PowerBuilder

Code for nerds, stuff that matters

Building a Custom Payment in Optimizely Commerce 14 (with a simple “Account” method)

ON 17 OCTOBER, 202517 OCTOBER, 2025 / BY FRANCISCO QUINTANILLA / IN OPTIMIZELY

There’s no official step-by-step documentation for this flow in Commerce 14, so I decompiled parts of the platform and leaned on the (older) Commerce books (https://leanpub.com/proepiservercommerce)to piece it together. This post shows a minimal, working path and explains only the bits you need—no deep dives.

I based the sample on Foundation; a few helper classes are reused from that project and not shown here. You can copy any missing helpers from the Foundation repo: https://github.com/episerver/Foundation/tree/main (https://github.com/episerver/Foundation/tree/main)

What we’re building

We’ll add a very simple **Account** payment:

1. A **Payment Option** that collects an `AccountNumber` and creates a `Payment` object to put into the order.
2. A **Payment Gateway** that Optimizely calls during processing (authorize/capture/refund).
3. A **Payment configuration** in the Admin UI, where you choose:
   - **Class Name** → your gateway class (implements `IPaymentGateway`)
   - **Payment Class** → the concrete `Payment` subclass your option returns (we’ll use `OtherPayment` to keep it simple).

The “Payment Class” is explicitly a class that inherits `Mediachase.Commerce.Orders.Payment` .

1) Payment Option — `AccountPaymentOption`

This is the UI model that surfaces “Account” at checkout, validates input, and creates a `Payment` .

```
using YourNamespace.Web.Infrastructure.Commer
using Mediachase.Commerce.Orders;
using Mediachase.Commerce.Orders.Managers;

namespace YourNamespace.Web.Features.Checkout
{
    public class AccountPaymentOption(
        LocalizationService localizationServi
        IOrderGroupFactory orderGroupFactory,
        ICurrentMarket currentMarket,
        LanguageService languageService,
        IPaymentService paymentService)
        : PaymentOptionBase(localizationServi
    {
        public override string SystemKeyword 

        public string AccountNumber { get; se

        public override IPayment CreatePaymen
        {
            var implementationClassName = Pay
                .PaymentMethod[0].PaymentImpl

            var type = Type.GetType(implement
            var payment = type == null
                ? orderGroup.CreatePayment(Or
                : orderGroup.CreatePayment(Or

            payment.PaymentMethodId = Payment
            payment.PaymentMethodName = Syste
            payment.Amount = amount;
            payment.PaymentType = PaymentType

            // Custom data for the gateway
            payment.Properties["AccountNumber

            return payment;
        }

        public override bool ValidateData()
        {
            // minimal validation; expand as
            return !string.IsNullOrWhiteSpace
        }
    }
}
```

Why this class exists

Checkout needs a component to **collect input** and return a `Payment` object to `OrderForm.Payments`.

2) Payment Gateway — `AccountPaymentGateway`

This is the final integration point between Optimizely Commerce and your payment provider/logic. Every new payment type should have a gateway.

```
using Mediachase.Commerce.Orders;
using Mediachase.Commerce.Plugins.Payment;

namespace YourNamespace.Web.Features.Checkout
{
    public class AccountPaymentGateway : Abst
    {
        public PaymentProcessingResult Proces
        {
            if (string.IsNullOrEmpty(payment.
            {
                return PaymentProcessingResul
            }

            // Your custom business logic / e
            return PaymentProcessingResult.Cr
        }

        public override bool ProcessPayment(P
        {
            var result = ProcessPayment(null,
            message = result.Message;
            return result.IsSuccessful;
        }
    }
}
```

Why this class exists

Gateways implement the **processing** side (authorize/capture/refund).

