ЧАСТЬ II Разработка приложений для SAP S/4HANA

# Глава 4. Разработка элементов SAP Fiori List Report и Object Page

*В предыдущей главе вы узнали о новой модели программирования ABAP RESTful. Теперь в этой главе вы научитесь создавать неуправляемое транзакционное приложение на основе шаблонов элементов SAP Fiori с использованием модели программирования ABAP RESTful. Мы подробно проиллюстрируем сквозной (E2E) процесс разработки с помощью пошаговых инструкций. Вы создадите таблицы базы данных и представления основных служб данных (CDS), добавите неуправляемую реализацию для обеспечения обработки транзакций в бизнес-объекте, добавите элементы пользовательского интерфейса (UI), а также создадите определение службы и привязку службы.*

В этой главе мы будем использовать бизнес-сценарий и дизайн, разработанный в главе 2, раздел 2.4.3, для создания приложения для управления документами покупки. Напоминаем, что дизайн приложения состоял из отчета со списком, отображающего документы покупки в таблице, и страницы объекта, предоставляющей подробную информацию о документе покупки, включая отдельные элементы документа покупки, которые можно предоставить с помощью быстрого действия.

Используя этот бизнес-сценарий, эта глава предоставит вам руководство по разработке E2E для разработки приложений списка отчетов и страниц объектов на основе новой модели программирования ABAP RESTful для SAP Fiori. В первом издании этой книги мы рассмотрели создание приложения элементов SAP Fiori с использованием концепций Business Object Processing Framework (BOPF) для транзакционных действий и создали службу OData для использования данных из серверной части с помощью клиента SAP Gateway. Эти две части являются основным отличием, когда речь идет о разработке приложений с использованием модели программирования ABAP RESTful. В модели программирования ABAP RESTful концепция BOPF полностью заменена, и вместо обычного создания службы OData с использованием SAP Gateway в этой главе вы создадите определение службы и привязку службы.

Для создания нашего приложения документа о покупке мы сначала создадим набор базовых, составных и потребительских представлений CDS вместе с необходимыми аннотациями пользовательского интерфейса, а затем воспользуемся преимуществами определения службы и привязки службы для регистрации службы OData и открыть представление CDS. Затем мы добавим транзакционное поведение через определение поведения. (Подробнее об определениях поведения см. в главе 3.)

На рис. 4.1 показана первая часть нашего примерного приложения, где мы создадим все представления интерфейса и составные представления. Позже мы свяжем с этими составными представлениями наши определения поведения, чтобы добавить транзакционные операции, такие как создание, чтение, обновление и удаление (CRUD) и быстрые действия, а затем мы предоставим их как службу OData с помощью определения службы и привязки службы.

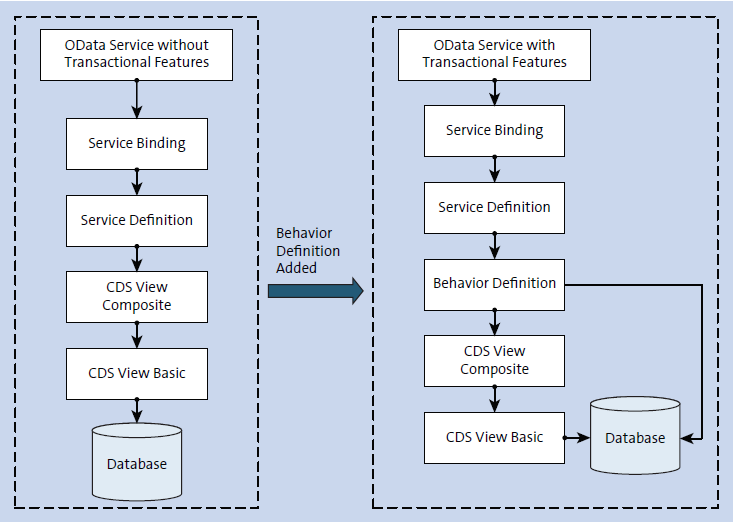


Рис. 4.1 Создание базовых представлений и последующее добавление определений поведения для включения транзакционных действий

Во второй части мы создадим представление проекции (тип VDM — потребление) поверх составного представления с добавлением к ним необходимых транзакционных возможностей через определение поведения. Эта практика позволит нам повторно использовать наши бизнес-объекты (представления, которые мы будем включать функции CRUD с помощью определений поведения) в нескольких приложениях. Нам также потребуется создать определение поведения проекции, чтобы мы могли повторно использовать базовое определение поведения, связанное с нашими бизнес-объектами.

На рис. 4.2 показано, как представления проекций и определения поведения проекций вступают в общую картину и позволяют повторно использовать базовые бизнес-объекты и определения поведения.

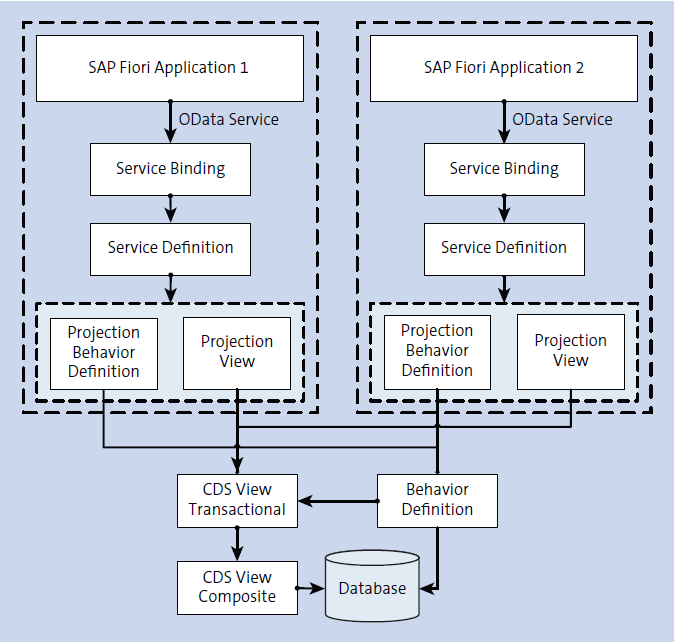


Рис. 4.2 Представления Projection CDS и определения поведения Projection, обеспечивающие возможность повторного использования бизнес-объектов и определения поведения

Теперь давайте начнем с создания необходимых базовых представлений для нашего примера.

## 4.1 Core Data Services

В этом разделе вы узнаете, как создать многоуровневую модель данных CDS в соответствии с рекомендациями по моделированию виртуальной модели данных (VDM) SAP S/4HANA, включая модульные тесты для вычисляемых значений полей с использованием модуля ABAP и платформы CDS Test Double Framework.

Прежде чем приступить к разработке, необходимо сначала установить средства разработки ABAP, поскольку Eclipse — это стандартная интегрированная среда разработки (IDE) для разработки представлений CDS и современного кода ABAP. Затем вы начнете разработку приложения снизу вверх, сначала представив модель данных и ее таблицы базы данных. Основываясь на этой основе, вы создадите несколько уровней представлений CDS, начиная с уровня базового интерфейса в качестве простого уровня проекции поверх таблиц базы данных. Используя базовый слой представления интерфейса в качестве основы, вы будете постепенно добавлять больше представлений и вычислений, специфичных для приложения, включая тесты.

### 4.1.1 Инструменты разработки ABAP в Eclipse

Для разработки современных ABAP-приложений на основе модели программирования ABAP для SAP Fiori вы больше не можете использовать классический ABAP Development Workbench (транзакция SE80), поскольку разработка моделей данных на основе CDS не поддерживается старыми инструментами. Поэтому разработчики должны загрузить и установить последнюю версию Eclipse, например, Eclipse Oxygen (IDE для разработчиков Java) с веб-сайта Eclipse Foundation, а также установить внешний интерфейс инструментов разработки ABAP в качестве подключаемого модуля Eclipse. Чтобы настроить среду, откройте Eclipse и в меню «Справка» выберите «Установить новое программное обеспечение», как показано на рис. 4.3.

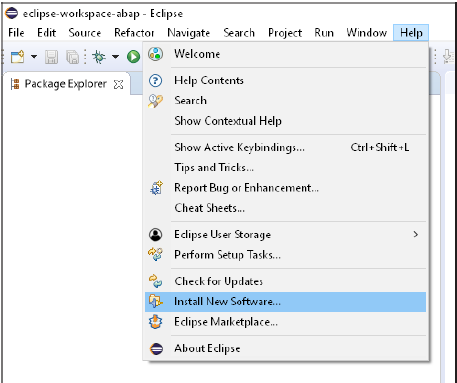


Рис. 4.3. Запуск процедуры установки средств разработки ABAP (ADT) в Eclipse

Откроется диалоговое окно установки Eclipse. Затем вы должны добавить сайт обновления средств разработки ABAP в качестве нового места назначения репозитория, как показано на рис. 4.4. Соответствующий URL-адрес установки инструментов разработки ABAP для текущей версии Eclipse доступен на веб-сайте инструментов разработки SAP по адресу <http://s-prs.co/498814>.

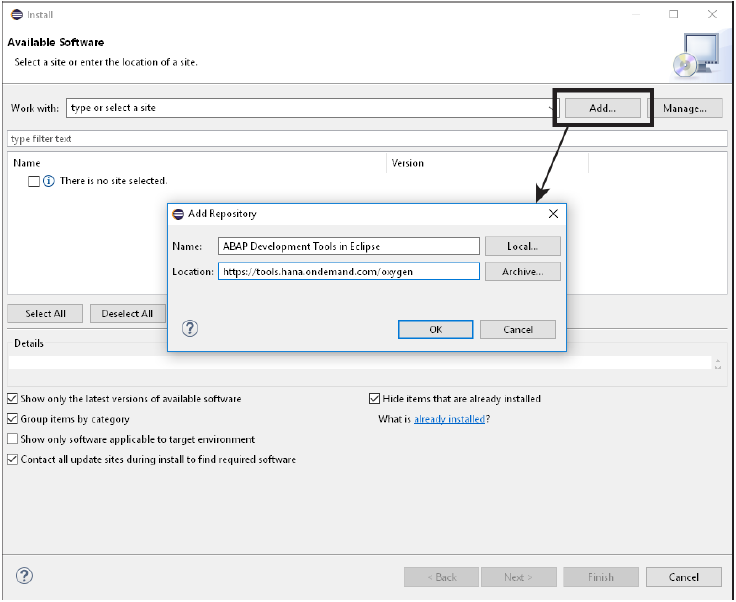


Рисунок 4.4 Добавление инструментов разработки ABAP в репозиторий Eclipse

Для разработки приложений с использованием модели программирования ABAP RESTful для SAP Fiori требуются только инструменты разработки ABAP для SAP NetWeaver, поэтому выберите только этот программный пакет, как показано на рис. 4.5. Нажмите «Далее», примите условия лицензионного соглашения и нажмите «Готово». После завершения установки Eclipse потребует перезагрузки.

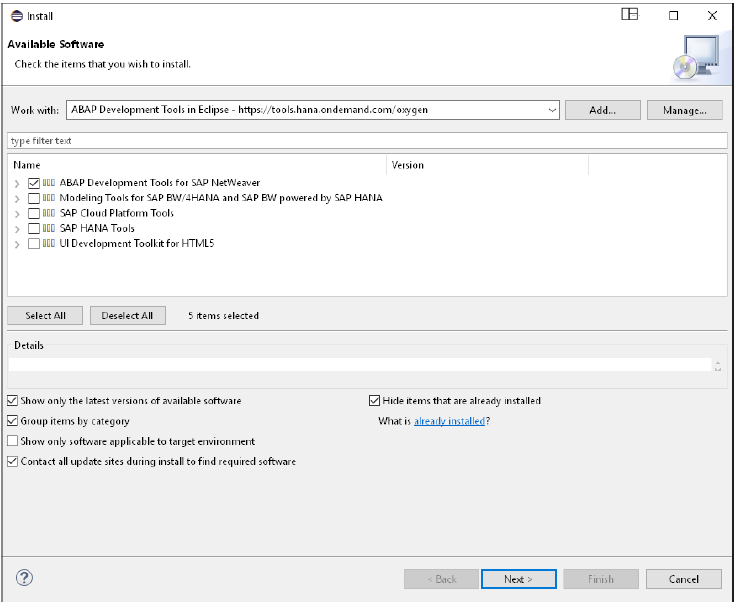


Рис. 4.5 Выбор инструментов разработки ABAP для SAP NetWeaver

Чтобы подключиться к внутреннему серверу SAP NetWeaver ABAP, сначала откройте теперь доступную перспективу ABAP, щелкнув **Window - Perspective - Open Perspective - Other… - ABAP**. С этой точки зрения вы создадите проект ABAP, который установит соединение с внутренним сервером. На рис. 4.6 показано, как можно запустить процесс создания проекта ABAP, щелкнув **File — New — ABAP Project** в строке меню Eclipse. Новый проект можно создать одним из следующих способов:

- Из предопределенного подключения к системе входа в систему SAP

- Путем определения ручного подключения к системе

- Путем дублирования существующего системного соединения

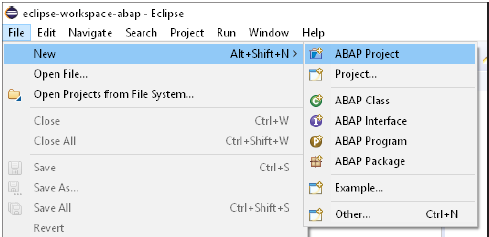


Рис. 4.6 Создание проекта ABAP с точки зрения ABAP в Eclipse

*Примечание*

*Чтобы включить серверную часть ABAP для использования инструментов разработки ABAP в Eclipse, см. руководство «Настройка серверной части ABAP для инструментов разработки ABAP», предоставленное SAP по адресу* [*http://s-prs.co/498815*](http://s-prs.co/498815)*.*

### 4.1.2 Введение в модель данных

Первым шагом к разработке любого приложения является создание модели данных, за которой следует создание фактических таблиц базы данных для сохранения транзакционных или основных данных. Поэтому в следующих разделах мы познакомим вас с базовой моделью данных для приложения и создадим необходимые таблицы базы данных.

Модель отношений сущностей Концептуальная модель данных приложения в форме модели отношений сущностей (ERM) показана на рис. 4.7. Эта модель данных состоит из типа сущности документа покупки и типа сущности позиции документа покупки. Документы покупки имеют отношение от 0 до n к товарам, а товары имеют отношение от 1 до 1 к документам, что составляет отношение родитель-потомок (документы покупки как родительские узлы и позиции документа покупки как дочерние узлы).

Сущность документа покупки имеет несколько атрибутов, включая общий статус документа и приоритет покупки, а также ответственную закупочную организацию. Первичный ключ документа покупки основан на единственном идентификаторе документа покупки. Сущность позиции документа покупки представляет фактические позиции покупки, включая цену и количество. Первичный ключ сущности элемента документа покупки состоит из идентификатора товара и идентификатора документа покупки, который ссылается на идентификатор документа покупки в сущности документа покупки посредством отношения внешнего ключа.

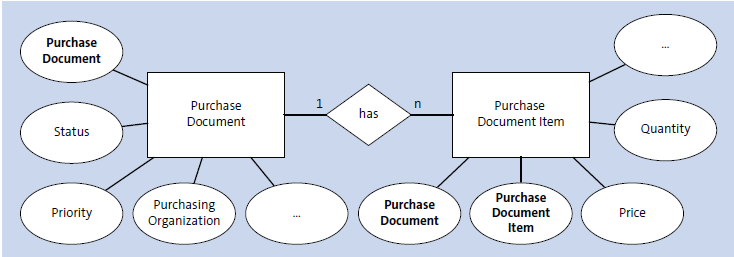


Рисунок 4.7 Концептуальная модель данных приложения, представленного в виде ERM

#### Таблицы базы данных

Основу любого приложения в SAP S/4HANA составляют таблицы базы данных, в которых данные хранятся в базе данных SAP HANA. Новые таблицы базы данных можно создавать в Eclipse с помощью редактора исходного кода, доступного, начиная с SAP NetWeaver Application Server для ABAP 7.52 SP 00 (SAP NetWeaver AS для ABAP). Чтобы сохранить описанные сущности документа покупки и позиции документа покупки и их значения в базе данных, вам потребуется создать две отдельные таблицы базы данных.

Для ***создания прозрачной таблицы в ABAP-словаре*** необходимо выполнить следующие шаги:

1. Откройте перспективу ABAP в инструментах разработки ABAP в Eclipse.

2. В проекте ABAP выберите узел пакета, в котором вы хотите создать таблицу, в представлении Project Explorer.

3. Откройте контекстное меню, щелкнув пакет правой кнопкой мыши и выбрав **«Создать» — «Другой объект репозитория ABAP» — «Словарь» — «Таблица базы данных»**.

4.Откроется мастер создания таблицы базы данных. Вы должны ввести имя и описание, как показано на рис. 4.8. В нашем случае мы создадим таблицу для документов покупки с именем ZPURCHDOCUMENT и таблицу для позиций документов покупки с именем ZPURCHDOCITEM.

5. Наконец, открывается редактор исходного кода таблицы базы данных, где вы можете добавить поля в таблицу и настроить ее свойства.

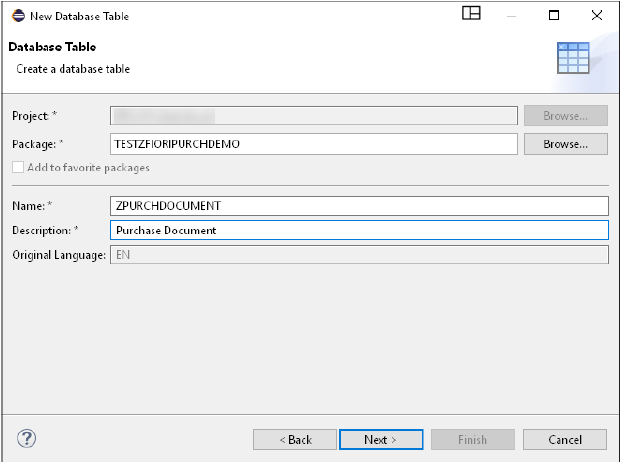


Рисунок 4.8 Мастер создания таблиц базы данных

Язык программирования редактора исходного кода очень похож на язык определения данных (DDL) CDS. Метаданные, определенные с помощью аннотаций и технических свойств, таких как поля, определяются в синтаксисе, подобном SQL, как показано в листинге 4.1. При сохранении или активации фактическое постоянство обновляется. Исходный код, отображаемый в редакторе, не является новым переносимым объектом разработки. Вся информация хранится и извлекается из классического словаря ABAP. Исходный код определения таблицы базы данных для сохраняемых документов покупки показан на рис. 4.9 и в листинге 4.1.

Используя редактор на основе исходного кода в Eclipse, вы можете определить как технические, так и семантические атрибуты таблицы с помощью аннотаций. Например, с помощью аннотации *@EndUserText.label* вы можете предоставлять понятные и переводимые метки и описания для объектов и полей. Для полей эта аннотация будет учитываться только в том случае, если полю не назначен элемент данных; в противном случае метка поля берется из элемента данных. Для простоты мы создали элемент данных только для поля закупочного документа (zpurchasedocumentdtel), так как этот элемент данных требуется для последующего обращения к полю через аннотацию внешнего ключа в таблице позиций документа закупки. Остальные поля набираются с использованием встроенных типов ABAP, например, *abap.char(4)*.

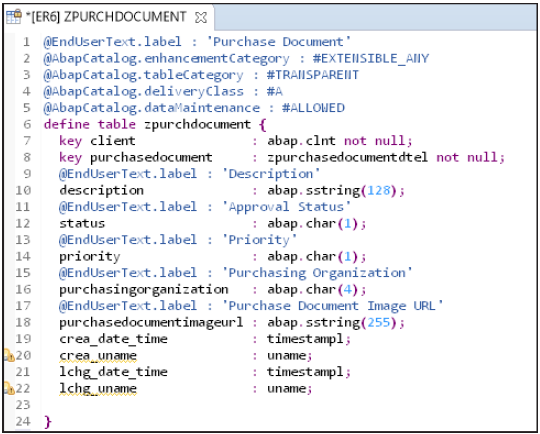


Рисунок 4.9 Редактор таблиц на основе исходного кода в Eclipse

Техническая аннотация ***@AbapCatalog.enhancementCategory: #EXTENSIBLE\_ANY*** в части заголовка определения таблицы указывает, что все компоненты структуры и базы данных, а также их расширения могут содержать компоненты, тип данных которых может относиться к любой категории.

Категория таблиц ***@AbapCatalog.tableCategory***: ***#TRANSPARENT*** является категорией по умолчанию для таблиц базы данных, и при активации будет создана как таблица базы данных в базе данных, так и идентичный вариант в классическом словаре ABAP.

Класс доставки ***@AbapCatalog.deliveryClass***: ***#A*** указывает, что таблица представляет собой таблицу приложений, содержащую либо транзакционные, либо основные данные, и заполняется прикладными программами.

Аннотация семантического обслуживания данных ***@AbapCatalog.dataMaintenance***: ***#ALLOWED*** делает таблицу доступной для редактирования с помощью браузера данных в транзакции SE16.

@EndUserText.label : 'Purchase Document'

@AbapCatalog.enhancementCategory : #EXTENSIBLE\_ANY

@AbapCatalog.tableCategory : #TRANSPARENT

@AbapCatalog.deliveryClass : #A

@AbapCatalog.dataMaintenance : #ALLOWED

define table zpurchdocument {

key client : abap.clnt not null;

key purchasedocument : zpurchasedocumentdtel not null;

@EndUserText.label : 'Description'

description : abap.sstring(128);

@EndUserText.label : 'Approval Status'

status : abap.char(1);

@EndUserText.label : 'Priority'

priority : abap.char(1);

@EndUserText.label : 'Purchasing Organization'

purchasingorganization : abap.char(4);

@EndUserText.label : 'Purchase Document Image URL'

purchasedocumentimageurl : abap.sstring(255);

crea\_date\_time : timestampl;

crea\_uname : uname;

lchg\_date\_time : timestampl;

lchg\_uname : uname;

}

Листинг 4.1. Определение таблицы базы данных для документов покупки

Определение таблицы базы данных для позиций документа покупки показано в листинге 4.2. В отличие от таблицы документов покупки, определение таблицы базы данных имеет составной первичный ключ, состоящий из идентификатора приобретенного документа и идентификатора приобретенного документа. Позиции документа покупки и таблицы документов покупки связаны отношением внешнего ключа в поле идентификатора купленного документа. Таблица содержит поля для цены товара и количества для покупки, а также поля для соответствующей валюты и единицы измерения. Эти поля связаны с помощью семантических аннотаций: ***@Semantics.amount.currencyCode*** для денежных значений и ***@Semantics.quantity.unitOfMeasure*** для количеств.

