **payment status is still updated correctly**.

**Sample Webhook JSON from PSP:**

|  |
| --- |
| {  "transactionRef": "123456789",  "status": "CAPTURED",  "amount": 100.00,  "currency": "USD"  } |

* SAP Commerce listens for this webhook and updates the **PaymentTransactionModel**.

**Code Example – Handling Webhook Notification:**

|  |
| --- |
| @WebServlet("/payment/callback")  public class PaymentCallbackServlet extends HttpServlet {  @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {  String transactionRef = request.getParameter("transactionRef");  String status = request.getParameter("status");  if ("CAPTURED".equals(status)) {  paymentService.updateTransactionStatus(transactionRef, TransactionStatus.ACCEPTED);  } else {  paymentService.updateTransactionStatus(transactionRef, TransactionStatus.REJECTED);  }  }  } |

**4. SAP Commerce Finalizes the Order**

1. **After receiving the payment confirmation** from the PSP (via redirect or webhook), SAP Commerce updates the **PaymentTransactionModel** with the final status.
2. **If the payment is successful**, SAP Commerce:
   * Creates the **OrderModel**.
   * Marks the **payment status** as PAID.
   * Triggers the **OrderProcess** to begin fulfillment.

**Code Example – Creating Order:**

|  |
| --- |
| if (paymentTransaction.getStatus().equals(TransactionStatus.ACCEPTED)) {  OrderModel order = orderService.createOrderFromCart(cartModel);  order.setPaymentStatus(PaymentStatus.PAID);  modelService.save(order);  // Start order process  businessProcessService.triggerEvent(order.getCode() + "\_OrderPlaced");  } |

**Error Handling & Payment Failures**

* If the payment fails:
  + The order is **not created**.
  + The cart remains active for the user to retry.
  + A **notification is sent** to the user informing them of the failure.

**Summary of Key Steps in HOP Payment Flow**

| **Step** | **Description** | **Involved Component** |
| --- | --- | --- |
| **Checkout & Redirect to PSP** | Customer redirected to PSP’s hosted page. | CheckoutService, PaymentService |
| **Payment on PSP’s Page** | Customer completes payment on PSP’s hosted page. | PSP (Adyen, PayPal, etc.) |
| **Customer Redirect to Store** | Customer redirected back to eCommerce store with status. | PaymentService, OrderProcess |
| **Webhook Notification (Callback)** | PSP sends payment confirmation via server-to-server call. | PSP, PaymentCallbackServlet |
| **Order Finalization** | Order created after successful payment confirmation. | OrderService, BusinessProcess |

**Conclusion**

In the **HOP payment flow**, SAP Commerce Cloud interacts with the **PSP** through:

1. **Redirect URLs**: To receive payment status after customer checkout.
2. **Webhooks (server-to-server callbacks)**: To ensure payment status is updated even if the customer doesn’t return to the site.

This dual mechanism ensures that **payment status updates are reliable,** and the order flow remains seamless.