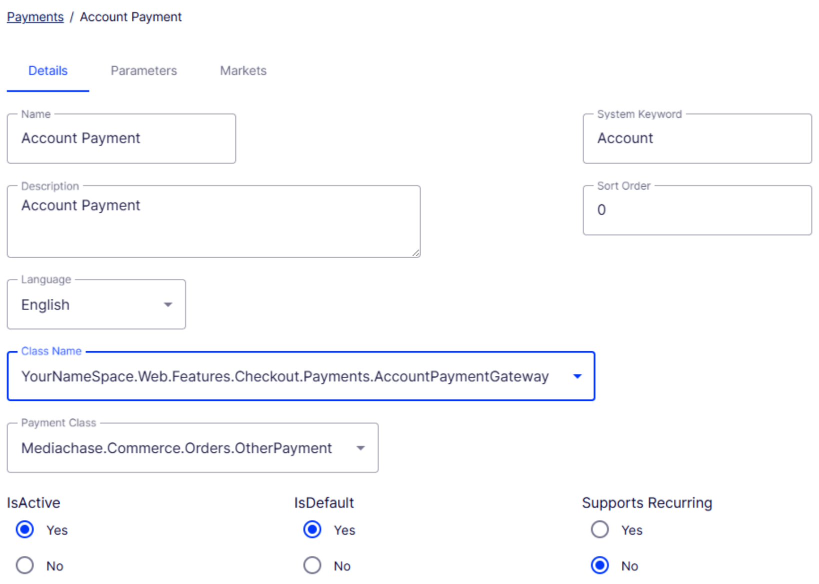
3) Wire-up (DI) and Admin setup

Register the payment option so your checkout can resolve it:

```
// in an initialization module
context.Services.AddTransient<IPaymentMethod,
```

Then in **Settings → Payments** add a new payment:

- **Class Name** → your fully-qualified `AccountPaymentGateway`
- **Payment Class** → `Mediachase.Commerce.Orders.OtherPayment`
- **Market** → Default Market


Payments / Account Payment

Name	Account Payment
System Keyword	Account
Description	Account Payment
Sort Order	0
Language	English
Class Name	YourNameSpace.Web.Features.Checkout.Payments.AccountPaymentGateway
Payment Class	Mediachase.Commerce.Orders.OtherPayment
IsActive	[✓ Yes] No
IsDefault	[✓ Yes] No
Supports Recurring	Yes [✓ No]

Why these two fields matter:

- **Class Name** must point to your gateway (implementation of `IPaymentGateway`).
- **Payment Class** must be a concrete subclass of `Payment` that your option returns.

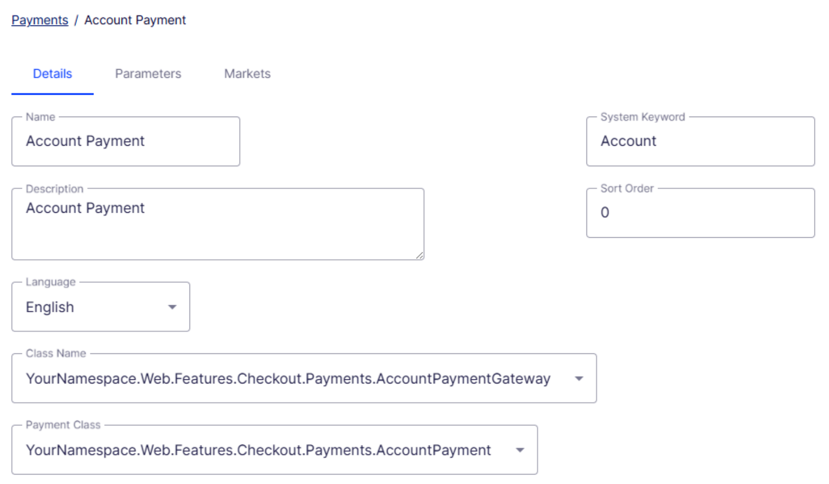
4) (Optional, advanced) Persist `AccountNumber` in its own table

⚠️ Not recommended unless you truly need a first-class table like `OrderFormPayment_CreditCard`. Commerce 14 removed the old Commerce Manager that used to create these for you; now you must do it yourself. Proceed only if you understand the risks to upgrades & maintenance.