@EndUserText.label : 'Purchase Document Item'

…

define table zpurchdocitem {

key client : abap.clnt not null;

@EndUserText.label : 'Purchase Document Item'

key purchasedocumentitem : abap.char(10) not null;

@AbapCatalog.foreignKey.keyType : #KEY

@AbapCatalog.foreignKey.screenCheck : true

key purchasedocument : zpurchasedocumentdtel not null

with foreign key [1..\*,1] zpurchdocument

where client = zpurchdocitem.client

and

purchasedocument = zpurchdocitem. purchasedocument;

@EndUserText.label : 'Description'

description : abap.sstring(128);

@EndUserText.label : 'Price'

@Semantics.amount.currencyCode : 'zpurchdocitem.currency'

price : abap.curr(13,2);

@EndUserText.label : 'Currency'

currency : abap.cuky;

@EndUserText.label : 'Quantity'

@Semantics.quantity.unitOfMeasure: 'zpurchdocitem.quantityunit'

quantity : abap.quan(13,2);

@EndUserText.label : 'Unit'

quantityunit : abap.unit(3);

@EndUserText.label : 'Vendor'

vendor : abap.sstring(32);

@EndUserText.label : 'Vendor Type'

vendortype : abap.sstring(32);

@EndUserText.label : 'Purchase Document Item Image URL'

purchasedocumentitemimageurl : abap.sstring(255);

crea\_date\_time : timestampl;

crea\_uname : uname;

lchg\_date\_time : timestampl;

lchg\_uname : uname;

}

Листинг 4.2. Определение таблицы базы данных для позиций документа покупки

*Примечание*

*Вы можете использовать сочетание клавиш (Ctrl)+(Shift)+(A) для быстрого открытия объектов разработки ABAP в Eclipse, включая пакеты, таблицы базы данных или представления CDS.*

### 4.1.3 Создание Basic Interface CDS View

В предыдущем разделе мы определили модель данных и базовые таблицы базы данных нашего приложения для сохранения документов покупки и их элементов. В этом разделе мы поместим семантический и многократно используемый уровень базового интерфейса CDS поверх этих ранее созданных таблиц. Мы будем придерживаться стандартных рекомендаций по разработке SAP S/4HANA VDM на основе представлений CDS.

Уровень базового интерфейса является базовым уровнем представлений CDS и развертывается непосредственно поверх таблиц базы данных. Как описано в главе 1, этот уровень не предназначен для выполнения каких-либо сложных вычислений или объединения больших наборов данных; вместо этого этот интерфейсный уровень просто проецирует поля, предоставляемые базовыми таблицами базы данных, чтобы обеспечить детальное повторное использование этих представлений.

Чтобы ***создать представление CDS*** с помощью средств разработки ABAP в Eclipse, выполните следующие действия.

1. Откройте перспективу ABAP в инструментах разработки ABAP в Eclipse.

2. В проекте ABAP выберите узел пакета, в котором вы хотите создать таблицу, с помощью представления Project Explorer.

3. Откройте контекстное меню, щелкнув пакет правой кнопкой мыши и выбрав ***«Создать» — «Другой репозиторий ABAP» — «Основные службы данных» — «Определение данных»***.

4. В появившемся мастере создания CDS введите Имя и Описание. В нашем случае мы назовем представление документов покупки Z\_I\_PurchaseDocument, а представление позиций документа покупки Z\_I\_PurchaseDocumentItem.

5. Выберите один из предоставленных шаблонов для создания скелетного представления CDS, например, Define View, который создаст простой каркас проекционного представления с одним единственным источником данных и без каких-либо ассоциаций или объединений.

6. Нажмите кнопку Готово в мастере, и откроется редактор исходного кода, чтобы вы могли добавить в представление поля или ассоциации.

*Примечание*

*Вы можете использовать сочетание клавиш (Ctrl)+(Пробел) в разделе выбора полей в редакторе исходного кода представления CDS, чтобы открыть всплывающее окно помощника по содержимому Eclipse, позволяющее легко выбирать поля базовой таблицы базы данных или представления CDS, как показано на рис. 4.10.*

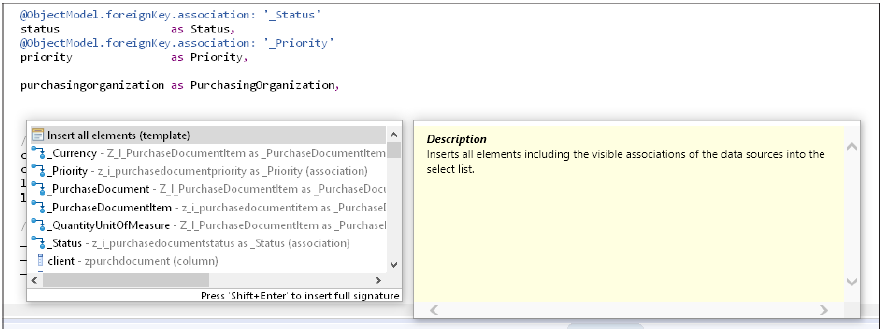


Рис. 4.10 Запуск помощника по содержимому Eclipse для полей CDS с помощью сочетания клавиш

В листинге 4.3 показано базовое представление интерфейса для документов покупки, которое создает простую проекцию таблицы документов покупки (таблица zpurchdocument) для отображения всех полей таблицы и предоставления псевдонимов для полей, имеющих отношение к конечным пользователям. Чтобы следовать соглашениям об именовании представлений уровня интерфейса в соответствии с рекомендациями по разработке SAP S/4HANA VDM, мы назвали представления, используя следующий шаблон: Z\_I\_<CDSViewName>. ***Z\_*** указывает, что это специально разработанное представление в пространстве имен клиента, а ***I\_*** указывает, что это представление является представлением уровня интерфейса.

Представление также снабжено аннотацией ***@VDM.viewType: #BASIC***, которая присваивает представление базовому уровню представления интерфейса VDM.

Помимо постоянных полей базы данных, представление также содержит несколько ассоциаций, которые распространяются среди потенциальных потребителей этого представления в конце списка выбора полей. По соглашению ассоциации начинаются со знака подчеркивания, например, \_PurchaseDocumentItem. Обычно большинство (ключевых) полей, содержащихся в представлении, будут иметь ассоциации внешнего ключа с другими представлениями, поскольку предполагается, что данные хранятся в нормализованной и свободной от избыточности форме. В некоторых случаях мы опускаем это для простоты, но, например, мы добавили ассоциации к представлениям Priority, Status и PurchasingOrganization. В таблице документов покупки будут храниться только ключи Priority, Status и PurchasingOrganization; однако дополнительные данные, такие как тексты, могут быть прочитаны через ассоциации внешнего ключа с этими представлениями.

Другой важной аннотацией является проверка авторизации ***@AccessControl.authorizationCheck: #CHECK***, которая разрешает ограниченный доступ к представлению CDS с использованием контроля доступа на языке управления данными (DCL). Мы подробно обсудим ограничения доступа для представлений CDS в разделе 4.1.5.

Аннотация ***@ObjectModel.representativeKey*** определяет поле первичного ключа, специфичное для этого представления, а ***@ObjectModel.semanticKey*** определяет первичный ключ этого представления. Для базового представления интерфейса документа покупки это поле является полем PurchaseDocument в обоих случаях. Поля, содержащие описания или текст, должны быть аннотированы ***@Semantics.text: true***. Основное поле описания сущности можно связать с ключом сущности с помощью аннотации ***@ObjectModel.text.element: ['Description']***.

@AbapCatalog.sqlViewName: 'ZIPURCHDOC'

@EndUserText.label: 'Purchase Document'

@AccessControl.authorizationCheck: #CHECK

@VDM.viewType: #BASIC

@ObjectModel.representativeKey: 'PurchaseDocument'

@ObjectModel.semanticKey: ['PurchaseDocument']

define view Z\_I\_PurchaseDocument

as select from zpurchdocument

association [0..\*] to Z\_I\_PurchaseDocumentItem as \_PurchaseDocumentItem

on $projection.PurchaseDocument = \_PurchaseDocumentItem.PurchaseDocument

association [0..1] to Z\_I\_PurchaseDocumentPriority as \_Priority

on $projection.Priority = \_Priority.Priority

association [0..1] to Z\_I\_PurchaseDocumentStatus as \_Status

on $projection.Status = \_Status.Status

association [0..1] to Z\_I\_PurchasingOrganization as \_PurchasingOrganization

on $projection.PurchasingOrganization = \_PurchasingOrganization.PurchasingOrganization

{

@ObjectModel.text.element: ['Description']

key purchasedocument as PurchaseDocument,

@Semantics.text: true

description as Description,

@ObjectModel.foreignKey.association: '\_Status'

status as Status,

@ObjectModel.foreignKey.association: '\_Priority'

priority as Priority,

@ObjectModel.foreignKey.association: '\_PurchasingOrganization'

purchasingorganization as PurchasingOrganization,

@Semantics.imageUrl: true

purchasedocumentimageurl as PurchaseDocumentImageURL,

crea\_date\_time,

crea\_uname,

lchg\_date\_time,

lchg\_uname,

// Associations

\_PurchaseDocumentItem,

\_Priority,

\_Status,

\_PurchasingOrganization

}

Листинг 4.3. Представление основного интерфейса в верхней части таблицы документов покупки (zpurchdocument)

Базовый интерфейс элемента документа покупки показан в листинге 4.4. Его репрезентативным ключом является поле PurchaseDocumentItem, которое является наиболее конкретной частью его первичного ключа и определяет этот объект. Семантический ключ состоит из полей PurchaseDocument и PurchaseDocumentItem и отражает первичный ключ объекта. Представление базового интерфейса элемента документа покупки содержит ассоциацию внешнего ключа с представлением документа покупки, а также для повторного использования представлений для валюты (I\_Currency) и единицы измерения (I\_UnitOfMeasure), предоставляемых стандартным слоем представления базового интерфейса SAP S/4HANA. Как только SAP явным образом выпустит представления, вы сможете использовать их в собственных разработках без риска внесения изменений, нарушающих представление в будущих выпусках. Аннотации семантической валюты и единицы измерения должны быть известны из определения таблицы. Мы добавили аннотацию ***@DefaultAggregation:#NONE*** к полям «Цена» и «Количество», потому что все поля сумм будут обрабатываться по умолчанию как меры и агрегироваться SADL или аналитическим механизмом. Однако в нашем случае мы явно хотим подавить это поведение по умолчанию.

@AbapCatalog.sqlViewName: 'ZIPURCHDOCITEM'

@EndUserText.label: 'Purchase Document Item'

@AccessControl.authorizationCheck: #NOT\_REQUIRED

@VDM.viewType: #BASIC

@ObjectModel.representativeKey: 'PurchaseDocumentItem'

@ObjectModel.semanticKey: ['PurchaseDocumentItem','PurchaseDocument']

define view Z\_I\_PurchaseDocumentItem

as select from zpurchdocitem

association [1..1] to Z\_I\_PurchaseDocument as \_PurchaseDocument on

$projection.PurchaseDocument = \_PurchaseDocument.PurchaseDocument

association [0..1] to I\_UnitOfMeasure as \_QuantityUnitOfMeasure on

$projection.QuantityUnit = \_QuantityUnitOfMeasure.UnitOfMeasure

association [0..1] to I\_Currency as \_Currency on

$projection.Currency = \_Currency.Currency

{

@ObjectModel.text.element: ['Description']

key purchasedocumentitem as PurchaseDocumentItem,

@ObjectModel.foreignKey.association: '\_PurchaseDocument'

key purchasedocument as PurchaseDocument,

@Semantics.text: true

description as Description,

vendor as Vendor,

vendortype as VendorType,

@Semantics.amount.currencyCode: 'Currency'

@DefaultAggregation: #NONE

price as Price,

@Semantics.currencyCode: true

@ObjectModel.foreignKey.association: '\_Currency'

currency as Currency,

@Semantics.quantity.unitOfMeasure: 'QuantityUnit'

@DefaultAggregation: #NONE

quantity as Quantity,

@Semantics.unitOfMeasure: true

@ObjectModel.foreignKey.association: '\_QuantityUnitOfMeasure'

quantityunit as QuantityUnit,

@Semantics.imageUrl: true

purchasedocumentitemimageurl as PurchaseDocumentItemImageURL,

crea\_date\_time,

crea\_uname,

lchg\_date\_time,

lchg\_uname,

// Associations

\_PurchaseDocument,

\_QuantityUnitOfMeasure,

\_Currency

}

Листинг 4.4. Представление основного интерфейса в верхней части таблицы позиций документа покупки (zpurchdocitem)

### 4.1.4 Добавление вычисляемых полей

С помощью CDS DDL вы можете динамически вычислять поля на основе других постоянных полей, уже имеющихся в базе данных. В прошлом такие расчетные значения могли рассчитываться на сервере приложений или, что еще хуже, предварительно вычислялись и сохранялись в избыточных сводных таблицах. Однако с запуском SAP HANA этот подход устарел, потому что теперь вы можете легко агрегировать и вычислять значения на лету, что уменьшает общий объем данных системы и повышает производительность за счет переноса вычислений и агрегирования с интенсивным использованием данных на уровень базы данных. . В следующих разделах мы вычислим несколько значений для нашего приложения, которые не нужно хранить в базе данных, что приведет к избыточности. Вместо этого эти значения могут быть рассчитаны динамически с использованием представлений CDS во время выполнения.

#### Расчет Overall Item Price

Простым примером вычисляемого значения является общая цена позиции документа покупки. Допустим, мы сохранили в базе данных цену отдельного товара и запрошенное количество, но не общую цену. Чтобы рассчитать общую цену, вы можете просто умножить запрошенное количество на цену одного товара и указать новое имя для поля, используя псевдоним. Например, при цене товара 50 и количестве 10 общая цена этого товара будет 50 \* 10 = 500. Как можно реализовать этот расчет в базовом представлении Z\_I\_PurchaseDocumentItem, показано в листинге 4.5:

…

@Semantics.amount.currencyCode: 'Currency'

@DefaultAggregation: #NONE

quantity \* price as OverallItemPrice,

…

Листинг 4.5. Реализация расчета в представлении позиции документа покупки

Хотя это поле является вычисляемым полем, вы также можете ссылаться на уже существующее поле валюты.

#### Расчет Overall Purchase Document Price путем введения Composite Interface View и использования разработки через тестирование

Кроме того, допустим, нам нужна общая цена документа о покупке. Чтобы рассчитать общую цену, вы будете использовать ранее рассчитанную общую цену товара, поскольку общую цену документа покупки можно рассчитать путем суммирования всех общих цен товаров, относящихся к его назначенным товарам. Чтобы выполнить этот расчет, вы должны поместить дополнительное представление поверх базового представления документа покупки, которое мы назовем Z\_I\_PurchDocOverallPrice и назначим составному интерфейсному слою VDM, так как это дополнительное представление будет содержать расчет на основе базового уровня интерфейса CDS. просмотров, но по-прежнему предназначен для повторного использования. А пока просто добавьте все поля базового представления документа покупки в список выбора нового представления.

Поскольку расчет общей цены уже является расчетом средней сложности, вам следует разработать этот расчет с использованием разработки через тестирование (TDD), чтобы проверить и защитить разработанную логику автоматизированным и воспроизводимым способом.

Разработка тестов на основе тестов означает, что перед выполнением фактических вычислений вы реализуете тесты. TDD позволит вам поэтапно выполнять вычисления с уверенностью, что уже реализованная логика предыдущих шагов не нарушит последующие шаги или выпуски. Этот подход также имеет то преимущество, что вводит тестирование на ранней стадии процесса разработки, чтобы автоматизация тестирования не была просто пропущена после того, как функция заработала. В конце концов, у вас также будет подстраховка для будущих изменений, которые могут привести к регрессионным ошибкам. Модульные тесты для представлений CDS основаны на модуле ABAP и платформе CDS Test Double. С помощью CDS Test Double Framework вы можете динамически создавать и настраивать тестовые заглушки или двойники для представлений или таблиц, от которых зависит тестируемое представление CDS.

Создавать модульные тесты для представлений CDS несложно, поскольку средства разработки ABAP в Eclipse уже обеспечивают хорошую поддержку модульных тестов. Как показано на рисунке 4.11, просто щелкните правой кнопкой мыши представление CDS и создайте новый глобальный класс модульного теста ABAP для этого представления, выбрав ***New Test Class***.

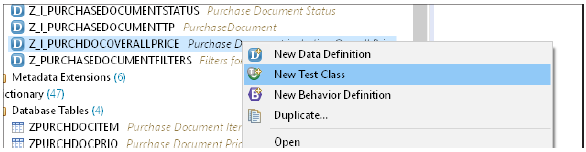


Рис. 4.11. Создание каркасного класса модульного теста для представления CDS в Eclipse Project Explorer.

Откроется диалоговое окно мастера создания теста. На первом экране мастера необходимо указать имя глобального тестового класса ABAP, пакет и описание, как показано на рис. 4.12. Доступное только для чтения поле CDS Under Test указывает, для какого представления CDS вы создаете тест. Хорошей практикой является размещение тестов рядом с тестируемым кодом, чтобы по умолчанию тесты помещались в тот же пакет, что и тестируемое представление CDS.

После заполнения этих полей перейдите к следующему экрану мастера, нажав кнопку «Далее».

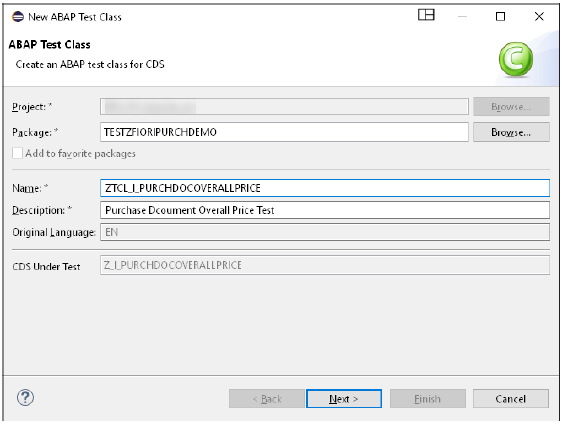


Рис. 4.12 Создание нового тестового класса для представления CDS общей цены документов покупки (Z\_I\_PurchDocOverallPrice)

На следующем экране вы выберете представления или таблицы CDS, от которых зависит ваше тестируемое представление и которые необходимы в качестве тестовых двойников для тестовых случаев, которые вы хотите реализовать, как показано на рисунке 4.13. Во время выполнения теста тестовые двойники заменяют фактические представления или таблицы базы данных и динамически предоставляют разные тестовые данные для разных тестовых случаев. Как правило, существует два типа тестов: модульный тест, учитывающий только зависимости первого уровня, и иерархический тест, позволяющий создавать двойные тесты для всего стека представлений CDS, от которых зависит тестируемое представление. В нашем случае просто нажмите «Далее», потому что мы создаем модульный тест, и оба базовых представления интерфейса, от которых зависит наше тестируемое представление (Z\_I\_PurchaseDocument и Z\_I\_PurchaseDocumentItem), уже предварительно выбраны и требуются в качестве зависимостей для двойного создания в наших тестах.

В дополнение к созданию глобального класса модульного теста ABAP и предоставлению необходимых зависимостей для создания тестового двойника мастер создания тестового класса также поддерживает создание тестовых двойников с простым созданием тестовых данных на основе форм, как показано на рис. 4.14. Для нашего первого теста, который мы назовем total\_price\_no\_items, мы не будем создавать никаких тестовых данных для позиций документов покупки, а только одну запись в таблице для документов покупки.

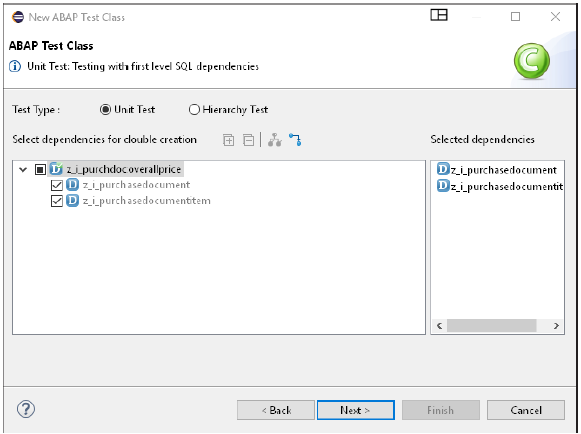


Рисунок 4.13 Экран выбора зависимости для теста CDS

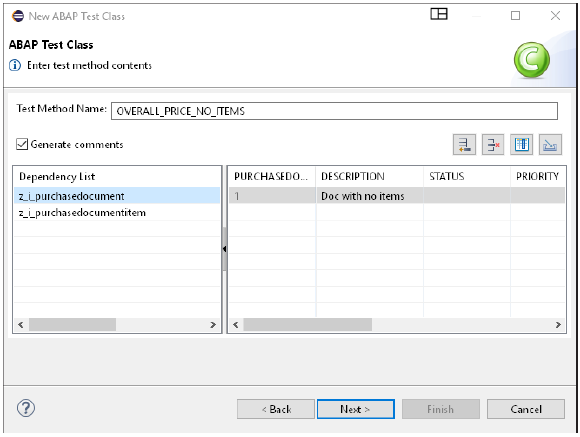


Рис. 4.14 Создание тестовых данных на основе форм для тестовых двойников CDS, от которых зависит тестируемое представление

Когда вы нажмете «Далее», только что сгенерированный глобальный класс ABAP откроется в редакторе исходного кода Eclipse. Однако в глобальном тестовом классе вы найдете только пустое определение и реализацию класса. Чтобы просмотреть сгенерированный локальный класс модульного теста ABAP, содержащий тестовую оснастку, создание данных и методы тестирования, выберите вкладку Test Classes в редакторе исходного кода. См. рисунок 4.15. Он содержит наш только что определенный метод тестирования total\_price\_no\_items и метод prepare\_test\_data\_set, который будет вставлять наши ранее указанные тестовые данные документа покупки в тестовый дубликат документа покупки тестовой среды.

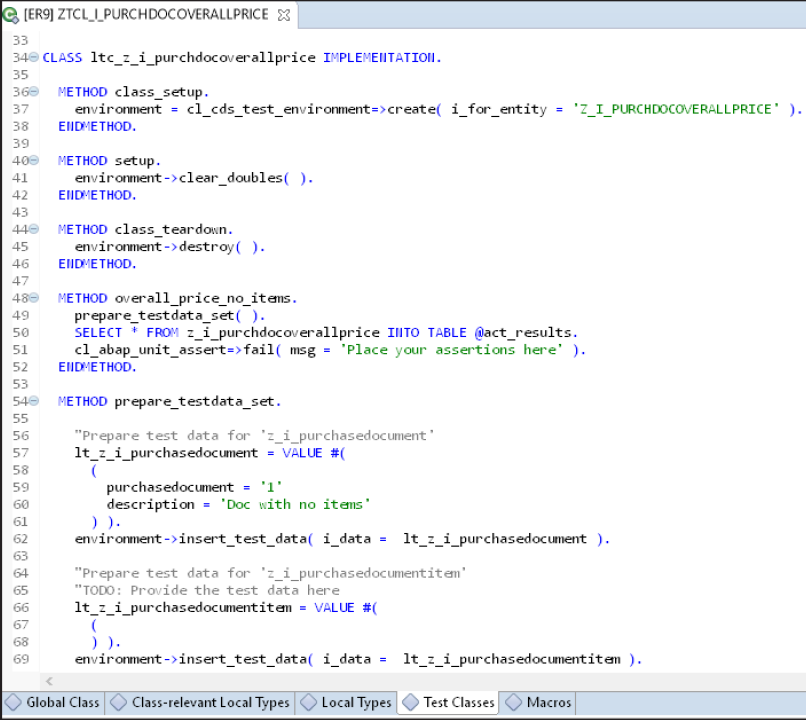


Рис. 4.15 Сгенерированный класс локального ABAP-модуля

Тестовые данные, определенные с помощью мастера создания тестовых данных, не будут вставлены в исходные таблицы базы данных зависимых от представлений. Вместо этого сгенерированный скелетный тестовый класс установит отдельную тестовую среду в своем методе class\_setup и не повлияет на продуктивные таблицы и их данные. Заглушки зависят от представлений (это копии исходных представлений, которые содержат предопределенные тестовые данные). Их данные будут доступны только во время выполнения метода тестирования и по умолчанию очищаются перед каждым выполнением теста в среде метода настройки тестовой оснастки environment->clear\_doubles().

Для нашего первого тестового случая мы не создавали никаких тестовых данных элемента, только один документ о покупке, который мы вставили в тестовую среду. Если в документе покупки нет товаров, общая цена документа покупки должна быть равна 0. Мы можем проверить это значение, адаптировав сгенерированный тестовый метод, который еще не проверяет правильность значения поля totalprice. Как правило, процедура тестирования представлений CDS всегда выполняется следующим образом:

1. Создайте тестовые данные для представлений, от которых зависит тестируемое представление (аранжировка или заданный шаг).

2. Выберите данные из тестируемого представления CDS (действие или шаг).

3. Убедитесь, что вычисленное значение имеет ожидаемое значение (утверждение или шаг).

Для проверки значений, рассчитанных представлением CDS, можно использовать стандартный класс CL\_ABAP\_UNIT\_ASSERT, который предоставляет множество статических методов для сравнения ожидаемых и фактических значений. В листинге 4.6 показан простой тест, гарантирующий, что общая цена документов покупки без товаров равна 0.

METHOD overall\_price\_no\_items.

prepare\_testdata\_set\_no\_items( ).

SELECT \* FROM z\_i\_purchdocoverallprice INTO TABLE @act\_results.

cl\_abap\_unit\_assert=>assert\_equals( exp = 0 act = act\_results[ 1 ]-overallprice ).

ENDMETHOD.

Листинг 4.6. Модульный тест ABAP для общей цены документа покупки без позиций

Этот листинг соответствует процедуре тестирования: сначала мы создадим тестовые данные аналогично методу создания сгенерированных тестовых данных, затем мы выберем данные из тестируемого представления CDS, используя Open SQL. Наконец, мы проверим, имеют ли выбранные данные ожидаемое расчетное значение. В настоящее время тест завершится с ошибкой синтаксиса, поскольку мы еще не добавили поле в представление. Чтобы исправить тест, просто добавьте поле в представление следующим образом:

0 as as OverallPrice,

*Примечание*

*Вы можете либо выполнить модульные тесты, щелкнув правой кнопкой мыши глобальный класс и выбрав «Запуск от имени — модульный тест ABAP», либо с помощью сочетания клавиш (Ctrl)+(Shift)+(F10).*

В нашем следующем тестовом примере мы создадим новый метод создания тестовых данных, который имеет два товара с общей ценой товара 10, как показано в листинге 4.7.

METHOD prepare\_testdata\_set.

"Prepare test data for 'z\_i\_purchasedocument'

lt\_z\_i\_purchasedocument = VALUE #(

( purchasedocument = '1'

description = 'Doc with no items' ) ).

environment->insert\_test\_data( i\_data = lt\_z\_i\_purchasedocument ).

"Prepare test data for 'z\_i\_purchasedocumentitem'

lt\_z\_i\_purchasedocumentitem = VALUE #(

( purchasedocument = '1' purchasedocumentitem = '1' overallitemprice = '10' )

( purchasedocument = '1' purchasedocumentitem = '2' overallitemprice = '10' ) ).

environment->insert\_test\_data( i\_data = lt\_z\_i\_purchasedocumentitem ).

ENDMETHOD.

Листинг 4.7. Тестирование двойного создания для документа покупки с двумя позициями документа покупки

Таким образом, наш второй тестовый метод будет выглядеть как код, показанный в листинге 4.8.

METHOD overall\_price.

prepare\_testdata\_set( ).

SELECT \* FROM z\_i\_purchdocoverallprice INTO TABLE @act\_results.

cl\_abap\_unit\_assert=>assert\_equals( exp = 20 act = act\_results[ 1 ]-overallprice ).

ENDMETHOD.

Листинг 4.8. Модульный тест ABAP для общей цены документа покупки с двумя позициями документа покупки

Чтобы исправить этот тест, теперь мы должны фактически агрегировать общие цены товаров документа покупки. Сначала мы добавим в представление поле общей цены товара документа покупки, используя следующее выражение пути:

\_PurchaseDocumentItem.OverallItemPrice

Однако это выражение увеличит кардинальность данных, возвращаемых представлением, с 1 документа покупки до 2 документов покупки с разными общими ценами товаров. Чтобы уменьшить кардинальность, необходимо сгруппировать общие цены товаров по их общим полям документов покупки и агрегировать общие цены товаров с помощью функции агрегирования СУММ, предполагая, что все позиции документов покупки используют одну и ту же валюту. В листинге 4.9 показан фрагмент представления CDS, вычисляющего общую цену документа покупки.

@AbapCatalog.sqlViewName: 'ZIPURCHDOCPRICE'

@EndUserText.label: 'Purchase Document Item'

@VDM.viewType: #COMPOSITE

define view Z\_I\_PurchDocOverallPrice

as select from Z\_I\_PurchaseDocument

{

key PurchaseDocument,

@Semantics.amount.currencyCode: 'Currency'

@DefaultAggregation: #NONE

@ObjectModel.foreignKey.association: '\_Currency'

sum( \_PurchaseDocumentItem.OverallItemPrice )as OverallPrice,

@Semantics.currencyCode: true

\_PurchaseDocumentItem.Currency,

…

// Associations

…

\_Currency

}

group by

PurchaseDocument,

\_PurchaseDocumentItem.Currency

…

Листинг 4.9. Представление составного интерфейса, вычисляющее общую цену документа покупки

*Примечание*

*Значения полей вычисляются математическим оператором (+, -, / и т.д.); агрегатная функция (max, min, sum и т. д.); или выражение CASE нельзя использовать в том же представлении для дополнительных вычислений. В таких случаях вы всегда должны создавать дополнительное представление сверху, чтобы использовать вычисляемое значение или поле.*

#### Вычисление Overall Price Criticality путем Composite Interface View и использования разработки через тестирование

Еще одно вычисляемое поле, которое вам позже понадобится в пользовательском интерфейсе нашего приложения, — это индикатор общей критичности цены. Мы будем использовать рассчитанное поле критичности, чтобы выделить общие цены, которые мы считаем критическими для пользователя, проверяющего документы о покупке. Реализуемая логика показана в таблице 4.1; например, если общая цена документа ниже 1000, документ не должен считаться критичным и будет обозначен цифрой 3. С другой стороны, если цена выше 10000, мы должны считать этот документ очень критическую и обозначим эту критичность цифрой 1.

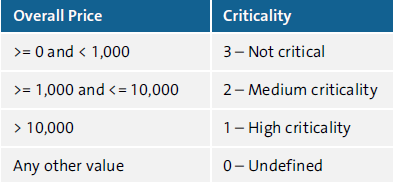


Таблица 4.1 Соотношение общей цены и показателя критичности

Чтобы реализовать это поле, мы поместим конечное представление потребления поверх предыдущего составного представления, которое назовем Z\_C\_PurchaseDocumentLrp, где C\_ означает представление потребления. Мы снова выполним реализацию на основе тестирования и создадим тестовый пример для каждого пограничного случая, прежде чем реализовывать фактическую логику расчета в представлении CDS. В листинге 4.10 показана реализация модульного теста для общей цены 1000. В этом случае мы ожидаем выхода 2 для поля, что указывает на среднюю критичность.

METHOD price\_1k\_medium\_critical.

prepare\_testdata\_price( 1000 ).

SELECT \* FROM z\_c\_purchasedocumentlrp INTO TABLE @act\_results.

cl\_abap\_unit\_assert=>assert\_equals( exp = 2 act = act\_results[ 1 ]-overallpricecriticality ).

ENDMETHOD.

Листинг 4.10. Модульный тест ABAP для сопоставления общей цены (1000) и критичности (2)

Остальные тесты будут выглядеть так же, за исключением параметра цены и ожидаемого значения. Результат реализации поля общей ценовой критичности в CDS DDL выглядит так, как показано в листинге 4.11.

case when OverallPrice >= 0 and OverallPrice < 1000 then 3

when OverallPrice >= 1000 and OverallPrice <= 10000 then 2

when OverallPrice > 10000 then 1

else 0 end as OverallPriceCriticality,

Листинг 4.11. Логика расчета для поля TotalPriceCriticality в представлении потребления Z\_C\_PurchaseDocumentLrp CDS

#### Вычисление Approval Required с использованием разработки через тестирование

Еще один простой индикатор, который нам позже понадобится в пользовательском интерфейсе, указывает, требует ли документ о покупке утверждения менеджером по закупкам или нет. Мы укажем, что документ о покупке требует утверждения, когда общая цена превышает 1000. Для этого простого индикатора вам понадобятся всего два теста: один, где общая цена составляет 1000 и не требует одобрения, и другой, где общая цена составляет 1001 и требует одобрения. Результат реализации должен выглядеть так, как показано в листинге 4.12.

case when OverallPrice > 1000 then 'X' else '' end as IsApprovalRequired,

Листинг 4.12. Логика расчета для поля IsApprovalRequired в представлении потребления Z\_C\_PurchaseDocumentLrp CDS

На рис. 4.16 показаны все успешно выполненные модульные тесты тестируемого представления CDS для поля общей ценовой критичности, а также для поля обязательного утверждения.

На данный момент, после добавления вычисляемых полей, VDM нашего приложения для закупок должен выглядеть так, как показано на рисунке 4.17. Уровень базы данных содержит прозрачные таблицы для основных сущностей приложения: закупочных документов и товаров.

Кроме того, мы создали отдельные таблицы для приоритетов документов, статуса и закупочных организаций, чтобы хранить их ключи и тексты в нормализованном виде и без избыточности данных. В дополнение к таблицам базы данных мы представили уровень базового представления с базовым представлением интерфейса для каждой таблицы базы данных.

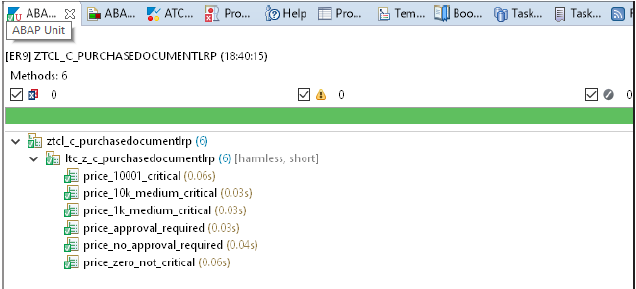


Рисунок 4.16. Успешно выполненные модульные тесты для полей «Общая ценовая критичность» и «Требуемое утверждение»

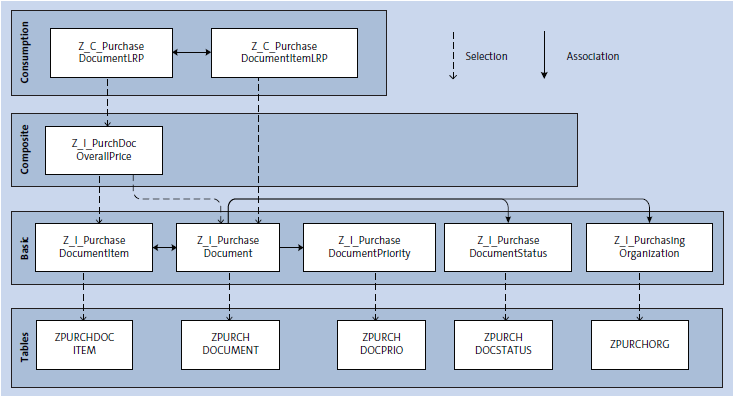


Рис. 4.17 VDM приложения для закупок, состоящее из четырех разных уровней

Приоритеты, статусы и закупочные организации связаны с документом закупки через ассоциации внешнего ключа. Документы и элементы также связаны с другим соответствующим объектом. В дополнение к представлениям документа о покупке и элементам мы помещаем составное представление интерфейса для расчета общей цены документов путем агрегирования общих цен элементов документа. Наконец, самый верхний слой состоит из двух представлений потребления для документов и товаров покупки. В представлении потребления для документов покупки мы добавили еще два вычисляемых поля для общей ценовой критичности и того, требуется ли утверждение документа или нет.

### 4.1.5 Добавление DCL файлов

Доступ к данным, предоставляемым представлениями CDS, может быть ограничен с помощью классических объектов авторизации ABAP и DCL CDS. Чтобы настроить это ограничение, сначала необходимо создать поле полномочий с помощью транзакции SU20, а также класс полномочий и объект с помощью транзакции SU21. Для нашего сценария приложения мы создадим объект авторизации для закупочной организации, который также является полем, отображаемым в представлении CDS документа покупки. В нашем случае мы хотим показать конечным пользователям только закупочные документы закупочных организаций, которым они назначены, поскольку по умолчанию они не должны иметь доступ ко всем закупочным документам в системе. Объект полномочий для закупочной организации показан на рисунке 4.18. Объект полномочий содержит два поля полномочий: одно для фактической закупочной организации (ZPURCHORGA) и одно для разрешенной деятельности (ACTVT), например, «01» для создания, «02» для изменения и «03» для просмотра.

Чтобы ***применить ограничения доступа на основе объектов авторизации***, вы должны создать контроль доступа с помощью DCL CDS и связать его с представлением CDS. Вы можете создать контроль доступа CDS в Eclipse, выполнив следующие шаги:

1. Откройте перспективу ABAP в инструментах разработки ABAP в Eclipse.

2. В проекте ABAP выберите узел пакета, в котором вы хотите создать таблицу, в представлении Project Explorer.

3. Откройте контекстное меню, щелкнув пакет правой кнопкой мыши и выбрав ***«Создать» — «Другой ABAP-репозиторий» — «Основные службы данных» — «Управление доступом».***

4. Когда откроется мастер создания, введите Имя и Описание. По соглашению имя источника DCL будет идентичным имени источника DCL. В нашем случае мы назовем файл DCL Z\_I\_PurchaseDocument.

5. Вы можете выбрать один из предоставленных шаблонов для создания каркаса управления доступом к DCL, например, «Определить роль экземпляра», который создаст простой скелетный файл DCL.

6. Нажмите кнопку Готово в мастере, и откроется редактор исходного кода, в котором теперь вы можете определить правила и условия доступа.

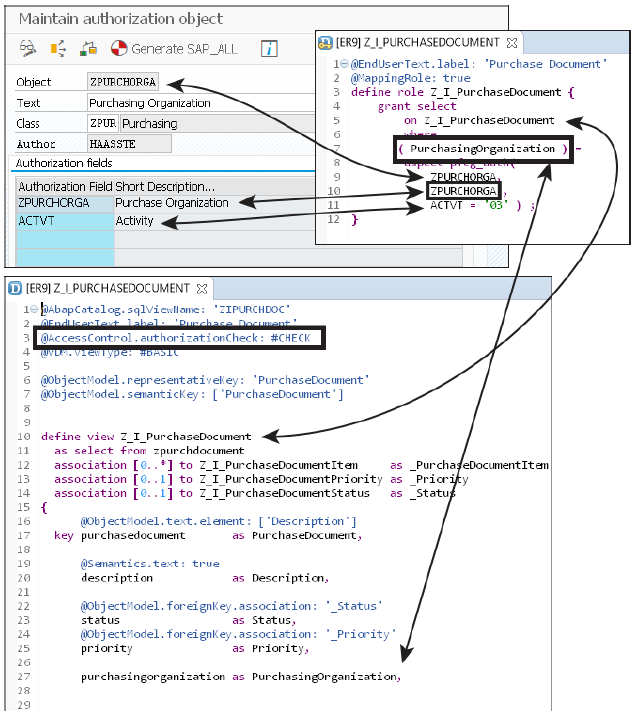


Рис. 4.18. Обзор связей между объектом авторизации, представлением CDS и контролем доступа

*Примечание*

*Чтобы включить контроль доступа со стороны представления CDS, вы также должны добавить аннотацию* ***@AccessControl.authorizationCheck: #CHECK*** *к представлению CDS. Эта аннотация приведет к неявному управлению доступом, когда данные выбираются из представления CDS с помощью Open SQL.*

Полное определение контроля доступа для базового представления интерфейса нашего документа о покупке показано в листинге 4.13. По соглашению имя роли внутри DCL будет идентично имени представления CDS в верблюжьем регистре. Определение роли содержит условие транзакции PFCG, которое будет оцениваться каждый раз при прямом доступе к представлению с помощью Open SQL. Левая часть условия состоит из поля PurchasingOrganization представления CDS, где правая часть состоит из имени объекта авторизации (ZPURCHORGA) в качестве первого параметра, за которым следуют его поля авторизации (ZPURCHORGA, ACTVT), подлежащие оценке.

@EndUserText.label: 'Role for Z\_I\_PurchaseDocument'

@MappingRole: true

define role Z\_I\_PurchaseDocument {

grant select

on Z\_I\_PurchaseDocument

where

( PurchasingOrganization ) =

aspect pfcg\_auth(

ZPURCHORGA, // Authorization object

ZPURCHORGA , // Matched auth. object field

ACTVT = '03' // Fixed auth. object field

) ;

}

Листинг 4.13. Определение управления доступом для представления Z\_I\_PurchaseDocument CDS с использованием языка управления данными

Во время выполнения при выполнении выбора Open SQL в представлении назначенные пользователю роли транзакции PFCG и авторизации будут извлечены и сравнены с данными, предоставленными представлением CDS. В нашем примере пользователь может быть назначен закупочной организации Org1, что приведет к следующему прозрачному расширению инструкции CDS SELECT:

SELECT \* FROM Z\_I\_PurchaseDocument WHERE PurchaseOrganization = ‘Org1’;

Кроме того, наше условие Transaction PFCG также содержит жестко закодированную проверку (ACTVT=’03’), которая гарантирует, что у пользователя есть права на отображение. Если назначенное ему значение ACTVT не равно «03», пользователь вообще не увидит никаких данных.

На рис. 4.18 показаны связи между представлением CDS, ролью DCL и объектом авторизации. Чтобы избежать повторения одних и тех же правил и условий доступа для схожих представлений, элементы управления доступом могут быть унаследованы. Вы также можете комбинировать унаследованные и новые условия доступа. Если вы просто хотите повторно использовать файл управления доступом, созданный для базового представления интерфейса документа о покупке, в нашем составном представлении интерфейса, представленном для расчета общей цены, вы можете создать управление доступом, показанное в листинге 4.14, без необходимости повторять условие транзакции PFCG.

@EndUserText.label: 'Purchase Document Overall Price'

@MappingRole: true

define role Z\_I\_Purchdocoverallprice {

grant select

on Z\_I\_PurchDocOverallPrice

inherit Z\_I\_PurchaseDocument;

}

Листинг 4.14. Пример унаследованного контроля доступа

Чтобы назначить определенные полномочия пользователям, необходимо создать транзакционную роль PFCG и включить объекты полномочий и их значения либо вручную, либо через приложение, которому назначены значения полномочий по умолчанию, предоставленные через транзакцию SU22.

Наконец, чтобы авторизация вступила в силу, пользователю должна быть назначена роль транзакции PFCG.

*Примечание*

*Элементы управления доступом DCL учитываются только при прямом доступе к представлению CDS. Они игнорируются при косвенном запросе через другие представления CDS. Контроль доступа можно протестировать с помощью транзакции SACMSEL, что позволяет оценить контроль доступа CDS для разных пользователей.*

## 4.2 Разработка Unmanaged Transaction Business Object с использованием модели программирования ABAP RESTful

На данный момент наше приложение состоит из VDM на основе представления CDS, которое обеспечивает доступ к документам и предметам покупки с использованием Open SQL, языка запросов CDS. Однако мы пока не можем создавать какие-либо документы или элементы. На данный момент приложение полностью доступно только для чтения. Если требуется какая-либо транзакционная логика, например, использование операций создания, чтения, обновления и удаления (CRUD) над сущностями, в игру вступает новая модель программирования ABAP RESTful, поскольку, начиная с SAP S/4HANA 1909, вы можете создавать Бизнес-объект на основе модели программирования ABAP RESTful и структура его узлов из модели данных CDS. Следовательно, в этом разделе вы узнаете, как добавить транзакционные возможности в VDM на основе CDS только для чтения с помощью определений поведения.

### 4.2.1 Создание Unmanaged Transaction Business Object из модели CDS

Поскольку мы хотим, чтобы наши базовые представления интерфейса можно было повторно использовать во всех сценариях приложений, мы не будем добавлять семантические ассоциации бизнес-объектов, необходимые для моделирования транзакционного бизнес-объекта, к существующим базовым представлениям интерфейса; вместо этого мы создадим новые составные представления интерфейса поверх основных представлений интерфейса.

Затем мы создадим неуправляемый бизнес-объект для документов покупки, а также для позиций документов покупки, который мы назовем Z\_I\_PurchaseDocument\_U и Z\_I\_PurchaseDocumentItem\_U, где *\_U* означает неуправляемый бизнес-объект.

Как описано в главе 4, для SAP S/4HANA 1909 была выпущена только неуправляемая версия модели программирования ABAP RESTful. Управляемая версия в настоящее время доступна в облаке, а локальная версия вскоре появится.

Соглашение, которое мы использовали для создания Z\_I\_PurchaseDocumentItem\_U для представления CDS с неуправляемой реализацией, не является обязательной практикой именования; идентифицировать эти типы представлений с помощью *\_U* просто проще и понятнее. В обоих представлениях вы добавите все поля базового представления в список полей выбора, чтобы создать простую проекцию базового представления.

Точно так же в этой главе мы будем следовать еще нескольким соглашениям об именах. Как правило, объекты разработки именуются как

***[/<пространство имен>][<префикс>]\_<имя\_объекта>\_[<суффикс>]***.

Следуя этому эмпирическому правилу, давайте рассмотрим практику именования различных элементов, которые мы будем создавать в этой главе:

***- Сервисные привязки (Service bindings)***

Для привязок службы префикс должен быть *UI\_* или *API\_*, в зависимости от типа службы OData, которую вы хотите создать. Суффикс будет *\_V2* или *\_V4* (в зависимости от выбранной версии протокола OData). В этом примере дополнительный *Z\_* перед именами соответствует наиболее распространенному пространству имен клиентов (которое также может быть эксклюзивным пространством имен для конкретных клиентов, например /customername/).