4.1 Create a concrete `Payment` subclass

```
using System.Runtime.Serialization;
using Mediachase.Commerce.Orders;
using Mediachase.MetaDataPlus.Configurator;

namespace YourNamespace.Web.Features.Checkout
{
    [Serializable]
    public class AccountPayment : Payment
    {
        public string AccountNumber
        {
            get => this.GetString(nameof(Acco
            set => this[nameof(AccountNumber)
        }

        private static MetaClass? _metaClass;

        public static MetaClass GenericAccoun
        {
            get
            {
                _metaClass ??= MetaClass.Load
                return _metaClass;
            }
        }

        public AccountPayment() : base(Generi

        private static object? GetDefaultValu
            => "PaymentType".Equals(fieldName

        public AccountPayment(SerializationIn
        {
            this.PaymentType = PaymentType.Ot
            this.ImplementationClass = GetTyp
        }
    }
}
```

If you use this class, change your Admin **Payment Class** to `YourNamespace.Web.Features.Checkout.Payments.AccountPayment`


Payments / Account Payment

Details | Parameters | Markets

Name: Account Payment

System Keyword: Account

Description: Account Payment

Sort Order: 0

Language: English

Class Name: YourNamespace.Web.Features.Checkout.Payments.AccountPaymentGateway

Payment Class: YourNamespace.Web.Features.Checkout.Payments.AccountPayment

4.2 Create the backing tables

Run this **exact** script in your Commerce DB to create the tables and FKs:

```sql
CREATE TABLE [dbo].[OrderFormPayment_Account]
    [ObjectId] [int] NOT NULL,
    [CreatorId] [nvarchar](100) NULL,
    [Created] [datetime] NULL,
    [ModifierId] [nvarchar](100) NULL,
    [Modified] [datetime] NULL,
    [AccountNumber] [nvarchar](512) NULL,
 CONSTRAINT [PK_OrderFormPayment_Account] PRI
(
    [ObjectId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUT
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderFormPayment_Account]
REFERENCES [dbo].[OrderFormPayment] ([Payment
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[OrderFormPayment_Account]
GO

CREATE TABLE [dbo].[OrderFormPayment_Account_
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [ObjectId] [int] NOT NULL,
    [ModifierId] [nvarchar](100) NULL,
    [Modified] [datetime] NULL,
    [Language] [nvarchar](20) NOT NULL,
    [AccountNumber] [nvarchar](512) NULL,
 CONSTRAINT [PK_OrderFormPayment_Account_Loca
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUT
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderFormPayment_Account_L
REFERENCES [dbo].[OrderFormPayment] ([Payment
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[OrderFormPayment_Account_L
GO
```