Пример:

Z\_UI\_PurchaseDocument\_U\_V2

***- Пулы поведения (Behavior pools) или классы реализации поведения (behavior implementation)***

Префикс будет *BP\_* (сокращение от пула поведения).

Пример:

ZBP\_I\_PurchaeDocument\_U

***- Классы обработчика поведения (behavior handler) и сохранения поведения (behavior saver)***

Эти классы являются локальными классами в пуле поведения:

– *LHC\_* для класса обработчика поведения

– *LSC\_* для класса сохранения поведения

Пример:

lhc\_Z\_I\_PurchaseDocument\_U

lsc\_Z\_I\_PurchaseDocument\_U

Чтобы добавить определение поведения в представление, вы должны сначала определить новое представление как корневое представление. Назовем это корневое представление, которое будет иметь транзакционные возможности, Z\_I\_PurchaseDocument\_U. Это корневое представление будет служить нашим заголовком документа о покупке и будет составным представлением.

Основные аннотации для этого представления показаны в листинге 4.15 на уровне заголовка.

@AbapCatalog.sqlViewName: 'ZIPURCHDOCU'

@AccessControl.authorizationCheck: #NOT\_REQUIRED

@EndUserText.label: 'PurchaseDocument'

@AbapCatalog.preserveKey: true

@VDM.viewType: #COMPOSITE

define root view Z\_I\_PurchaseDocument\_U

Листинг 4.15. Аннотации заголовка корневого узла бизнес-объекта

Бизнес-объект представляется в виде иерархического дерева узлов, и благодаря добавлению синтаксиса определения корневого представления представление характеризуется как корневое представление иерархии узлов. Параметр ***@AbapCatalog.preserveKey: true*** должен быть добавлен, когда представление установлено ***как корневое*** представление. Кроме того, чтобы построить иерархию узлов бизнес-объектов, вы также должны аннотировать ассоциацию с представлением элементов документа покупки как составное, как показано в листинге 4.16.

define root view Z\_I\_PurchaseDocument\_U

as select from Z\_I\_PurchDocOverallPrice

composition [0..\*] of Z\_I\_PurchaseDocumentItem\_U as \_PurchaseDocumentItem,

Листинг 4.16. Ассоциация дочернего узла корневого узла бизнес-объекта BOPF

Кроме того, вы должны добавить несколько аннотаций на уровне заголовка в представлении позиции документа покупки (Z\_I\_PurchaseDocumentItemTP), как показано в листинге 4.17.

@AbapCatalog.sqlViewName: 'ZIPURCHDOCITEMU'

@AccessControl.authorizationCheck: #NOT\_REQUIRED

@EndUserText.label: 'Purchase Document Item'

@AbapCatalog.preserveKey: true

@VDM.viewType: #COMPOSITE

…

Листинг 4.17. Аннотации заголовка дочернего узла бизнес-объекта

Как и ранее для документов покупки, вам также необходимо включить все транзакционные операции для этого узла бизнес-объекта. Связь с документом покупки должна быть аннотирована как родительская. Более того, представления, не представляющие корневой узел иерархии, также должны указывать свою ассоциацию с корневым узлом, которая в нашем случае эквивалентна родительской ассоциации, как показано в листинге 4.18.

define view Z\_I\_PurchaseDocumentItem\_U

as select from Z\_I\_PurchaseDocumentItem

association to parent Z\_I\_PurchaseDocument\_U as \_PurchaseDocument on

$projection.PurchaseDocument = \_PurchaseDocument.PurchaseDocument

Листинг 4.18. Ассоциация родительского дочернего узла бизнес-объекта

### 4.2.2 Определение службы (Service Definition)

После активации представлений транзакций или бизнес-объектов вы можете напрямую перейти к созданию определения поведения, которое добавляет в представления функциональные возможности CRUD. Но сначала давайте зарегистрируем наши представления, используя определение службы и привязку службы, и посмотрим, как должна выглядеть служба OData. Чтобы создать новое определение службы, щелкните правой кнопкой мыши имя пакета в папке проекта и выберите «**Новый другой объект репозитория ABAP**». Затем разверните папку Business Services и выберите Service Definition, как показано на рис. 4.19.

Теперь введите имя и описание для определения службы. Определения службы не следуют каким-либо специальным соглашениям об именах, поскольку они являются частью бизнес-службы и не имеют типов или спецификаций, в отличие от привязки службы. Введите данные для определения сервиса, как показано на рис. 4.20, и нажмите «Далее».

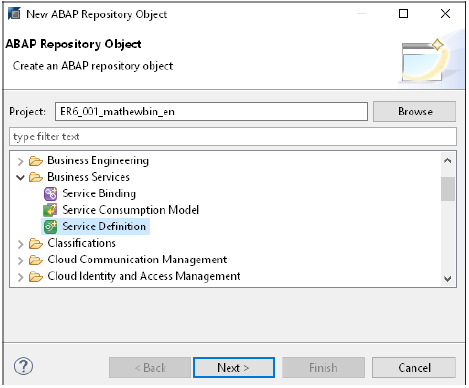


Рисунок 4.19 Создание нового определения службы для неуправляемых представлений

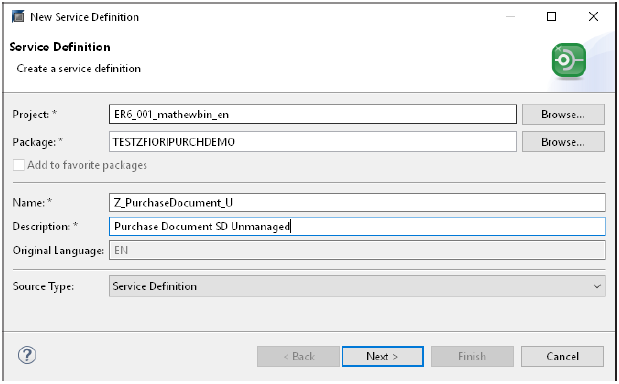


Рисунок 4.20 Детали определения службы

Как только определение службы будет создано, вы увидите пустое определение службы, в котором вы будете отображать все представления CDS как сущности. Если вы знакомы с транзакцией SEGW для создания проектов OData (или если вы читали первое издание этой книги), вы заметите, что тот же набор сущностей отображается в разделе «Модель данных» «Ссылки на источники данных» через SADL. CDS-Entity раскрывает типы объектов внутри любого из проектов в SEGW, которые вы создаете на основе представления CDS. Хотя это почти то же самое, в этом случае мы не будем использовать SAP GUI для доступа к транзакции SEGW, а вместо этого вручную предоставим представления CDS и их ассоциации, необходимые для нашего проекта. При таком подходе вам никогда не придется открывать SAP GUI для какой-либо части разработки. Для нашего сценария, чтобы убедиться, что все значения и ассоциации внешних ключей правильно подобраны, мы предоставим все основные представления \_U, базовые представления I\_ и соответствующие представления справки по значениям, как показано на рис. 4.21.

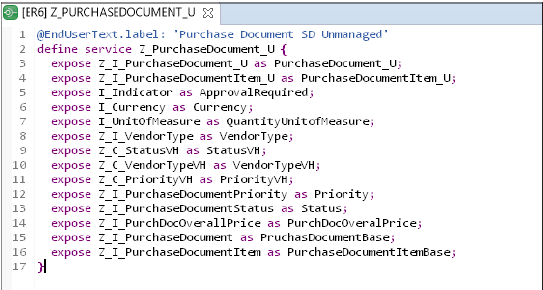


Рис. 4.21. Предоставление обязательных полей представления CDS через определение службы

### 4.2.3 Привязка службы (Service Binding)

Теперь, когда мы определили, какие представления CDS должны предоставляться как служба OData, давайте зарегистрируем их как службу OData, создав привязку службы. Этот процесс аналогичен тому, когда вы впервые активировали проект построителя службы SAP Gateway (транзакция SEGW), а ваша служба OData активируется в фоновом режиме. Чтобы создать новую привязку службы, щелкните правой кнопкой мыши имя пакета в папке проекта и выберите «**Создать — Другой объект репозитория ABAP**». Затем разверните папку Business Services и выберите Service Binding, как показано на рис. 4.22.

На следующем экране дайте привязке службы имя (в соответствии с соглашением об именах префикс должен быть *UI\_*, поскольку мы регистрируем OData для использования в приложениях SAPUI5, а суффикс должен быть *\_V2*, поскольку мы используем протокол OData V2).

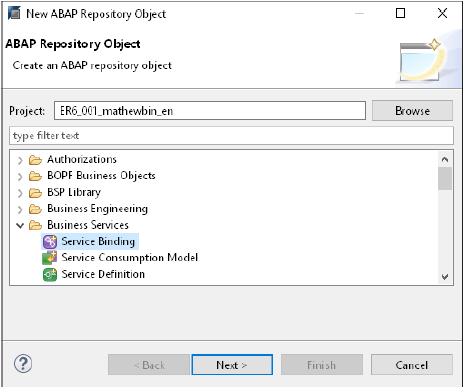


Рис. 4.22 Привязка службы для регистрации службы OData

Кроме того, вы должны указать имя определения службы, как показано на рис. 4.23, чтобы привязка службы могла ссылаться на определение службы и соответствующим образом отображать представления CDS в результирующей службе OData.

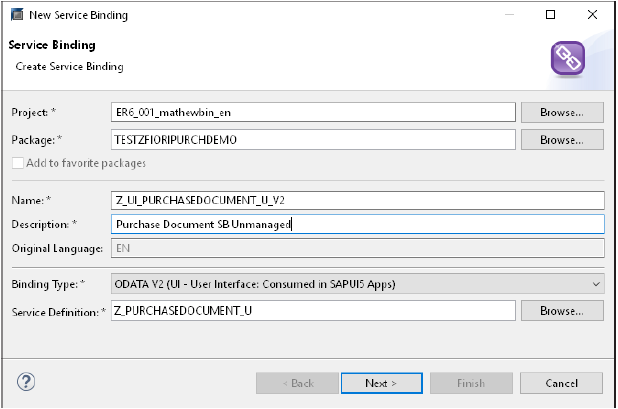


Рисунок 4.23 Детали привязки службы

Как показано на рис. 4.24, при создании определений службы доступно несколько параметров. В дополнение к параметрам OData V2 и V4 вы также можете увидеть подкатегорию для служб пользовательского интерфейса и веб-API. (Веб-API — это обычные представления CDS, предоставляемые как службы OData без аннотаций пользовательского интерфейса, которые обычно используются для приложений SAP Fiori на основе шаблонов, таких как элементы SAP Fiori.)

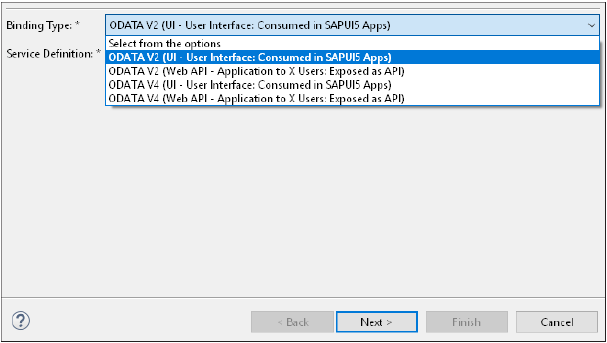


Рисунок 4.24 Типы привязки службы

После завершения работы мастера привязки службы вы увидите имя и версию для определения службы. Теперь вы должны активировать привязку службы. Нажмите кнопку «Активировать», показанную на рис. 4.25, которая зарегистрирует вашу службу OData в системе.

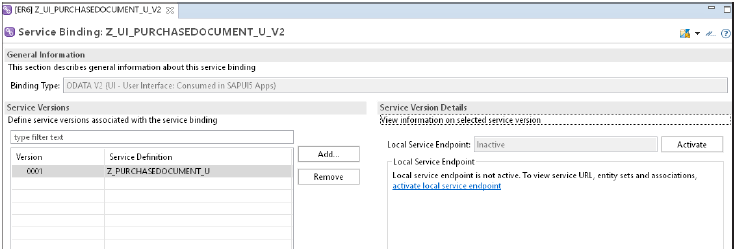


Рисунок 4.25 Активация привязки службы

Теперь вы увидите службу OData в поле URL-адрес службы. Вы можете щелкнуть ссылку URL-адрес службы, чтобы просмотреть вызов службы OData. Но модель программирования ABAP RESTful предоставляет еще более интересный и мощный вариант: вы можете просмотреть свой сервис в реальном приложении элементов SAP Fiori, которое автоматически создается на основе аннотаций, представленных в наших представлениях CDS. Щелкните правой кнопкой мыши верхний набор сущностей представления CDS PurchaseDocument\_U, который мы предоставили из представления Z\_I\_PurchaseDocument\_U CDS, и щелкните ***Open Fiori Elements App Preview***, как показано на рис. 4.26.

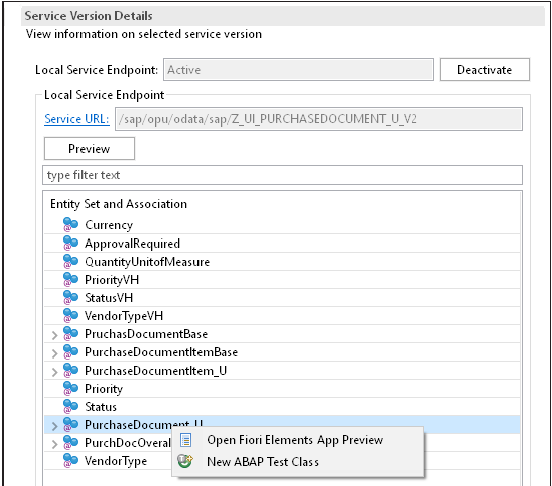


Рис. 4.26 Открытие функции предварительного просмотра приложения SAP Fiori Elements в определении службы

Теперь ваше приложение элементов SAP Fiori автоматически откроется в браузере. По умолчанию в таблице приложения не будут выбраны столбцы. (Поскольку мы не добавили аннотации пользовательского интерфейса для указания столбцов и соответствующих свойств пользовательского интерфейса, мы добавим их позже в этой главе, в разделе 4.4.4). Чтобы увидеть результат просмотра CDS, щелкните значок настроек в правом верхнем углу таблицы, выберите необходимые столбцы и нажмите кнопку «Перейти», как показано на рис. 4.27. Обратите внимание, что в этом предварительном просмотре у вас по-прежнему не включены никакие транзакционные функции. Мы добавим их в наши представления \_U в следующем разделе.

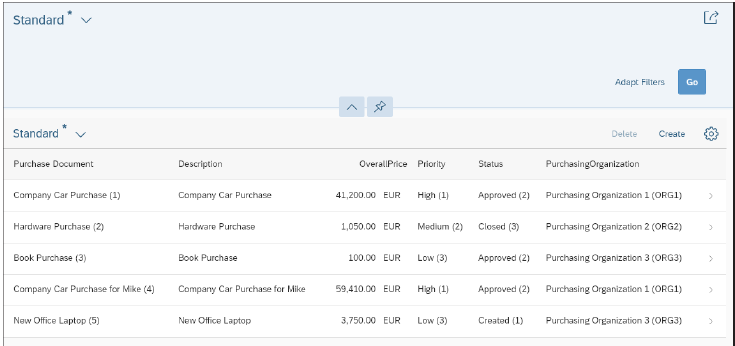


Рис. 4.27 Предварительный просмотр приложения SAP Fiori Elements

### 4.2.4 Язык управления сущностями (EML)

Прежде чем вы начнете добавлять транзакционные операции с помощью определений поведения, давайте сначала познакомимся с языком манипулирования сущностями (EML), который формирует основу определения поведения. EML управляет поведением бизнес-объекта, а в модели программирования ABAP RESTful EML является частью языка ABAP. EML обеспечивает доступ CRUD к данным, найденным в транзакционных объектах.

EML имеет следующие три основных **свойства**: *изменять сущности, читать сущности и фиксировать сущности*.

#### Изменить сущности (Modify Entities)

Это свойство охватывает операции, позволяющие изменять данные в сущностях, и поддерживает следующие операции:

- Создать: создание новых записей в объекте.

- Создать по ассоциации: создает новые записи в дочернем объекте, передавая ключ родительского объекта.

- Обновление: обновление/изменение данных в объекте.

- Удалить: удаление данных из сущности.

- Действия: пользовательская логика, которую в обычных сценариях невозможно реализовать с помощью операций создания, чтения, обновления и удаления (CRUD). (В нашем примере мы используем эту настраиваемую логику для утверждения или отклонения документов о покупке.)

Все операции CRUD имеют входные параметры в виде таблицы экземпляров. Все методы операции CRUD имеют входные параметры табличного типа. Также для действий можно добавить параметр RESULT на тот случай, если действие должно возвращать какой-то результат. MODIFY также имеет параметры ответа с именами FAILED, MAPPED и REPORTED, эти параметры результата можно использовать для передачи ответов для каждой операции. Вы узнаете о параметрах результата более подробно в Разделе 4.2.6.

#### Чтение сущностей (Read Entities)

Это свойство разрешает все операции, не связанные с изменением/модификацией данных в сущностях, и поддерживает следующие операции:

- Чтение: извлекает данные из объекта с помощью ключа.

- Чтение по ассоциации: извлекает данные из дочерних сущностей на основе родительской дочерней ассоциации путем передачи ключа.

Операция чтения имеет свойство RESULT, которое необходимо использовать для указания возвращаемого результата. Операция Read by Association имеет свойство LINK, которое содержит только список ключей, ключ исходного объекта и ключ целевого объекта. Это свойство LINK используется для обеспечения того, чтобы процесс чтения по ассоциации был допустимым на основе ключа.

#### Фиксировать сущности (Commit Entities)

Все операции, упомянутые в разделе MODIFY (которые выполняются в классе реализации поведения или пуле поведения), не сохраняют данные в базу данных. Вместо этого эти данные хранятся в промежуточном буфере (внутренних таблицах), и содержимое этих буферов действительно до конца этого сеанса ABAP. Таким образом, данные должны быть сохранены ближе к концу сеанса.

Эти последовательности сохранения запускаются с помощью оператора COMMIT ENTITIES. Во время выполнения эти последовательности сохранения делятся на следующие методы, выполняемые в указанном порядке:

- finalize: Все окончательные расчеты выполняются этим методом.

- check\_before\_save: здесь выполняются все окончательные проверки и проверки.

- save: Окончательная фиксация в базе данных. Если используется функция поздней нумерации (для замены временного ключа фактическим перед сохранением данных), то на этом этапе также инициируется вызов метода Adjust\_numbers.

### 4.2.5 Behavior Definitions для добавления транзакционных функций (Transactional Features)

Как упоминалось в предыдущей главе, определение поведения используется для описания поведения бизнес-объекта с помощью языка определения поведения (BDL). Определение поведения добавляет такие возможности, как операции CRUD и аспекты моделирования, такие как концепция блокировки между родительским и дочерним представлениями. В дополнение к стандартным операциям CRUD определение поведения также позволяет добавлять настраиваемые действия в бизнес-объект. *Бизнес-объект может иметь только одно определение поведения, и бизнес-объект должен быть определен как корневое представление.*

Теперь давайте создадим представление определения поведения, чтобы включить транзакционные действия в приведенных выше представлениях. Чтобы создать определение поведения, щелкните правой кнопкой мыши свой пакет и выберите «**Создать — Другой объект репозитория ABAP**». Затем разверните папку *Core Data Services* и выберите *Behavior Definition*, как показано на рис. 4.28.

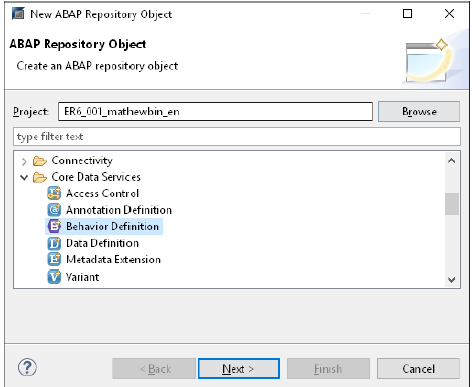


Рисунок 4.28 Создание нового определения поведения

На следующем экране укажите сведения об определении поведения. На этом шаге вы не можете указать собственное имя для определения поведения. Вместо этого вы можете указать только имя корневой сущности представления CDS, для которого создается определение поведения. Мы введем имя нашего корневого представления Z\_I\_PurchaseDocument\_U, как показано на рис. 4.29, а затем нажмем «Далее» и «Готово». На рис. 4.29 вы могли заметить поле Implementation Type. Вы не увидите это поле, если не используете облачную систему. В настоящее время в локальных системах поддерживаются только неуправляемые реализации.

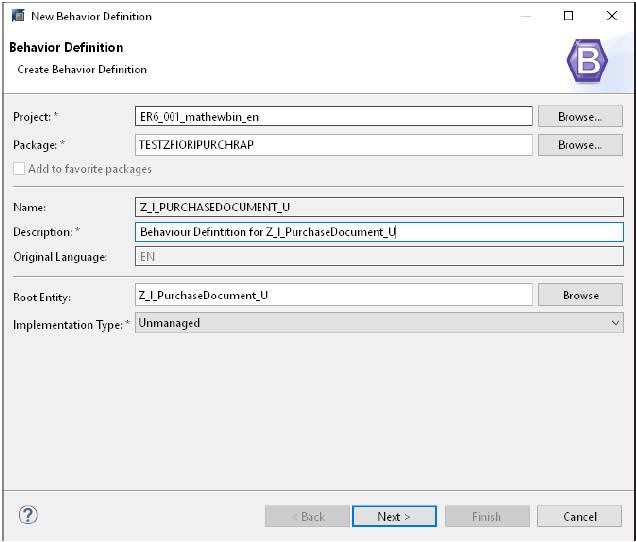


Рисунок 4.29 Детали определений поведения

Корневое представление CDS, указанное вами при создании определения поведения, как показано на рис. 4.29. Вы можете видеть, что имя корневого представления CDS введено в поле Root Entity, а само определение поведения имеет отношение 1:1. Другими словами, определение поведения может иметь только одно корневое представление CDS, и наоборот. Это корневое представление CDS должно обрабатывать все необходимые ассоциации с другими представлениями.

Как показано на рисунке 4.30. мастер определения поведения автоматически создал код для реализации действий CRUD. Мастер также определил связь родитель-потомок между Z\_I\_PurchaseDocument\_U и Z\_I\_PurchaseDocumentItem\_U и создал функцию «Создать по ассоциации», которая также реализована автоматически. Эта связь важна, если вы хотите создать глубокие сущности, такие как элемент заголовка документа покупки. На следующем шаге, когда вы создаете реализацию определения поведения, платформа автоматически создаст методы класса для всех действий и ассоциаций (Создать по ассоциации, Чтение по ассоциации и т. д.). В строке заголовка определения поведения вы увидите, что имя класса реализации было автоматически предложено платформой, как показано на рис. 4.30. Мы создадим этот класс как часть реализации поведения в следующем разделе. Кроме того, в области заголовка вы можете увидеть уникальное свойство, определенное после имени класса. Это объявление означает, что каждая операция выполняется ровно один раз. Другими словами, один и тот же глобальный класс поведения, определенный на этом шаге, применяется ко всем объектам, определенным в определении, в данном случае к Z\_I\_Purchasedocument\_u и Z\_PurchaseDocumentItem. Если вам нужны другие классы реализации глобального поведения, вы должны удалить имя класса и объявление уникальности из строки заголовка и определить имена отдельных классов рядом с сущностями, например, реализация в классе *<имя класса> [unique]*, синтаксис показан в листинге 4.19.

***define behavior for*** *Z\_I\_PurchaseDocument\_U* ***alias*** *PurchaseDocument*

***implementation in class*** *zbp\_i\_purchasedocument\_u* ***unique***

***define behavior for*** *Z\_I\_PurchaseDocumentItem\_U* ***alias*** *PurchaseDocumentItem*

***implementation in class*** *zbp\_i\_purchasedocumentItem\_U* ***unique***

Листинг 4.19. Синтаксис класса Definition Implementation

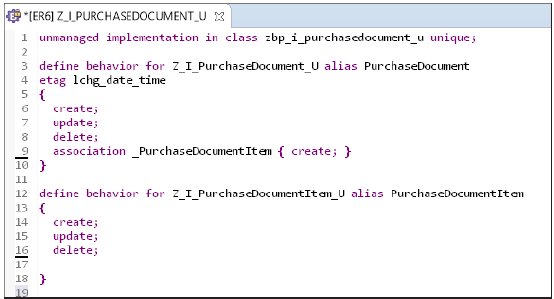


Рисунок 4.30 Behavior Definition Implementation

Определение поведения похоже на реализацию BOPF. Если вы привыкли к концепциям BOPF, обратите внимание, что определение поведения воспроизводит шаги BOPF в аналогичной последовательности. В BOPF также есть часть для определения действий, а затем часть для реализации логики этих действий через классы. *Определения поведения (Behavior Definition) заменяют концепцию BOPF другим подходом*.

Глядя на определения уровня объекта, показанные на рис. 4.30, обратите внимание также на определение **etag** (которое мы подробно описали в главе 3, разделе 3.4.3). **ETag** используется для обработки параллелизма в соответствии с той же концепцией в SAP Gateway, где платформа запускает запрос READ от серверной части на стороне клиента перед выполнением действия обновления/изменения. Вы также можете воспользоваться преимуществами других определений на уровне объекта, основанных на требованиях приложения, таких как блокировка (главная блокировка и зависимая от блокировки) или поздняя нумерация (назначает временный ключ полю, который, исходя из бизнес-логики, может получить только фактический непосредственно перед фиксацией/сохранением данных в бэкенде). Подробное объяснение этих концепций представлено в главе 3.

Другой частью определения поведения являются его поддерживаемые операции. Операции, поддерживаемые определениями поведения:

- Создавать

- Обновлять

- Удалять

- Создать по ассоциации (пример: *association \_PurchaseDocumentItem {create}*, как показано на рис. 4.30)

Ключевое слово *{create;}* используется для объявления того, что эта ассоциация разрешена для создания, что может использоваться для создания глубоких сущностей, таких как документ о покупке и создание элемента.

Еще одна функция, поддерживаемая определениями поведения, — это действия, то есть операции, не являющиеся частью стандартного предложения определения поведения, например операции CRUD.

Действия могут быть определены следующим образом:

*[static] action ActionName [external ‘ExternalActionName’] [parameter*

*InputParameterEntity] [result[cardinality] ResultParameterEntity]*

Это заявление состоит из следующих элементов:

- static: необязательны (эти действия не относятся к какой-либо сущности).

- external: ключевое слово является необязательным, используется для предоставления псевдонима для внутренних действий.

- parameter: является необязательным, используется для обработки входных данных.

- result: необязательный выходной параметр для действия.

- cardinality: определяет кардинальность результирующего объекта.

- ResultParameterEntity: определяет возвращаемый тип параметра результата, это будет сущность CDS, если сущность результата действия такая же, то вместо параметра результата можно использовать ключевое слово $self, как показано на рисунке 4.31.

Пример синтаксиса:

***action*** *ChangePriority* ***parameter*** *proirity* ***result [0..1] $self***

***static action*** *CancelOrder;*

***static action*** *GetAlternateVendor* ***result [0..\*]*** *I\_Vendor;*

Теперь давайте добавим два настраиваемых действия с именами *Approve\_Order* и *Reject\_Order* в наше определение поведения, поскольку нашему приложению нужна возможность утверждать или отклонять заказы, как показано на рис. 4.31. Нам также нужно сделать несколько полей доступными только для чтения и обязательными. В этом случае мы не хотим, чтобы пользователь назначал номер документа покупки, поэтому мы сделаем поле *PurchaseDocument* доступным только для чтения, то же самое относится к полям «*Created By*», «*Created Date*», «*Last Changed By*» и «*Last Changed Date*».

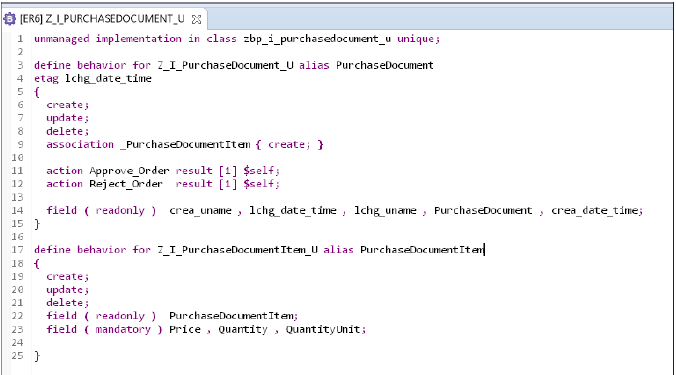


Рис. 4.31 Добавление настраиваемых кнопок действий через определение поведения

В листинге 4.20 показан полный код определения поведения с такими свойствами, как *обязательные* – ***mandatory*** и *только для чтения* – ***readonly*** для определенных полей (которые будут автоматически выбраны и добавлены к полям в приложении SAP Fiori).

*unmanaged implementation in class zbp\_i\_purchasedocument\_u unique;*

*define behavior for Z\_I\_PurchaseDocument\_U alias PurchaseDocument*

*{*

*create;*

*update;*

*delete;*

*association \_PurchaseDocumentItem { create; }*

*action Approve\_Order result [1] $self;*

*action Reject\_Order result [1] $self;*

*field ( readonly ) crea\_uname , lchg\_date\_time , lchg\_uname , PurchaseDocument , crea\_date\_time;*

*}*

*define behavior for Z\_I\_PurchaseDocumentItem\_U alias PurchaseDocumentItem*

*{*

*create;*

*update;*

*delete;*

*field ( readonly ) PurchaseDocumentItem;*

*field ( mandatory ) Price , Quantity , QuantityUnit;*

*}*

Листинг 4.20. Определение поведения (Behavior Definition)

### 4.2.6 Язык реализации поведения (BIL)

Реализация поведения реализует функции/свойства, определенные в определении поведения, и сама определяется с использованием языка реализации поведения (BIL), что является просто причудливым способом сказать, что для реализации логики используются объектно-ориентированные классы ABAP. Реализация осуществляется с помощью одного или нескольких глобальных классов. Внутри этих глобальных классов несколько локальных классов управляют поведением. В общем, два локальных класса, сгенерированные фреймворком реализации:

- Локальный класс обработчика поведения

- Поведение сохраняет локальный класс

### 4.2.7 Behavior Handler – локальный класс

Этот локальный класс определяется как абстрактный конечный класс, наследуемый от класса CL\_ABAP\_BEHAVIOR\_HANDLER. Основными методами этого класса являются MODIFY, LOCK и READ. Эти методы также определяются на основе того, какой объект должен иметь эти методы с помощью ключевого слова ***FOR <OPERATION>***.

#### (1) MODIFY

Этот метод обрабатывает все операции и действия CRUD над объектом. Операция создания определяется MODIFY с использованием синтаксиса:

***METHODS*** *<method\_name>* ***FOR MODIFY IMPORTING*** *<import\_parameter>*

***FOR CREATE*** *<entity\_name>.*

#### Modify параметры импорта

<import\_parameter> представляет собой таблицу, содержащую структуру, показанную на рис. 4.32.

Все поля сущности и ее значения:

*-* ***%cid***

Идентификатор контента, предоставляемый платформой SADL, но обычно полезный только для функции поздней нумерации. Это поле также полезно для глубокого создания сущности, когда параметр импорта дочерней сущности будет иметь другое поле *%cid\_ref*, которое содержит то же значение, что и поле *%cid* родительской сущности, что позволяет связывать записи между обеими сущностями во время выполнения.

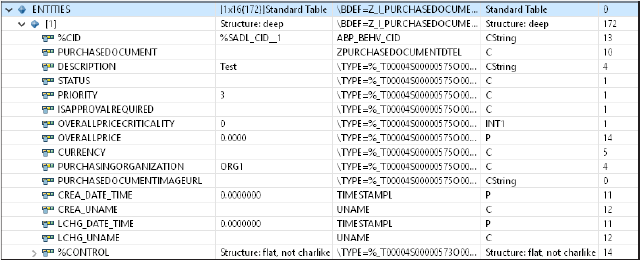


Рис. 4.32 Параметр импорта для операции создания в разделе MODIFY

- ***%control***

Структура, содержащая имя всех свойств объекта и дополнительное поле флага свойства, указывающее, было ли изменено значение поля свойства объекта пользователем или нет. Это поле необходимо при выполнении обновлений базовой таблицы в базе данных. Тип данных этого поля флага — ***ABP\_BEHV\_FLAG***, как показано на рисунке 4.33.

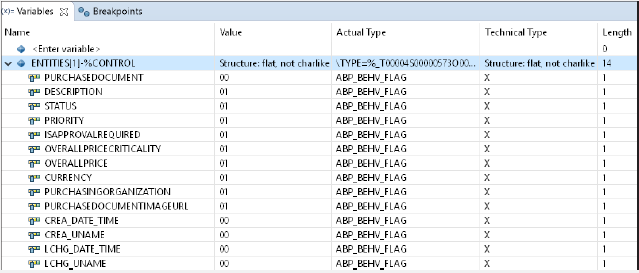


Рис. 4.33 Таблица CONTROL в возвращаемом параметре операции Create

#### Modify параметры экспорта

<export/Changing\_Parameters>: по умолчанию, когда вы смотрите на сгенерированный код класса реализации поведения, вы не увидите никаких CHANGING параметров, которые существуют как неявные параметры. Вы можете передавать значения параметрам этих методов во время выполнения без необходимости объявлять их вручную. Эта концепция также показана на рис. 4.27.

Теперь давайте посмотрим на каждую из таблиц параметров экспорта:

1) ***FAILED***

Эта таблица экспорта представляет собой вложенную таблицу с таблицей для каждой сущности, определенной в определении поведения. Преимущество заполнения этой таблицы экспорта заключается в том, что если возникает ошибка и вы хотите отменить последовательность сохранения, все, что вам нужно сделать, это вернуть ключ через эту таблицу, и последовательность сохранения будет автоматически отменена фреймворком. Как показано на рис. 4.34, поля этой таблицы следующие:

– ***%cid***: идентификатор контента.

– ***%pid***: предварительный идентификатор предоставляется приложением при создании набора сущностей. Этот атрибут используется в качестве замены, когда временный номер не используется. Этот атрибут действителен, только если определена *LATE NUMBERING*, за которой не следует определение *IN PLACE*.

– ***ID***: этот атрибут может быть одним или несколькими полями для объекта и в основном представляет собой свойства поля CDS, замаскированные как ключевые поля.

– ***%fail***: в этом атрибуте указана причина ошибки.

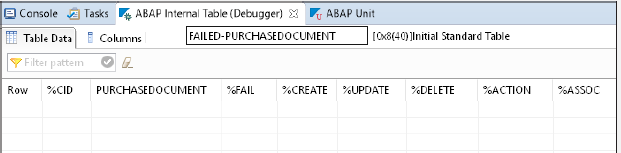


Рисунок 4.34 Таблица параметров экспорта для FAILED

2) ***MAPPED***

Эта таблица экспорта также является вложенной таблицей с одной таблицей для каждого набора сущностей, определенного в определении поведения. Параметры в этой таблице предоставляют информацию об отображении идентификаторов обратно во внешний интерфейс, когда используется функция *LATE NUMBERING*. В основном, как показано на рис. 4.35, в этой таблице хранятся созданные объекты:

– ***%cid***: идентификатор контента

– ***%pid***: предварительный идентификатор

– ***ID***: значения ключевых полей

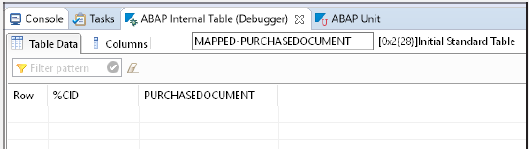


Рисунок 4.35 Таблица параметров экспорта для MAPPED

3) ***REPORTED***

Эта таблица также является таблицей экспорта с одной таблицей для каждого набора сущностей, определенного в определении поведения. Параметры в этой таблице могут использоваться для передачи сообщений, специфичных для объекта, или могут передаваться вместе с таблицей *%OTHERS* для передачи сообщений, не относящихся к набору объектов. Для сообщений, специфичных для объектов, как показано на рис. 4.36, используются следующие параметры:

– ***%CID***: идентификатор контента.

– ***%PIDC***: предварительный идентификатор.

– ***ID***: значения ключевого поля.

– ***%MSG***: это поле является экземпляром интерфейса *IF\_ABAP\_BEHV\_MESSAGE*.

– ***%ELEMENT***: Это необязательное поле имеет тип данных *ABP\_BEHV\_FLAG* (что совпадает с полем ***%control*** параметра ***CHANGING***).

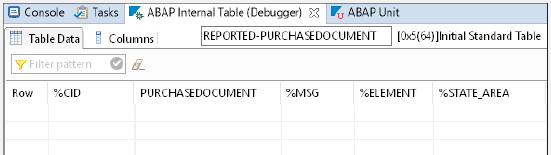


Рисунок 4.36 Таблица параметров экспорта для REPORTED

Для общих сообщений используется в ***REPORTED*** таблица ***REPORTED-%OTHERS***. Этот тип строки таблицы является ссылкой на интерфейс *IF\_ABAP\_BEHV\_MESSAGE*, как показано на рисунке 4.37.

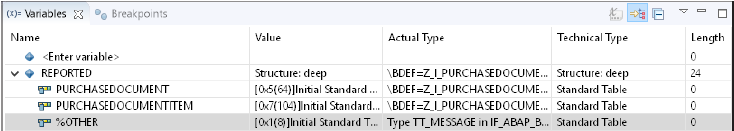


Рисунок 4.37 Таблица параметров экспорта для REPORTED-%OTHERS

#### Создать по ассоциации (Create by Association)

Создать по ассоциации — это подмножество операции создания, которая используется для создания экземпляров дочерних объектов с использованием свойства ассоциации. Параметр импорта для этой операции будет содержать ключевое поле родительского объекта и таблицу, содержащую данные для дочернего объекта, как показано ниже:

***METHODS*** *<method\_name>* ***FOR MODIFY IMPORTING*** *<import\_parameter>*

***FOR CREATE*** *<entity\_name>\<association\_name>.*

#### Create by Association параметры импорта

Поля *Create by Association таблицы <import\_parameter>* содержат следующие параметры, показанные на рис. 4.38:

- ***Все ключевые поля родительской сущности***.

- ***%cid\_ref***: ссылка на идентификатор контента в дочернем объекте.

- ***%target***: еще одна таблица, содержащая данные для дочерней сущности, как показано на рис. 4.39.

- ***%cid***: все свойства дочерней сущности.

- ***%control***: Другая структура, которая содержит все свойства поля представления CDS в первом столбце и флаг свойства (указывающий, было ли изменено значение поля свойства объекта пользователем или нет) во втором столбце тип данных *ABP\_BEHV\_FLAG*.

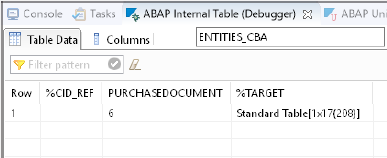


Рисунок 4.38 Таблица параметров импорта для создания по ассоциации

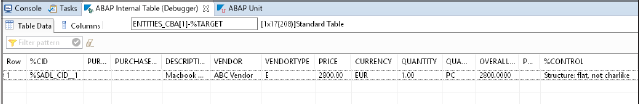


Рисунок 4.39 Таблица TARGET внутри таблицы параметров импорта для создания по ассоциации

#### Create by Association параметры экспорта

Как и другие операции оператора *MODIFY*, операция обновления также имеет следующие поля *CHANGING*:

- ***FAILED***: Та же таблица, что и в параметрах экспорта при операции создания.

- ***REPORTED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***MAPPED***: та же таблица, что и в параметрах экспорта при операции создания.

#### Update

Операция обновления определяется с использованием метода MODIFY с использованием следующего синтаксиса:

***METHODS*** *<method\_name>* ***FOR MODIFY IMPORTING*** *<import\_parameter>*

***FOR UPDATE*** *<entity\_name>.*

#### Update параметры импорта и changing

Поля обновления таблицы *<import\_parameter>* содержат следующие поля, также показанные на рис. 4.40:

- ***Все свойства объекта.***

- ***%cid\_ref***: ссылка на идентификатор контента в дочернем объекте.

- ***%pid***: предварительный идентификатор.

- ***%key***: все ключевые поля базового представления CDS.

- ***%data***: содержит данные всех полей представления CDS.

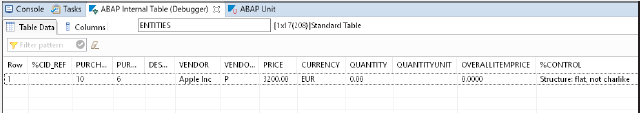


Рисунок 4.40 Таблица параметров импорта для обновления

***%control*** — еще одна структура, содержащая все свойства поля представления CDS в первом столбце и флаг свойства, указывающий, было ли изменено значение поля свойства сущности пользователем или нет – во втором столбце типа данных *ABP\_BEHV\_FLAG*, как показано на рисунке 4.41. Столбцы с новыми значениями отмечены значком значение «*01*», а неизмененные значения поля по умолчанию отмечены значением «*00*».

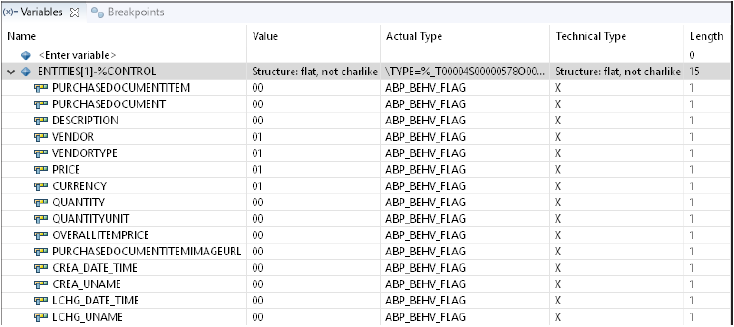


Рисунок 4.41 Таблица CONTROL внутри таблицы параметров импорта для обновления

Как и другие операции оператора *MODIFY*, операция обновления также имеет поля *CHANGING*:

- ***FAILED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***REPORTED***: та же таблица, что и в параметрах экспорта при операции создания.

#### Delete

Операция удаления определяется с использованием метода *MODIFY* с использованием следующего синтаксиса:

***METHODS*** *<method\_name>* ***FOR MODIFY IMPORTING*** *<import\_parameter>*

***FOR DELETE*** *<entity\_name>.*

#### Delete параметры импорта

Поля для удаления таблицы *<import\_parameter>* содержат следующие поля, как показано на рис. 4.42:

- ***ID fields***: все ключевые поля базового представления CDS.

- ***%cid\_ref***: ссылка на идентификатор содержимого в дочернем объекте.

- ***%pid***: предварительный идентификатор.

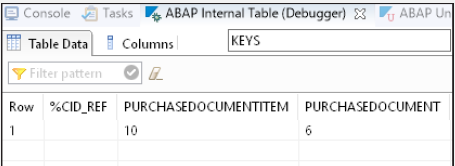


Рисунок 4.42 Таблица параметров импорта для удаления

#### Delete параметры экспорта и changing

Как и другие операции оператора *MODIFY*, операция удаления также имеет следующие поля *CHANGING*:

- ***FAILED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***REPORTED***: та же таблица, что и в параметрах экспорта при операции создания.

#### Action

Действие определяется с помощью метода *MODIFY* с использованием следующего синтаксиса:

#### Action параметры импорта

Поля импорта таблицы *<import\_parameter>* содержат следующие поля, как показано на рис. 4.43:

- ***ID fields***: все ключевые поля базового представления CDS.

- ***%cid\_ref***: ссылка на идентификатор контента в дочернем объекте

- ***%pid***: предварительный идентификатор

- ***%param***: необязательное поле, используемое, если определены какие-либо параметры импорта.

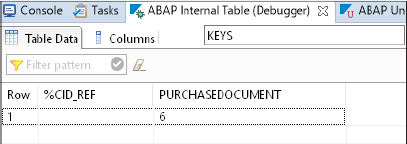


Рисунок 4.43 Таблица параметров импорта для действия

#### Action параметры экспорта и changing

Как и другие операции оператора *MODIFY*, операция удаления также имеет следующие поля *CHANGING*:

- ***FAILED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***REPORTED***: та же таблица, что и в параметрах экспорта при операции создания.

Необязательный параметр *<result\_parameter>* может содержать результирующие данные действия, если он определен в определении поведения.

#### (2) LOCK

Этот метод определяется с использованием следующего синтаксиса:

***METHODS*** *<method\_name>* ***FOR LOCK*** *<import\_parameter>*

***FOR LOCK*** *<entity\_name>.*

#### Lock параметры импорта

Поля импорта таблицы *<import\_parameter>* содержат следующие параметры:

- ***ID fields***: все ключевые поля базового представления CDS.

- ***%cid\_ref***: ссылка на идентификатор содержимого в дочернем объекте.

- ***%pid***: предварительный идентификатор

#### Lock параметры экспорта и changing

Для таблицы *<export\_parameters>*, как и для других операций инструкции *MODIFY*, операция блокировки также имеет следующие поля *CHANGING*:

- ***FAILED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***REPORTED***: та же таблица, что и в параметрах экспорта при операции создания.

#### (3) READ

Этот метод предназначен для доступа только для чтения. У этого метода есть два варианта: *чтение по ключу* и *чтение по ассоциации*. Далее рассмотрим подробно каждый вариант.