4.3 Register the MetaClass and field relations

This **upserts** a `MetaClass` named `AccountPayment`, adds a `MetaField` called `AccountNumber`, mirrors standard fields used by CashCard, and relates your new field:

```sql
BEGIN TRY
    BEGIN TRAN;

    ----------------------------------------
    -- 1) Upsert MetaClass: AccountPayment
    ----------------------------------------
    DECLARE @NewMetaClassId int;

    SELECT @NewMetaClassId = mc.MetaClassId
    FROM MetaClass mc
    WHERE mc.[Namespace] = N'Mediachase.Comm
      AND mc.[Name]      = N'AccountPayment'

    IF @NewMetaClassId IS NULL
    BEGIN
        DECLARE @tNewClass table (MetaClassI

        INSERT INTO MetaClass
            (Namespace, Name, FriendlyName,
        OUTPUT INSERTED.MetaClassId INTO @tN
        SELECT TOP (1)
            N'Mediachase.Commerce.Orders.Sys
            N'AccountPayment',
            N'Account Payment',
            0,
            0,
            10,
            N'OrderFormPayment_Account',
            N'PK_OrderFormPayment_Account',
            N'Account payment'
        FROM MetaClass
        WHERE Name LIKE N'CashCard%';

        SELECT @NewMetaClassId = MetaClassId
    END;

    IF @NewMetaClassId IS NULL
    BEGIN
        INSERT INTO MetaClass
            (Namespace, Name, FriendlyName,
        VALUES
            (N'Mediachase.Commerce.Orders.Sy
             N'AccountPayment',
             N'Account Payment',
             0, 0, 10,
             N'OrderFormPayment_Account',
             N'PK_OrderFormPayment_Account',
             N'Account payment');

        SET @NewMetaClassId = SCOPE_IDENTITY
    END;

    ----------------------------------------
    -- 2) Upsert MetaField: AccountNumber
    ----------------------------------------
    DECLARE @NewMetaFieldId int;

    SELECT @NewMetaFieldId = mf.MetaFieldId
    FROM MetaField mf
    WHERE mf.[Name] = N'AccountNumber'
      AND mf.[Namespace] = N'Mediachase.Comm

    IF @NewMetaFieldId IS NULL
    BEGIN
        INSERT INTO MetaField
            (Name, Namespace, SystemMetaClas
             DataTypeId, [Length], AllowNull
        VALUES
            (N'AccountNumber', N'Mediachase.
             N'Cash Card Number', N'Contains
             31, 512, 1, 0, 0, 0, 0);

        SET @NewMetaFieldId = SCOPE_IDENTITY
    END;

    ----------------------------------------
    -- 3) Relations: MetaClassMetaFieldRelat
    ----------------------------------------
    -- 3a) Mirror fields related to any *Cas
    INSERT INTO MetaClassMetaFieldRelation (
    SELECT DISTINCT
        @NewMetaClassId,
        f.MetaFieldId,
        0,
        1
    FROM MetaField f
    INNER JOIN MetaClassMetaFieldRelation r
    INNER JOIN MetaClass c ON c.MetaClassId
    WHERE c.[Name] LIKE N'%CashCard%'
      AND f.SystemMetaClassId = 10
      AND NOT EXISTS (
            SELECT 1
            FROM MetaClassMetaFieldRelation
            WHERE r2.MetaClassId = @NewMetaC
              AND r2.MetaFieldId = f.MetaFie
        );

    -- 3b) Ensure the new "AccountNumber" fi
    IF NOT EXISTS (
        SELECT 1
        FROM MetaClassMetaFieldRelation
        WHERE MetaClassId = @NewMetaClassId
          AND MetaFieldId = @NewMetaFieldId
    )
    BEGIN
        INSERT INTO MetaClassMetaFieldRelati
        VALUES (@NewMetaClassId, @NewMetaFie
    END;

    -- 3c) Run the metadata proc generator f
    EXEC mdpsp_sys_CreateMetaClassProcedure

    COMMIT TRAN;
END TRY
BEGIN CATCH
    IF XACT_STATE() <> 0 ROLLBACK TRAN;
    DECLARE @Err nvarchar(2048) = ERROR_MESS
    RAISERROR('Provisioning AccountPayment f
END CATCH;
GO
```

4.4 Hook deletion into `ecf_OrderForm_Delete`

This step modifies a stock proc to delete your custom rows. Not recommended—but if you mirror the legacy pattern, you must add your delete proc call:

```
-- inside the "Delete payments" cursor loop:
EXEC [dbo].[mdpsp_avto_OrderFormPayment_Accoun
-- (the built-in ones are already there:)
EXEC [dbo].[mdpsp_avto_OrderFormPayment_CashCa
EXEC [dbo].[mdpsp_avto_OrderFormPayment_Credit
EXEC [dbo].[mdpsp_avto_OrderFormPayment_GiftCa
EXEC [dbo].[mdpsp_avto_OrderFormPayment_Invoic
EXEC [dbo].[mdpsp_avto_OrderFormPayment_Other_
EXEC [dbo].[mdpsp_avto_OrderFormPayment_Exchan
```