#### Read by Key

Этот метод имеет следующий синтаксис:

***METHODS*** *<read>* ***FOR READ IMPORTING*** *<keys>* ***FOR READ*** *<entity\_name>* ***RESULT***

*<exporting\_parameter>*

#### Read параметры импорта

Поля импорта таблицы *<import\_parameter>* содержат следующие параметры:

- ***ID fields***: все ключевые поля базового представления CDS.

- ***%pid***: предварительный идентификатор.

- ***%control***: структура, содержащая все свойства поля представления CDS в первом столбце и флаг свойства во втором столбце типа данных ABP\_BEHV\_FLAG.

#### Read параметры экспорта

Как и другие операции оператора MODIFY, операция чтения также имеет следующие поля CHANGING:

- ***FAILED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***REPORTED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***<export\_parameter>***: данные из оператора чтения.

- ***%pid***: предварительный идентификатор.

- ***Все свойства целевого объекта с их значениями***.

#### Read by Association

Чтение по ассоциации имеет следующий синтаксис:

***METHODS*** *<read>* ***FOR READ IMPORTING*** *<import\_parameter>* ***FOR READ*** *<parent\_entity\_name>\<\_association> FULL <result\_requested> RESULT*

***RESULT*** *<exporting\_parameter> LINK <association\_links>*

#### Read by Associationпараметры импорта

Поля Read by Association таблицы *<import\_parameter>* содержат следующие параметры:

- ***ID fields***: все ключевые поля базового представления CDS.

- ***%pid***: предварительный идентификатор.

- ***<parent\_entity\_name>\<\_association>***: имя родительского объекта и дочернего объекта или любая другая связь с родительским объектом.

- ***<result\_requested>***: индикатор. Если <result\_requested> является INITIAL, то возвращается только таблица возврата, связанная с LINK (проверьте приведенный выше синтаксис для Read by Association). Если <result\_requested> не INITIAL, то возвращаются таблицы возврата, связанные как с LINK, так и с RESULT.

#### Read by Association параметры экспорта

Как и другие операции оператора MODIFY, операция обновления также имеет следующие поля CHANGING:

- ***FAILED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***REPORTED***: та же таблица, что и в параметрах экспорта при операции создания.

- ***<association\_links>***: этот тип возвращаемой таблицы содержит родительский и дочерний ключи/ключи ассоциации.

- ***<exporting\_parameter>***: этот параметр возвращает данные из запроса на чтение.

Как упоминалось ранее, возвращаемый параметр RESULT будет заполнен только в том случае, если <result\_requested> не является INITIAL.

– ***%pid***: предварительный идентификатор.

– ***Все поля целевых сущностей с их значениями***.

#### Синтаксис MODIFY, LOCK и READ

Следующие примеры синтаксиса охватывают все операции, основанные на *MODIFY*, *LOCK* и *READ* (см. также Раздел 4.2.7, подраздел «*MODIFY*»):

***METHODS*** *<create>* ***FOR MODIFY IMPORTING*** *<entities>* ***FOR CREATE*** *<entity\_name>.*

***METHODS*** *<update>* ***FOR MODIFY IMPORTING*** *<entities>* ***FOR UPDATE*** *<entity\_name>.*

***METHODS*** *<delete>* ***FOR MODIFY IMPORTING*** *<keys>* ***FOR DELETE*** *<entity\_name>.*

***METHODS*** *<approve\_order>* ***FOR MODIFY IMPORTING*** *<keys>* ***FOR ACTION*** *<approve\_order>*

***RESULT*** *<result\_parameter>.*

***METHODS*** *<lock>* ***FOR LOCK IMPORTING*** *<import\_parameter>* ***FOR LOCK*** *<entity\_name>.*

***METHODS*** *<read>* ***FOR MODIFY IMPORTING*** *<keys>* ***FOR CREATE*** *<entity\_name>*

*<entity\_name\\_association\_name>. - (for Create by Association)*

***METHODS*** *<read>* ***FOR READ IMPORTING*** *<keys>* ***FOR READ***

*<entity\_name\\_association\_name>* ***FULL*** *<result\_requested>* ***RESULT LINK*** *<result\_*

*parameter>* ***LINK*** *association\_links*

***METHODS*** *read* ***FOR MODIFY IMPORTING*** *<import\_parameter>* ***FOR FUNCTION*** *<entity\_*

*name~action\_name>* ***RESULT*** *<result\_parameter>*

*Примечание*

*Данные, передаваемые в операции MODIFY, на самом деле не сохраняются в базе данных, как показано ранее на рис. 4.27. На самом деле эти данные хранятся в промежуточных буферных таблицах. Данные будут сохранены в базе данных с помощью транзакционных методов в локальных классах сохранения поведения, которые мы обсудим в следующем разделе.*

### 4.2.8 Behavior Saver – локальный класс

Этот класс (концепция, обсуждавшаяся ранее и показанная на рис. 4.27) реализован как локальный класс в пуле определений поведения, чтобы включить транзакционные методы, такие как *finalize, check\_before\_save, save* и т. д. Этот класс определен как абстрактный конечный класс и наследует из CL\_ABAP\_BEHAVIOR\_SAVER, как показано в листинге 4.21.

CLASS lcl\_behavior\_saver DEFINITION INHERITING FROM cl\_abap\_behavior\_saver

ABSTRACT FINAL.

PROTECTED SECTION.

METHODS finalize REDEFINITION.

METHODS check\_before\_save REDEFINITION.

METHODS adjust\_numbers REDEFINITION.

METHODS save REDEFINITION.

METHODS cleanup REDEFINITION.

ENDCLASS.

Листинг 4.21. Локальный класс Behavior Saver

Давайте кратко рассмотрим каждый из методов в локальном классе сохранения поведения:

1) ***Finalize***

Данные дорабатываются перед сохранением в базе данных. Этот метод также имеет параметры экспорта *FAILED* и *REPORTED*, которые имеют тот же тип, что и параметры экспорта для операций CRUD, которые мы обсуждали ранее в Разделе 4.2.7 о локальном классе обработчика поведения, с той лишь разницей, что *%cid* не является частью этих параметров.

2) ***check\_before\_save***

Проверка данных в буферной таблице на непротиворечивость, этот метод является последней точкой, где можно откатить COMMIT данных в базу данных. В случае ошибки следующий метод в последовательности, *save*, можно пропустить, заполнив параметр экспорта *FAILED*. Также метод *cleanup* может быть вызван для очистки буферных таблиц. Этот метод имеет следующие параметры экспорта:

– *FAILED*: та же таблица, что и для параметра экспорта операций CRUD, за исключением поля *%cid*.

– *REPORTED*: та же таблица, что и для параметра экспорта операций CRUD, за исключением поля *%cid*.

3) ***Save***: с помощью этого метода извлекаются значения из буферных таблиц и выполняется окончательная фиксация данных в базе данных.

4) ***cleanup***

Этот метод содержит логику для очистки всех буферных таблиц и других связанных переменных, которые необходимо инициализировать. Этот метод может быть вызван любым из вышеперечисленных методов в классе. Например, методы *finalize* и *check\_before\_save* будут вызывать метод *cleanup* в случае обнаружения каких-либо ошибок в данных, требующих отката операции. Вы также можете вызвать этот метод после сохранения, чтобы все очистить.

### 4.2.9 Behavior Implementation - Реализация поведения

Реализация поведения создаст соответствующие классы обработчиков для логики транзакционных операций, ссылаясь на соответствующие операции, определенные в определении поведения.

#### Создание новой реализации поведения

Теперь давайте создадим реализацию поведения. Щелкните правой кнопкой мыши Behavior Definition. Затем выберите папку «Новая реализация поведения», как показано на рис. 4.44.

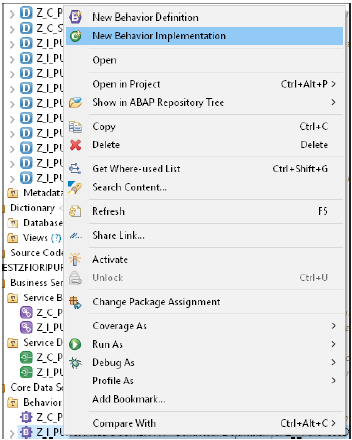


Рисунок 4.44 Создание реализации поведения

Укажите то же имя для класса реализации поведения, которое было автоматически сгенерировано платформой в определении поведения, как показано на рис. 4.45. Введите информацию в поле «Описание» и нажмите «Далее».

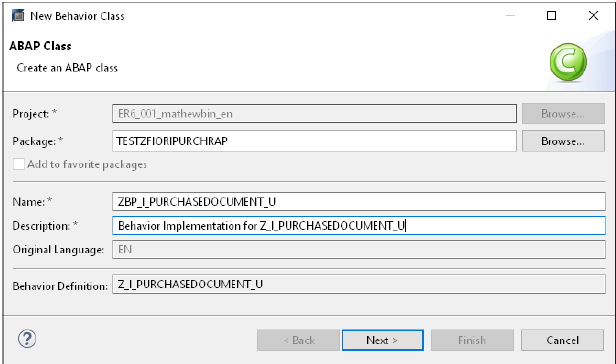


Рисунок 4.45 Детали класса реализации поведения

Пул определений поведения или глобальный класс, показанный на рис. 4.46, будет иметь последующую реализацию локальных классов обработчиков поведения для обработки логики операций CRUD, блокировок и т. д. Этот глобальный класс также имеет локальный класс-сохранитель для фиксации изменений из CRUD-операции.

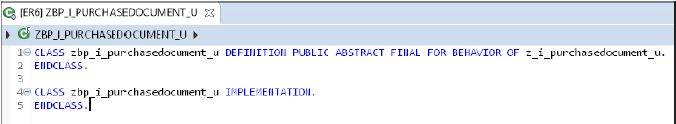


Рис. 4.46 Пул определений поведения или глобальный класс

После реализации класса поведения щелкните вкладку «Локальные типы» в нижней части класса. Вы увидите, что были сгенерированы два набора локальных классов. Первый, *lhc\_purchaseDocument*, показанный на рис. 4.47, наследуется от *CL\_ABAP\_BEHAVIOR\_HANDLER* (который обрабатывает действия над бизнес-объектом, определенные Behavior Definition, такие как операции CRUD или пользовательские действия). Здесь методы использовали типизированные параметры, в отличие от предыдущей модели разработки (модель программирования ABAP для SAP Fiori), и поэтому с ними намного проще работать, поскольку типы параметров являются прямыми ссылками на типы сущностей, на которых основаны параметры. Второй локальный класс *lsc\_Z\_I\_PurchaseDocument\_U*, показанный на рис. 4.48, наследуется от *CL\_ABAP\_BEHAVIOR\_SAVER* (для обработки части сохранения/фиксации транзакции).

Полный код класса показан в листинге 4.22.

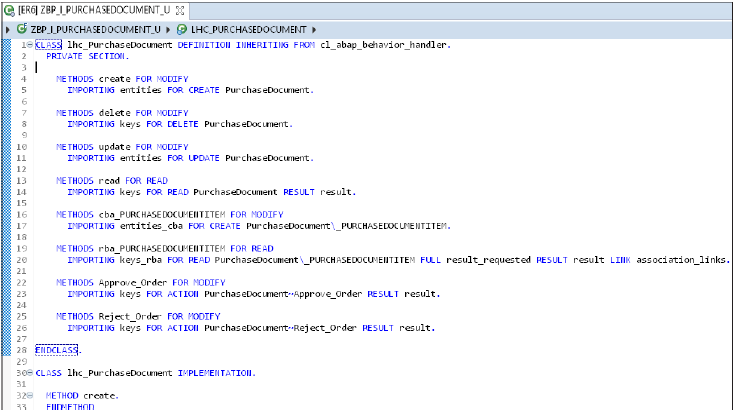


Рис. 4.47 Локальный класс реализации поведения для обработки действий пользователя

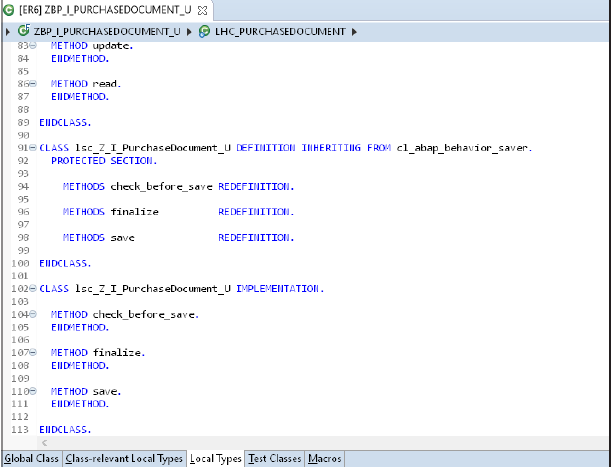


Рис. 4.47 Локальный класс реализации поведения для обработки действий пользователя

В листинге 4.22 приведен полный листинг реализации класса локального обработчика для документа о покупке.

CLASS lhc\_PurchaseDocument DEFINITION INHERITING FROM cl\_abap\_behavior\_handler.

PRIVATE SECTION.

METHODS create FOR MODIFY

IMPORTING entities FOR CREATE PurchaseDocument.

METHODS delete FOR MODIFY

IMPORTING keys FOR DELETE PurchaseDocument.

METHODS update FOR MODIFY

IMPORTING entities FOR UPDATE PurchaseDocument.

METHODS read FOR READ

IMPORTING keys FOR READ PurchaseDocument RESULT result.

METHODS cba\_PURCHASEDOCUMENTITEM FOR MODIFY

IMPORTING entities\_cba FOR CREATE PurchaseDocument\\_purchasedocumentitem.

METHODS rba\_PURCHASEDOCUMENTITEM FOR READ

IMPORTING keys\_rba FOR READ PurchaseDocument\\_purchasedocumentitem

FULL result\_requested RESULT result LINK association\_links.

METHODS Approve\_Order FOR MODIFY

IMPORTING keys FOR ACTION PurchaseDocument~Approve\_Order RESULT result.

METHODS Reject\_Order FOR MODIFY

IMPORTING keys FOR ACTION PurchaseDocument~Reject\_Order RESULT result.

ENDCLASS.

CLASS lhc\_PurchaseDocument IMPLEMENTATION.

METHOD create.

ENDMETHOD.

METHOD delete.

ENDMETHOD.

METHOD update.

ENDMETHOD.

METHOD read.

ENDMETHOD.

METHOD cba\_PURCHASEDOCUMENTITEM.

ENDMETHOD.

METHOD rba\_PURCHASEDOCUMENTITEM.

ENDMETHOD.

METHOD Approve\_Order.

ENDMETHOD.

METHOD Reject\_Order.

ENDMETHOD.

ENDCLASS.

CLASS lhc\_PurchaseDocumentItem DEFINITION INHERITING FROM cl\_abap\_behavior\_handler.

PRIVATE SECTION.

METHODS create FOR MODIFY

IMPORTING entities FOR CREATE PurchaseDocumentItem.

METHODS delete FOR MODIFY

IMPORTING keys FOR DELETE PurchaseDocumentItem.

METHODS update FOR MODIFY

IMPORTING entities FOR UPDATE PurchaseDocumentItem.

METHODS read FOR READ

IMPORTING keys FOR READ PurchaseDocumentItem RESULT result.

DATA et\_messages TYPE zif\_prchdc\_logic=>tt\_if\_t100\_message.

ENDCLASS.

CLASS lhc\_PurchaseDocumentItem IMPLEMENTATION.

METHOD create.

ENDMETHOD.

METHOD delete.

ENDMETHOD.

METHOD update.

ENDMETHOD.

METHOD read.

ENDMETHOD.

ENDCLASS.

CLASS lsc\_Z\_I\_PurchaseDocument\_U DEFINITION INHERITING FROM cl\_abap\_behavior\_saver.

PROTECTED SECTION.

METHODS check\_before\_save REDEFINITION.

METHODS finalize REDEFINITION.

METHODS save REDEFINITION.

METHODS cleanup REDEFINITION.

ENDCLASS.

CLASS lsc\_Z\_I\_PurchaseDocument\_U IMPLEMENTATION.

METHOD check\_before\_save.

ENDMETHOD.

METHOD finalize.

ENDMETHOD.

METHOD save.

ENDMETHOD.

METHOD cleanup.

ENDMETHOD.

ENDCLASS.

Листинг 4.22. Класс реализации пула действий для приложения PurchaseDocument

#### Обзор кодирования класса реализации поведения

Прежде чем перейти к кодированию для реализации в этих локальных классах, давайте рассмотрим обзор зависимых классов и методов, которые мы будем использовать в нашем примере, как показано на рис. 4.49, для реализации нашего пула поведения.

Локальные классы обработчиков поведения *lhc\_PurchaseDocument* и *lhc\_PurchaseDocumentItem* (которые принадлежат основному глобальному классу *zpb\_i\_purchasedocument\_u*) для документа покупки и позиции документа покупки соответственно. Соответствующий класс сохранения поведения *lsc\_Z\_I\_PurchaseDocument\_U* будет зависеть от классов для обработки логики транзакций в их соответствующих методах.

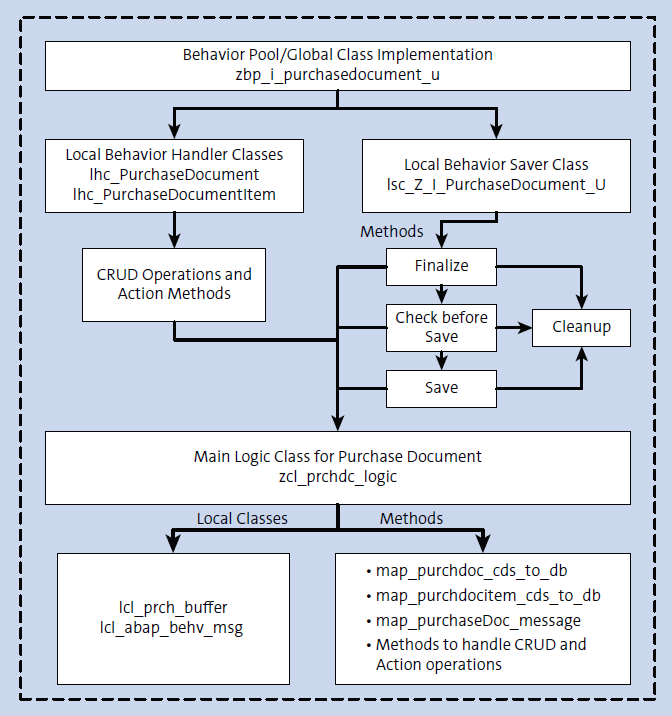


Рис. 4.49. Логика реализации пула действий для приложения PurchaseDocument Management.

Для обработки логики CRUD и методов действий в классе обработчика поведения мы создадим новый класс с именем *zcl\_prchdc\_logic*, который будет действовать как мост для хранения и передачи данных между классом локального обработчика и классом сохранения. В *zcl\_prchdc\_logic* есть несколько методов и локальных классов для обработки логики наших приложений:

1) ***map\_purchdoc\_cds\_to\_db***

Как упоминалось ранее в разделе 4.2.6, параметры импорта операций CRUD и методы действий имеют несколько полей в дополнение к фактическим данным, которые нам нужно сохранить в соответствующей таблице базы данных. Итак, нам нужно сопоставить соответствующие поля с совместимой структурой, а затем мы можем сохранить их в буферной таблице, чтобы позже сохранить их в классе сохранения. В следующих методах вы можете видеть, что входящая структура из операций сущности PurchaseDocument сопоставляется с соответствующей структурой таблицы базы данных для документа покупки, что помогает хранить данные в буферной таблице для последующего сохранения данных непосредственно в соответствующую таблицу базы данных (см. листинг 4.23).

CLASS-METHODS map\_purchdoc\_cds\_to\_db

IMPORTING is\_i\_purchdoc\_u TYPE Z\_I\_PurchaseDocument\_U

RETURNING VALUE(rs\_purchdoc) TYPE zif\_prchdc\_logic=>ts\_purchasedocument.

METHOD map\_purchdoc\_cds\_to\_db.

rs\_purchdoc = CORRESPONDING #(

is\_i\_purchdoc\_u MAPPING purchasedocument = PurchaseDocument

crea\_date\_time = crea\_date\_time

crea\_uname = crea\_uname

description = Description

lchg\_date\_time = lchg\_date\_time

lchg\_uname = lchg\_uname

priority = Priority

purchasedocumentimageurl = PurchaseDocumentImageURL

purchasingorganization = PurchasingOrganization

status = status ).

ENDMETHOD.

Листинг 4.23. Метод сопоставления CDS документа покупки с базой данных

2) ***map\_purchdocitem\_cds\_to\_db***

С той же целью, что и у предыдущего метода, этот метод используется для сопоставления элементов документа покупки с соответствующей структурой таблицы базы данных вместо документа покупки, как показано в листинге 4.24.

CLASS-METHODS map\_purchdocitem\_cds\_to\_db

IMPORTING is\_i\_purchdocitem\_u TYPE z\_i\_purchasedocumentitem\_u

RETURNING VALUE(rs\_purchdocitem) TYPE zif\_prchdc\_logic=>ts\_purchasedocitem.

METHOD map\_purchdocitem\_cds\_to\_db.

rs\_purchdocitem = CORRESPONDING #(

is\_i\_purchdocitem\_u MAPPING purchasedocument = PurchaseDocument

crea\_date\_time = crea\_date\_time

crea\_uname = crea\_uname

description = Description

lchg\_date\_time = lchg\_date\_time

lchg\_uname = lchg\_uname

purchasedocumentitem = PurchaseDocumentItem

quantity = Quantity

quantityunit = QuantityUnit

vendor = Vendor

vendortype = VendorType

price = Price

purchasedocumentitemimageurl = PurchaseDocumentItemImageURL

currency = Currency ).

ENDMETHOD.

Листинг 4.24. Метод сопоставления CDS позиции документа покупки с базой данных

3) ***map\_purchaseDoc\_message***

Точно так же, как вам нужно сопоставить входящие поля для сохранения данных в базу данных, такой же тип сопоставления требуется для заполнения содержимого таблиц экспорта, таких как *FAILED, MAPPED* и *REPORTED*. Этот метод сопоставляет данные с соответствующими полями этих экспортируемых параметров в зависимости от требования. Например, листинг 4.25 используется для сопоставления ответа со структурой таблицы *REPORTED* параметра экспорта объекта PurchaseDocument.

CLASS-METHODS map\_purchaseDoc\_message

IMPORTING iv\_cid TYPE string OPTIONAL

iv\_purchasedocument TYPE zpurchasedocumentdtel OPTIONAL

is\_message TYPE LINE OF zif\_prchdc\_logic=>tt\_if\_t100\_message

is\_messageType TYPE symsgty

RETURNING VALUE(rs\_report) TYPE LINE OF

zif\_prchdc\_logic=>tt\_purchasedocumet\_reported.

METHOD map\_purchaseDoc\_message.

DATA: ls\_message TYPE LINE OF tt\_message.

MOVE-CORRESPONDING is\_message->t100key TO ls\_message.

ls\_message-msgty = is\_messageType.

ls\_message-msgv1 = iv\_purchasedocument.

DATA(lo) = new\_message( id = ls\_message-msgid

number = ls\_message-msgno

severity = if\_abap\_behv\_message=>severity-error

v1 = ls\_message-msgv1

v2 = ls\_message-msgv2

v3 = ls\_message-msgv3

v4 = ls\_message-msgv4 ).

rs\_report-%cid = iv\_cid.

rs\_report-PurchaseDocument = iv\_purchasedocument.

rs\_report-%msg = lo.

ENDMETHOD.

Листинг 4.25. Метод сопоставления сообщения документа покупки

Теперь давайте рассмотрим локальные классы:

- ***lcl\_abap\_behv\_msg***

Этот класс создает сообщения об исключениях и по мере необходимости передает эти сообщения в соответствующие таблицы возвращаемых параметров соответствующих операций.

- ***lcl\_prch\_buffer***

Этот локальный класс отвечает за хранение всех данных, передаваемых из класса поведения, и за сохранение этих данных в соответствующих буферных таблицах. Позже к этой же буферной таблице будет обращаться класс-сохранитель для окончательной фиксации данных в таблицах базы данных. Как показано в листинге 4.26, мы объявили буферные таблицы на основе соответствующих структур таблиц базы данных. Мы также определили методы для обработки различных транзакционных функций класса обработчика поведения, а также методы класса сохранения поведения.

CLASS lcl\_prch\_buffer DEFINITION FINAL CREATE PRIVATE.

PUBLIC SECTION.

" Buffer Tables

DATA:

mt\_create\_buffer\_PurchDoc TYPE zif\_prchdc\_logic=>tt\_purchasedocument,

mt\_update\_buffer\_PurchDoc TYPE zif\_prchdc\_logic=>tt\_purchasedocument,

mt\_delete\_buffer\_PurchDoc TYPE zif\_prchdc\_logic=>tt\_purchasedocumentKey,

mt\_create\_buffer\_PurchDocItem TYPE zif\_prchdc\_logic=>tt\_purchdocumentitem,

mt\_update\_buffer\_PurchDocItem TYPE zif\_prchdc\_logic=>tt\_purchdocumentitem,

mt\_delete\_buffer\_PurchDocItem TYPE zif\_prchdc\_logic=>tt\_purchdocitem\_key.

CLASS-METHODS: get\_instance RETURNING VALUE(ro\_instance) TYPE REF TO

lcl\_prch\_buffer.

METHODS buffer\_PurchDoc\_for\_Create

IMPORTING it\_purchaseDocument TYPE zif\_prchdc\_logic=>tt\_purchasedocument

EXPORTING et\_purchaseDocument TYPE zif\_prchdc\_logic=>tt\_purchasedocument

et\_messages TYPE zif\_prchdc\_logic=>tt\_if\_t100\_message.

METHODS buffer\_purchdoc\_for\_delete

IMPORTING it\_purchaseDocumentkey TYPE zif\_prchdc\_logic=>tt\_purchasedocumentkey

it\_purchaseDocitemtkey TYPE zif\_prchdc\_logic=>tt\_purchdocitem\_key

EXPORTING et\_messages TYPE zif\_prchdc\_logic=>tt\_if\_t100\_message.

METHODS buffer\_purchdoc\_for\_update

IMPORTING it\_purchaseDocument TYPE zif\_prchdc\_logic=>tt\_purchasedocument

EXPORTING et\_messages TYPE zif\_prchdc\_logic=>tt\_if\_t100\_message.

METHODS buffer\_PurchDocItem\_for\_create

IMPORTING it\_purchaseDocItem TYPE zif\_prchdc\_logic=>tt\_purchdocumentitem

EXPORTING et\_messages TYPE zif\_prchdc\_logic=>tt\_if\_t100\_message.

METHODS buffer\_purchdocitem\_for\_update

IMPORTING it\_purchaseDocItem TYPE zif\_prchdc\_logic=>tt\_purchdocumentitem

EXPORTING et\_messages TYPE zif\_prchdc\_logic=>tt\_if\_t100\_message.

METHODS buffer\_purchdocitem\_for\_delete

IMPORTING it\_purchaseDocItemKey TYPE zif\_prchdc\_logic=>tt\_purchdocitem\_key

EXPORTING et\_messages TYPE zif\_prchdc\_logic=>tt\_if\_t100\_message.

METHODS save.

METHODS initialize.

PRIVATE SECTION.

CLASS-DATA go\_instance TYPE REF TO lcl\_prch\_buffer.

TYPES:

BEGIN OF ts\_purchasedocument\_id,

purchasedocument TYPE zpurchasedocumentdtel,

END OF ts\_purchasedocument\_id,

tt\_purchasedocument\_id TYPE SORTED TABLE OF ts\_purchasedocument\_id

WITH UNIQUE KEY purchasedocument.

DATA lt\_purchasedocument\_id TYPE tt\_purchasedocument\_id.

ENDCLASS.

Листинг 4.26. Определение класса буфера

Как показано в листинге 4.27, для класса буфера реализованы соответствующие методы для обработки различных транзакционных функций и методы класса сохранения. Например, если вы проверяете метод *buffer\_purchDoc\_for\_Create*, мы проверяем разные сценарии, например, присутствует ли номер входящего документа покупки в базе данных или нет, является ли поле документа покупки исходным или нет и т. д. Эти проверки помогут нам с основными проверками перед сохранением данных в буферных таблицах. Если какая-либо из этих основных проверок ошибочна, мы сгенерируем ответ с соответствующим сообщением об исключении и вызовем метод инициализации для очистки буферных таблиц. Если все проверки верны, то таблица сообщений останется пустой, а данные будут сохранены в соответствующей буферной таблице.

В методе сохранения при его вызове мы извлекаем значения из буферной таблицы и сохраняем данные в соответствующей таблице базы данных. Вы увидите, как эти методы класса вызываются обработчиком поведения и классом сохранения поведения в следующих разделах.

CLASS lcl\_prch\_buffer IMPLEMENTATION.

METHOD get\_instance.

go\_instance = COND #( WHEN go\_instance IS BOUND THEN go\_instance

ELSE NEW #( ) ).

ro\_instance = go\_instance.

ENDMETHOD.

METHOD buffer\_PurchDoc\_for\_Create.

CLEAR: et\_purchaseDocument,

et\_messages,

lt\_purchasedocument\_id.

CHECK it\_purchaseDocument IS NOT INITIAL.

SELECT FROM zpurchdocument

FIELDS purchasedocument

FOR ALL ENTRIES IN @it\_purchaseDocument

WHERE purchasedocument = @it\_purchaseDocument-purchasedocument

INTO CORRESPONDING FIELDS OF TABLE @lt\_purchasedocument\_id.

LOOP AT it\_purchaseDocument INTO DATA(ls\_purchdoc\_create) ##INTO\_OK.

" Check Purchase Document number is initial or not

IF ls\_purchdoc\_create-purchasedocument IS INITIAL.

"add exception to message class if purchase document ID is initial

APPEND NEW zcx\_purchdoc\_excptns( textid = zcx\_purchdoc\_excptns=>

purchasedocintial mv\_purchasedocument = ls\_purchdoc\_create-purchasedocument )

TO et\_messages.

initialize( ).

RETURN.

ENDIF.

" Check if the Purchase Document ID already Exists

IF line\_exists( lt\_purchasedocument\_id[ purchasedocument

= ls\_purchdoc\_create-purchasedocument ] ).

"add exception to message class if Purchase Do cument ID exists

APPEND NEW zcx\_purchdoc\_excptns( textid = zcx\_purchdoc\_excptns=>

purchasedocexists mv\_purchasedocument = ls\_purchdoc\_create-purchasedocument )

TO et\_messages.

initialize( ).

RETURN.

ENDIF.

" Check in buffer table if the Purchase Document ID already exists or not

IF line\_exists( mt\_create\_buffer\_PurchDoc[ purchasedocument

= ls\_purchdoc\_create-purchasedocument ] ).

"add exception to message class if Purchase Document ID exists in buffer table

APPEND NEW zcx\_purchdoc\_excptns( textid = zcx\_purchdoc\_excptns=>

purchasedocexitsinbuffer mv\_purchasedocument =

ls\_purchdoc\_create-purchasedocument )

TO et\_messages.

initialize( ).

RETURN.

ENDIF.

INSERT ls\_purchdoc\_create INTO TABLE mt\_create\_buffer\_PurchDoc.

ENDLOOP.

et\_purchaseDocument = mt\_create\_buffer\_PurchDoc.

ENDMETHOD.

METHOD buffer\_purchdoc\_for\_delete.

CLEAR: et\_messages.

CHECK it\_purchaseDocumentkey IS NOT INITIAL.

MOVE-CORRESPONDING it\_purchasedocumentkey TO mt\_delete\_buffer\_PurchDoc.

MOVE-CORRESPONDING it\_purchasedocitemtkey TO mt\_delete\_buffer\_purchdocitem.

ENDMETHOD.

METHOD buffer\_purchdoc\_for\_update.

CLEAR: et\_messages.

CHECK it\_purchasedocument IS NOT INITIAL.

MOVE-CORRESPONDING it\_purchasedocument TO mt\_update\_buffer\_purchdoc.

ENDMETHOD.

METHOD buffer\_PurchDocItem\_for\_create.

CLEAR et\_messages.

CHECK it\_purchasedocitem IS NOT INITIAL.

MOVE-CORRESPONDING it\_purchasedocitem TO mt\_create\_buffer\_PurchDocItem.

ENDMETHOD.

METHOD buffer\_purchdocitem\_for\_update.

CLEAR: et\_messages.

CHECK it\_purchasedocitem IS NOT INITIAL.

MOVE-CORRESPONDING it\_purchasedocitem TO mt\_update\_buffer\_purchdocitem.

ENDMETHOD.

METHOD buffer\_purchdocitem\_for\_delete.

CLEAR: et\_messages.

CHECK it\_purchasedocitemkey IS NOT INITIAL.

MOVE-CORRESPONDING it\_purchasedocitemkey TO mt\_delete\_buffer\_purchdocitem.

ENDMETHOD.

METHOD save.

"Here in the save method, the corresponding buffer tables are read

"and the relevant data is updated,deleted or created in the

"PurchaseDocument and PurchasedocumentItem tables

INSERT zpurchdocument FROM TABLE @mt\_create\_buffer\_PurchDoc.

UPDATE zpurchdocument FROM TABLE @mt\_update\_buffer\_PurchDoc.

DELETE zpurchdocument FROM TABLE

@( CORRESPONDING #( mt\_delete\_buffer\_PurchDoc ) ).

"Same logic is applied to the PurchaseDocument Table

INSERT zpurchdocitem FROM TABLE @mt\_create\_buffer\_PurchDocItem.

UPDATE zpurchdocitem FROM TABLE @mt\_update\_buffer\_PurchDocItem.

DELETE zpurchdocitem FROM TABLE

@( CORRESPONDING #( mt\_delete\_buffer\_PurchDocItem ) ).

ENDMETHOD.

METHOD initialize.

CLEAR: mt\_create\_buffer\_PurchDoc,

mt\_update\_buffer\_purchdoc,

mt\_delete\_buffer\_purchdoc,

mt\_delete\_buffer\_PurchDocItem,

mt\_update\_buffer\_PurchDocItem.

ENDMETHOD.

ENDCLASS.

Листинг 4.27. Реализация класса буфера

#### Реализация Local Behavior Handler Class для документа покупки

Теперь давайте приступим к кодированию методов класса локального обработчика поведения для обработки логики операций и действий CRUD для PurchaseDocument. Мы добавим логику только к тем методам, которые нам нужны для нашего примера.

#### Create

В методе *create*, показанном в листинге 4.28, входящие данные (параметры импорта определения создания сущности из определения поведения) зацикливаются и извлекаются соответствующие поля. В этом случае мы также заполним некоторые поля, которые пользователи не поддерживают через пользовательский интерфейс, такие как время создания данных, время последнего изменения данных, создание по имени пользователя, последнее изменение по имени пользователя, начальный статус созданного купленного document и т. д. Как только все необходимые данные будут заполнены и отсортированы соответствующим образом, мы вызовем соответствующий метод класса *zcl\_purchdc\_logic* для сохранения наших данных *PurchaseDocument* в буферной таблице. Метод *create\_purchasedocument* имеет параметр импорта *et\_messages* (который будет заполнен в случае каких-либо исключений или сбоев проверки). Если этот возвращаемый параметр является *INITIAL*, то мы заполним сообщение об успехе в таблице *REPORTED* (которая является параметром экспорта для метода *CREATE*), чтобы сообщения об успехе отображались в приложении. (Вместо заполнения таблицы *REPORTED* на этом этапе мы также можем сделать то же самое с методом *check\_before\_save* локального класса сохранения поведения; разработчик может решить, как и где заполнить эти возвращаемые параметры таблицы в зависимости от сложности приложения).

Точно так же, если таблица *et\_messages* не является *INITIAL*, что-то пошло не так при проверке данных в классе буфера. Поэтому нам нужно заполнить таблицу *FAILED* (как объяснялось ранее, каждая таблица *FAILED, REPORTED* и *MAPPED* имеет специальную таблицу в своем элементе строки для всех наборов сущностей, определенных в определении поведения). Поскольку метод *CREATE* предназначен для сущности *PurchaseDocument*, соответствующей таблицей для *FAILED* будет *FAILEDPurchaseDocument*) с соответствующим сообщением в таблице *et\_messages*. Заполнение таблицы *FAILED* также предотвращает запуск метода *save* в локальном классе сохранения поведения, тем самым гарантируя, что данные не будут сохранены в базе данных.

METHOD create.

TYPES tt\_message TYPE STANDARD TABLE OF symsg.

CLEAR et\_messages.

DATA:

ls\_purchdocument TYPE zpurchdocument,

lt\_purchdocument TYPE zif\_prchdc\_logic=>tt\_purchasedocument,

lv\_purchasedocument TYPE ZPURCHASEDOCUMENTDTEL,

lv\_cid TYPE string.

\* Selecting the highest PurchaseDocument number from the DB table to assign the next \* PurchaseDocument Number

SELECT FROM zpurchdocument

FIELDS MAX( purchasedocument )

INTO @lv\_purchasedocument.

\* Loop at the importing parameter of the method to retrieve the

PurchaseDocument details for creation

LOOP AT entities ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocument\_Create>).

CLEAR ls\_purchdocument.

\* The incoming PurchaseDocument structure is not an 1:1 match with the DB table

\* structure for PurchaseDocument

\* Hence, we need to map the relevant fields of the incoming structure to the

\* compatible DB Structure.

ls\_purchdocument = CORRESPONDING #( zcl\_prchdc\_logic=>get\_instance( )->

map\_purchdoc\_cds\_to\_db( CORRESPONDING #( <fs\_PurchaseDocument\_Create> ) ) ).

lv\_cid = <fs\_PurchaseDocument\_Create>-%cid.

if <fs\_PurchaseDocument\_Create>-PurchaseDocument > lv\_purchasedocument .

lv\_purchasedocument = <fs\_PurchaseDocument\_Create>-PurchaseDocument.

ENDIF.

lv\_purchasedocument = lv\_purchasedocument + 1.

condense lv\_purchasedocument.

ls\_purchdocument-purchasedocument = lv\_purchasedocument.

\* Setting the Time Stamp Field for Created Date Time and Last Changed Date Time

GET TIME STAMP FIELD ls\_purchdocument-crea\_date\_time.

GET TIME STAMP FIELD ls\_purchdocument-lchg\_date\_time.

\* Details of the Created and Last changed User name is also added

ls\_purchdocument-crea\_uname = sy-uname.

ls\_purchdocument-lchg\_uname = sy-uname.

ls\_purchdocument-status = 1.

\* You can make the ImageURL dynamic for your example, For this example.

\* We’re giving a hard-coded value of one of the existing MIME objects in the

\* BSP application source code of this project

ls\_purchdocument-purchasedocumentimageurl = './images/book.jpg'.

APPEND ls\_purchdocument TO lt\_purchdocument.

ENDLOOP.

\* Calling up the relevant method from the ZCL\_PRCHDC\_LOGIC class to store the

PurchaseDocument details in the Buffer Table,

\* This Class method will also do some validations on the incoming Purchase

Document data and it will will its Exporting Parameter

\* et\_messages with the relevant error message

zcl\_prchdc\_logic=>get\_instance( )->create\_purchasedocument(

EXPORTING it\_purchasedocument = lt\_purchdocument

IMPORTING et\_messages = et\_messages ).

IF et\_messages IS INITIAL.

\* If there is no errors returned by the create\_purchasedocument method, then

\* we set the success message to the REPORTED

\* Parameter with new purchaseDocument number to be displayed as a Toast

\* message in the Application

INSERT VALUE #(

purchasedocument = ls\_purchdocument-purchasedocument

%msg = new\_message( id = 'ZPURCHDOC\_EXCEPTIONS' number = '000'

v1 = ls\_purchdocument-purchasedocument

severity = if\_abap\_behv\_message=>severity-success )

%element-purchasedocument = cl\_abap\_behv=>flag\_changed

%cid = lv\_cid )

INTO TABLE reported-PurchaseDocument.

ELSE.

LOOP AT et\_messages INTO DATA(ls\_message).

\* In case of a Failed Scenario, the FAILED Parameter is filled with the

\* relevant PurchaseDocument number, This will stop any further execution of the

\* Class methods

INSERT VALUE #(

%cid = lv\_cid purchasedocument = ls\_purchdocument-purchasedocument )

INTO TABLE failed-purchasedocument.

\* We will also push the relevant error message to REPORTED parameter so that

\* it is displayed in the front-end application as a toast message

INSERT zcl\_prchdc\_logic=>map\_purchasedoc\_message(

iv\_purchasedocument = ls\_purchdocument-purchasedocument

is\_message = ls\_message

is\_messagetype = 'E' )

INTO TABLE reported-purchasedocument.

ENDLOOP.

ENDIF.

ENDMETHOD.

Листинг 4.28. Реализация метода для операции Create

#### Delete

В методе удаления входящий параметр зацикливается, и извлекаются соответствующие значения ключевых полей (в нашем случае документ покупки является единственным ключом). Поскольку удаление происходит на уровне заголовка (документа покупки), нам также придется извлечь все элементы из таблицы элементов документа покупки, как показано в листинге 4.29.

После заполнения соответствующей структуры таблицы для документов покупки и позиций документов покупки мы передадим эти таблицы в метод *delete\_purchasedocument* класса *zcl\_purchdc\_logic*. В результате данные будут переданы классу локальной буферной таблицы для хранения в соответствующей буферной таблице. В этом методе таблица *et\_messages* также будет проверена, чтобы определить, нужно ли заполнять таблицу *REPORTED* или нет.

METHOD delete.

DATA:

ls\_purchdocument TYPE zpurchdocument,

lv\_purchasedocumentItem TYPE ZPURCHASEDOCUMENTDTEL,

ls\_purchdocumentkey TYPE zif\_prchdc\_logic=>ts\_purchdocitem\_key,

lt\_purchdocumentKey TYPE zif\_prchdc\_logic=>tt\_purchasedocumentkey,

lt\_purchdocitemKey TYPE zif\_prchdc\_logic=>tt\_purchdocitem\_key.

CLEAR et\_messages.

LOOP AT keys ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocument\_Delete>).

MOVE-CORRESPONDING <fs\_PurchaseDocument\_Delete> TO ls\_purchdocument.

\* The relevant PurchaseDocument number for deletion is retrieved

APPEND VALUE #( purchasedocument = ls\_purchdocument-purchasedocument )

TO lt\_purchdocumentKey.

ENDLOOP.

IF lt\_purchdocumentKey is NOT INITIAL.

\* If any PurchaseDocument Items exists for this Document, those items will be

\* fetched and passed to the relevant delete method as well.

SELECT \*

FROM zpurchdocitem

into CORRESPONDING FIELDS OF TABLE lt\_purchdocitemKey

FOR ALL ENTRIES IN lt\_purchdocumentKey

where purchasedocument = lt\_purchdocumentKey-purchasedocument.

\* The delete\_purchasedocument method is called by passing the relevant

\* PurchaseDocument and PurchaseDocument Item numbers

zcl\_prchdc\_logic=>get\_instance( )->delete\_purchasedocument(

EXPORTING it\_purchasedocumentkey = lt\_purchdocumentKey

it\_purchdocitemkey = lt\_purchdocitemKey

IMPORTING et\_messages = et\_messages ).

IF et\_messages IS INITIAL.

\* If no errors are returned, then the REPORTED parameter is filled with the relevant

\* success message

APPEND VALUE #(

purchasedocument = ls\_purchdocument-purchasedocument

%msg = new\_message(

id = 'ZPURCHDOC\_EXCEPTIONS'

number = '004'

v1 = ls\_purchdocument-purchasedocument

severity = if\_abap\_behv\_message=>severity-success )

%element-purchasedocument = cl\_abap\_behv=>flag\_changed )

TO reported-PurchaseDocument.

ENDIF.

ENDIF.

ENDMETHOD.

Листинг 4.29. Реализация метода для операции удаления

#### Update

В методе обновления при зацикливании параметров импорта нам нужно учитывать поле *%control* в таблице параметров импорта, чтобы определить, какие поля объекта были изменены пользователем. Произошло ли изменение, представлено значением в структуре *%control* по сравнению с соответствующими именами свойств объекта.

Например, если поле *%control-status* равно «00», это поле не изменится; если значение было «01», то значение поля состояния было изменено. Чтобы упростить эту проверку, мы можем воспользоваться логической функцией *xsdbool* (которая возвращает «X» в случае истинности и пробел в случае ложности), как показано в листинге 4.30. Соответствующие значения этой проверки сохраняются во внутренней таблице *lt\_purchasedocumentControl*, которая будет передана в метод *update\_purchasedocument* класса *zcl\_purchdc\_logic* для сохранения данных в буферной таблице. В методе *update\_purchasedocument* таблица *lt\_purchasedocumentControl* будет использоваться в качестве флага для создания окончательной структуры таблицы (в которой соответствующие значения полей будут обновлены новыми значениями) для данных перед их сохранением в буферной таблице. (Вы можете проверить логику исходного кода, прилагаемого к этой книге, на веб-сайте SAP PRESS по адресу [www.sap-press.com/4988](http://www.sap-press.com/4988)).

METHOD update.

DATA:

ls\_purchdocument TYPE zpurchdocument,

ls\_purchdocumentControl TYPE zif\_prchdc\_logic=>ts\_purchdocumentControl,

lt\_purchasedocument TYPE zif\_prchdc\_logic=>tt\_purchasedocument,

lt\_purchasedocumentControl TYPE zif\_prchdc\_logic=>tt\_purchdocumentcontrol.

\* The incoming structure containing the PurchaseDocument Details to be updated is

\* looped

LOOP AT entities ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocument\_Create>).

\* Here the relevant fields of the incoming Structure is mapped to the

\* compatible structure of the PurchaseDocument DB table.

ls\_purchdocument = CORRESPONDING #(

zcl\_prchdc\_logic=>get\_instance( )->map\_purchdoc\_cds\_to\_db(

CORRESPONDING #( <fs\_PurchaseDocument\_Create> ) ) ).

\* As mentioned in chapter 4, the incoming Structure also has some column

\* %CONTROL which contains the

\* flag against all the fields of the PurchaseDocuments that were modified by the user

\* and hence we need to store them in a separate structure so that it can be used as

\* a reference later to determine which fields from the DB needs to be updated

ls\_purchdocumentControl-action = 'U'.

ls\_purchdocumentControl-purchasedocument = ls\_purchdocument-purchasedocument.

ls\_purchdocumentControl-description =

xsdbool( <fs\_PurchaseDocument\_Create>-%control-Description =

cl\_abap\_behv=>flag\_changed ).

ls\_purchdocumentControl-status =

xsdbool( <fs\_PurchaseDocument\_Create>-%control-Status =

cl\_abap behv=>flag\_changed ).

ls\_purchdocumentControl-priority =

xsdbool( <fs\_PurchaseDocument\_Create>-%control-Priority =

cl\_abap\_behv=>flag\_changed ).

ls\_purchdocumentControl-purchasingorganization =

xsdbool( <fs\_PurchaseDocument\_Create>-%control-PurchasingOrganization =

cl\_abap\_behv=>flag\_changed ).

ls\_purchdocumentControl-purchasedocumentimageurl =

xsdbool( <fs\_PurchaseDocument\_Create>-%control-PurchaseDocumentImageURL=

cl\_abap\_behv=>flag\_changed ).

ls\_purchdocumentControl-crea\_date\_time =

xsdbool( <fs\_PurchaseDocument\_Create>-%control-crea\_date\_time =

cl\_abap\_behv=>flag\_changed ).

ls\_purchdocumentControl-crea\_uname =

xsdbool( <fs\_PurchaseDocument\_Create>-%control-crea\_uname =

cl\_abap\_behv=>flag\_changed ).

ls\_purchdocumentControl-lchg\_date\_time = 'X'.

ls\_purchdocumentControl-lchg\_uname = 'X'.

\* The last change date time needs to be updated as well

GET TIME STAMP FIELD ls\_purchdocument-lchg\_date\_time.

\* Also the details about the user who made the changes needs to be stored

ls\_purchdocument-lchg\_uname = sy-uname.

APPEND ls\_purchdocument TO lt\_purchasedocument.

APPEND ls\_purchdocumentControl TO lt\_purchasedocumentControl.

ENDLOOP.

IF lt\_purchasedocument IS NOT INITIAL.

\* The method update\_purchasedocument is called to update the relevant

\* PurchaseDocument Buffer table

\* which will be later referred to update the relevant DB table

zcl\_prchdc\_logic=>get\_instance( )->update\_purchasedocument(

EXPORTING it\_purchasedocument = lt\_purchasedocument

it\_purchdocumentcontrol = lt\_purchasedocumentcontrol

IMPORTING et\_messages = et\_messages ).

IF et\_messages IS INITIAL.

\* If no error messages are returned from the update\_purchasedocument method,

\* then the relevant success message is pushed to the REPORTED parameter

APPEND VALUE #(

purchasedocument = ls\_purchdocument-purchasedocument

%msg = new\_message(

id = 'ZPURCHDOC\_EXCEPTIONS'

number = '011'

v1 = ls\_purchdocument-purchasedocument

severity = if\_abap\_behv\_message=>severity-success )

%element-purchasedocument = cl\_abap\_behv=>flag\_changed )

TO reported-PurchaseDocument.

ENDIF.

ENDIF.

ENDMETHOD.

Листинг 4.30. Реализация метода для операции обновления

#### Read (Read by Key)

В методе чтения будут возвращены данные из буфера. Если буферные таблицы пусты, то данные будут считаны из базы данных. Запросы на чтение автоматически передаются структурой SADL в базу данных.

Однако логика метода чтения обычно реализуется, когда требуется/включена обработка ETag. Когда данные необходимо изменить, платформа SADL блокирует данные, а затем запускается метод чтения для получения текущего значения ETag. Когда данные совпадают, и только тогда будут вызываться соответствующие методы модификации. Но в нашем примере мы не используем функцию ETag, и данные, которые мы извлекаем, не являются достаточно важными, чтобы их можно было сравнить с записью в буферной таблице. Вы можете добавить логику для чтения из буферных таблиц, которую вы можете создать для метода чтения точно так же, как мы создали для операций CRUD. Таким образом, этот метод остается без изменений.

#### cba\_PurchaseDocumentItem (Create by Association)

В методе Create by Association данные из параметра импорта зацикливаются, чтобы получить значение документа покупки ключевого поля, а затем структуру *%target* (как описано ранее, поле *%target* в импортируемом параметре метода Create by Association будет содержать структуру с данными для child/association entity) зацикливается, чтобы получить соответствующие данные для создания элементов документа покупки. Таблица элементов *PurchaseDocument* передается в метод *create\_purchasedocitem* класса *zcl\_prchdc\_logic*.

METHOD cba\_PURCHASEDOCUMENTITEM.

DATA:

ls\_purchdocument TYPE zpurchdocument,

ls\_purchdocumentItem TYPE zpurchdocitem,

lt\_purchdocumentItem TYPE zif\_prchdc\_logic=>tt\_purchdocumentitem,

lv\_purchdocitem TYPE zpurchasedocumentdtel.

\* In this Create By Association method for creating the PurchaseDocument Item

\* in relation with the parent PurchaseDocument,

\* The highest value of the current PurchaseDocumentItem for the respective

\* PurchaseDocument is retrieved to determine the next number

SELECT FROM zpurchdocitem

FIELDS MAX( purchasedocumentitem )

INTO @lv\_ purchdocitem.

\* The incoming parameter is looped to retrieve the PurchaseDocumentItem details

LOOP AT entities\_cba ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocument>).

ls\_purchdocument-purchasedocument = <fs\_PurchaseDocument>-PurchaseDocument.

\* As explained in Chapter 4, the incoming parameter also has a field %TARGET

\* which contains the Parent View Key fields,

\* We need these key field values to create the Items in the PurchaseDocumentItem DB

\* table with the correct PurchaseDocument Number

LOOP AT <fs\_PurchaseDocument>-%target

ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocumentItem>).

MOVE-CORRESPONDING <fs\_PurchaseDocumentItem> TO ls\_purchdocumentItem.

if <fs\_PurchaseDocumentItem>-Purchasedocumentitem > lv\_purchdocitem .

lv\_purchdocitem = <fs\_PurchaseDocumentItem>-Purchasedocumentitem.

ENDIF.

lv\_purchdocitem = lv\_purchdocitem + 1.

condense lv\_purchdocitem.

ls\_purchdocumentItem-purchasedocumentitem = lv\_purchdocitem.

ls\_purchdocumentItem-purchasedocument = ls purchdocument-purchasedocument.

ls\_purchdocumentItem-lchg\_uname = sy-uname.

ls\_purchdocumentItem-crea\_uname = sy-uname.

\* You can make the ImageURL dynamic for your example, For this example.

\* We are giving a hard-coded value of one of the existing MIME objects in the

\* BSP application source code of this project

ls\_purchdocumentItem-purchasedocumentitemimageurl = './images/book.jpg'.

\* Setting the Time Stamp Field for Created Date Time and Last Changed Date Time

GET TIME STAMP FIELD ls\_purchdocumentItem-lchg\_date\_time.

GET TIME STAMP FIELD ls\_purchdocumentItem-crea\_date\_time.

APPEND ls\_purchdocumentItem to lt\_purchdocumentItem.

ENDLOOP.

ENDLOOP.

IF lt\_purchdocumentItem IS NOT INITIAL.

\* The create\_purchasedocitem method of the ZCL\_PRCHDC\_LOGIC is called to store the

\* relevant PurchaseDocument Items in the buffer table

zcl\_prchdc\_logic=>get\_instance( )->create\_purchasedocitem(

EXPORTING it\_purchasedocItem = lt\_purchdocumentItem

IMPORTING et\_messages = et\_messages ).

ENDIF.

ENDMETHOD.

Листинг 4.31. Реализация метода Create by Association для создания позиции документа покупки на основе ключа из заголовка документа покупки

#### Approve\_Order (Action)

В методе настраиваемого действия Approve\_Order, как показано в листинге 4.32, мы решили обновить таблицу базы данных напрямую, а не передавать изменения в буферную таблицу, что работает, но, по общему признанию, немного лениво. Вы также всегда можете передать эти значения в буферную таблицу и обновить позже в методе сохранения в локальном классе сохранения поведения. Передача значений действия в буферную таблицу зависит от варианта ее использования. В идеале действие должно быть немедленным, например, подтверждение заказа или бронирование билета. В таких случаях передача действия в буфер может быть не лучшим подходом.

METHOD Approve\_Order.

DATA ls\_purchdocument TYPE zpurchdocument.

CLEAR result.

\* In the custom Action method for Approving a PurchaseDocument,

\* The incoming parameter is looped and the relevant PurchaseDocument status is

\* set as approved and details of the Action is store in the RESULT parameter of

\* the method

LOOP AT keys ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocument>).

UPDATE zpurchdocument SET status = 2

WHERE purchasedocument = <fs\_PurchaseDocument>-PurchaseDocument.

if sy-subrc eq 0.

APPEND VALUE #(

purchasedocument = <fs\_PurchaseDocument>-PurchaseDocument

%param-purchasedocument = <fs\_PurchaseDocument>-PurchaseDocument

%param-status = '2')

TO result.

endif.

ENDLOOP.

\* The relevant Success Message is mapped to the REPORTED parameter of the method

APPEND VALUE #( purchasedocument = ls\_purchdocument-purchasedocument

%msg = new\_message(

id = 'ZPURCHDOC\_EXCEPTIONS'

number = '002'

v1 = <fs\_PurchaseDocument>-PurchaseDocument

severity = if\_abap\_behv\_message=>severity-success )

%element-purchasedocument = cl\_abap\_behv=>flag\_changed )

TO reported-PurchaseDocument.

ENDMETHOD.

Листинг 4.32. Реализация метода для действия Approve\_Order

#### Reject\_Order (Action)

Кнопка действия Reject\_Order также имеет логику, аналогичную методу Approve\_Order, как показано в листинге 4.33. Параметр импорта зацикливается, и соответствующий статус документа покупки обновляется. Сообщение об успешном завершении будет отправлено обратно в пользовательский интерфейс после заполнения таблицы параметров экспорта *REPORTED*.

METHOD Reject\_Order.

DATA ls\_purchdocument TYPE zpurchdocument.

\* In the custom Action method for Rejecting a PurchaseDocument,

\* The incoming parameter is looped and the relevant PurchaseDocument status is

\* set as rejected and details of the Action is store in the RESULT parameter of

\* the method

LOOP AT keys ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocument>).

UPDATE zpurchdocument SET status = 3

WHERE purchasedocument = <fs\_PurchaseDocument>-PurchaseDocument.

ENDLOOP.

\* The relevant Success Message is mapped to the REPORTED parameter of the method

APPEND VALUE #(

purchasedocument = ls\_purchdocument-purchasedocument

%msg = new\_message( id = 'ZPURCHDOC\_EXCEPTIONS'

number = '003'

v1 = <fs\_PurchaseDocument>-PurchaseDocument

severity = if\_abap\_behv\_message=>severity-success )

%element-purchasedocument = cl\_abap\_behv=>flag\_changed )

TO reported-PurchaseDocument.

ENDMETHOD.

Листинг 4.33. Реализация метода для действия Reject\_Order

#### Реализация методов Local Behavior Handler Class для позиции документа покупки

Теперь давайте посмотрим на кодовую часть методов нашего локального класса обработчика поведения для *PurchaseDocumentItem*.

#### Delete

Логика метода удаления в элементе документа покупки аналогична операции удаления объекта *PurchaseDocument*, как показано в листинге 4.34. Мы перейдем к параметру импорта, получим значение ключевого поля и вызовем метод *delete\_PurchaseDocItem* класса *zcl\_purchdoc\_logic*. Если возвращаемый параметр *et\_messages* для *delete\_PurchaseDocItem* является *INITIAL*, мы сопоставляем сообщение об успешном завершении с параметром экспорта *REPORTED* операции удаления.

METHOD delete.

DATA: ls\_purchdocItem TYPE zpurchdocitem,

lt\_purchdocItemKey TYPE zif\_prchdc\_logic=>tt\_purchdocitem\_key.

CLEAR et\_messages.

\* The relevant PurchaseDocumentItem number for deletion is retrieved

LOOP AT keys ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocItem\_Delete>).

MOVE-CORRESPONDING <fs\_PurchaseDocItem\_Delete> TO ls\_purchdocItem.

APPEND VALUE #( purchasedocument = ls\_purchdocItem-purchasedocument purchasedocumentItem = ls\_purchdocItem-purchasedocumentitem ) TO lt\_

purchdocItemKey.

ENDLOOP.

IF lt\_purchdocItemKey IS NOT INITIAL.

\* The delete\_purchasedocumentItem method is called by passing the relevant

PurchaseDocument Item numbers

zcl\_prchdc\_logic=>get\_instance( )->delete\_PurchaseDocItem( EXPORTING it\_

purchasedocitemkey = lt\_purchdocItemKey

IMPORTING et\_

messages = et\_messages ).

IF et\_messages IS INITIAL.

\* If no errors are returned by the delete\_PurchaseDocItem method, then the relevant Success Message is mapped to the REPORTED parameter of the method

APPEND VALUE #( purchasedocumentItem = ls\_purchdocItem-purchasedocumentitem

%msg = new\_message( id = 'ZPURCHDOC\_EXCEPTIONS' number = '015' v1 = ls\_purchdocItem-purchasedocument severity = if\_abap\_behv\_message=>

severity-success )

%element-purchasedocument = cl\_abap\_behv=>flag\_changed )

TO reported-purchasedocumentitem.

ENDIF.

ENDIF.

ENDMETHOD.

Листинг 4.34. Реализация метода для операции удаления

#### Update

Операция обновления также имеет аналогичную логику операции обновления для объекта *PurchaseDocument*. Как показано в листинге 4.35, мы зациклимся на импортируемой таблице метода *Update*. В листинге 4.35 видно, что цикл идет по параметру с именем *entity*. Этот параметр импорта содержит сведения о *PurchaseDocumentItem*, и мы в цикле определим данные, которые необходимо обновить.

Затем мы передаем структуру методу *map\_purchdocitem\_cds\_to\_db*, чтобы получить структуру, подобную таблице базы данных элемента документа покупки. Мы также зациклим таблицу *%control*, чтобы получить флаги и определить, какие свойства объекта были изменены пользователем. Эта структура в параметре импорта содержит сведения о том, какие поля в *PurchaseDocumentItem* были обновлены пользователем. Затем мы передадим сопоставленную таблицу (*lt\_purchasedocumentItem*) и таблицу, содержащую поля, полученные из поля *%control* (*lt\_purchasedocItemControl*), в метод *update\_purchasedocitem* класса *zcl\_prchdc\_logic*.

METHOD update.

DATA: ls\_purchdocumentItem TYPE zpurchdocitem,

ls\_purchdocumentItemControl TYPE zif\_prchdc\_logic=>ts\_

purchdocumentitemControl,

lt\_purchasedocumentItem TYPE zif\_prchdc\_logic=>tt\_

purchdocumentitem,

lt\_purchasedoctItemControl TYPE zif\_prchdc\_logic=>tt\_

purchdocumentitemcontrol.

\* The incoming structure containing the PurchaseDocumentItem Details to be

updated is looped

LOOP AT entities ASSIGNING FIELD-SYMBOL(<fs\_PurchaseDocItem\_Create>).

\* Here the relevant fields of the incoming Structure is mapped to the

compatible structure of the PurchaseDocumentItem DB table.

ls\_purchdocumentItem = CORRESPONDING #( zcl\_prchdc\_logic=>get\_instance(

)->map\_purchdocitem\_cds\_to\_db( CORRESPONDING #( <fs\_PurchaseDocItem\_Create> )

) ).

\* As mentioned in chapter 4, the incoming Structure also has a column %CONTROL

which contains the

\* flag against all the fields of the PurchaseDocumentItems that were modified

by the user and hence we need to store them in a separate structure so that it

can be used as

\* a reference later to determine which fields from the DB needs to be updated

ls\_purchdocumentItemControl-action = 'U'.

ls\_purchdocumentItemControl-purchasedocument = ls\_

purchdocumentItem-purchasedocument.

ls\_purchdocumentItemControl-purchasedocumentitem = ls\_

purchdocumentItem-purchasedocumentitem.

ls\_purchdocumentItemControl-crea\_date\_time = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-crea\_date\_time = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-crea\_uname = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-crea\_uname = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-currency = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-Currency = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-description = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-Description = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-lchg\_date\_time = 'X'.

ls\_purchdocumentItemControl-lchg\_uname = 'X'.

ls\_purchdocumentItemControl-price = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-Price = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-purchasedocumentitemimageurl = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-PurchaseDocumentItemImageURL = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-quantity = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-Quantity = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-quantityunit = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-QuantityUnit = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-vendor = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-Vendor = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-vendortype = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-VendorType = cl\_abap\_

behv=>flag\_changed ).

ls\_purchdocumentItemControl-lchg\_uname = xsdbool( <fs\_PurchaseDocItem\_Create>-%control-lchg\_uname = cl\_abap\_

behv=>flag\_changed ).

\* Time Stamp Field for Last Changed Date Time and last changed user name needs

to be updated

GET TIME STAMP FIELD ls\_purchdocumentItem-lchg\_date\_time.

ls\_purchdocumentItem-lchg\_uname = sy-uname.

APPEND ls\_purchdocumentItem TO lt\_purchasedocumentItem.

APPEND ls\_purchdocumentItemControl TO lt\_purchasedoctItemControl.

ENDLOOP.

IF lt\_purchasedoctItemControl IS NOT INITIAL.

\* The method update\_purchasedocument is called to update the relevant

PurchaseDocument Buffer table

\* which will be later referred to update the relevant DB table

zcl\_prchdc\_logic=>get\_instance( )->update\_purchasedocitem( EXPORTING it\_purchasedocitem = lt\_purchasedocumentItem

it\_purchdocitemcontrol = lt\_purchasedoctItemControl

IMPORTING

et\_messages = et\_messages ).

IF et\_messages IS INITIAL.

\* If no error messages are returned from the update\_purchasedocument method,

then the relevant success message is pushed to the REPORTED parameter

APPEND VALUE #( purchasedocumentItem = ls\_purchdocumentItem-

purchasedocumentitem

%msg = new\_message( id = 'ZPURCHDOC\_EXCEPTIONS' number = '001'

v1 = ls\_purchdocumentItem-purchasedocumentitem severity = if\_abap\_behv\_

message=>severity-success )

%element-purchasedocument = cl\_abap\_behv=>flag\_changed ) TO reported-purchasedocumentitem.

ENDIF.

ENDIF.

ENDMETHOD.

Листинг 4.35. Реализация метода для операции обновления

#### Реализация методов класса Local Behavior Saver

Теперь давайте посмотрим на логику реализации локального класса сохранения поведения, как показано в листинге 4.36. В нашем примере мы не обрабатываем *finalize* (поскольку мы не используем функцию поздней нумерации, в нашем приложении нет особой необходимости в методе *finalize*) или *check\_before\_save* (поскольку наше примерное приложение не нуждается в дополнительной проверке данных перед сохранением). Это может отличаться для других сценариев. Например, если вы используете систему бронирования, которая может потребовать выполнения других операций перед окончательным бронированием, эта окончательная проверка поможет вам либо отменить изменения, либо зафиксировать изменения. Кроме того, в нашем примере мы уже обрабатывали сообщения об исключениях во время сохранения данных в класс буфера. Чтобы упростить реализацию, вы можете переместить логику обработки сообщений об исключениях в метод *check\_before\_save*, как показано в листинге 4.36.

CLASS lsc\_Z\_I\_PurchaseDocument\_U IMPLEMENTATION.

\* These methods are executed in the below listed Sequence, except for the

cleanup method,

\* which is called within any of the other three methods in case the execution

logic is aborted

METHOD check\_before\_save.

ENDMETHOD.

METHOD finalize.

ENDMETHOD.

METHOD save.

\* The final method where all the data from the relevant buffer tables are used

to either created, updated or deleted

\* the records in the relevant DB tables based on the execution logic

zcl\_prchdc\_logic=>get\_instance( )->save( ).

ENDMETHOD.

METHOD cleanup.

\* This method is used to cleanup all the buffer tables in case the Execution

flow is terminated

zcl\_prchdc\_logic=>get\_instance( )->initialize( ).

ENDMETHOD.

ENDCLASS.

Листинг 4.36. Реализация класса Behavior Saver

#### Предварительный просмотр приложения SAP Fiori Elements в Service Binding

Теперь, если вы откроете привязку службы и щелкните предварительный просмотр приложения Open Fiori Elements для объекта PurchaseDocument после его повторной активации, как показано на рисунке 4.26. Вы увидите предварительный просмотр приложения с функциями «Создать» и «Удалить» в верхней части таблицы, как показано на рис. 4.50. Но даже после указания настраиваемых действий в определении поведения эти действия по-прежнему не будут видны, поскольку отсутствуют соответствующие аннотации пользовательского интерфейса для представлений (которые мы добавим позже в разделе 4.4).

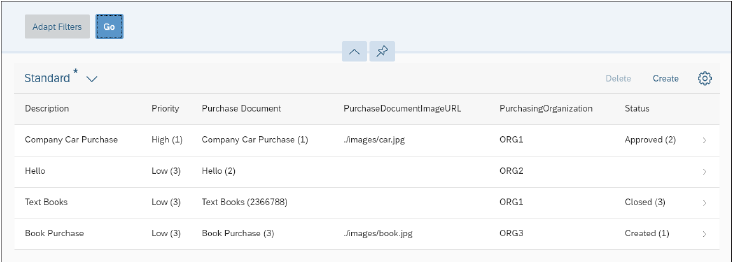


Рис. 4.50 Предварительный просмотр приложения с включенными Transactional Actions

### 4.2.10 Создание Projection Views (Представления проекции)

Как упоминалось в начале этой главы и как показано на рис. 4.2, теперь нам нужно создать представления проекции (Projection Views) поверх представлений типа «\_U», так чтобы эти базовые представления можно было использовать повторно. Используя этот подход, добавляя все наши аннотации пользовательского интерфейса непосредственно в представлении «\_U», любая другая команда или человек, которые хотят использовать эти представления, в конечном итоге получат пользовательский интерфейс, который мы настроили в соответствии с нашими требованиями. Таким образом, в идеальной среде разработки каждое приложение SAP Fiori должно иметь свои собственные конечные consumption view, чтобы мы могли настроить их для конкретных требований нашего приложения, не вмешиваясь в базовый бизнес-объект. По этой же причине мы создали два последних consumption view CDS C\_PurchaseDocumentLRP и C\_PurchaseDocumentItemLRP ранее в Разделе 4.1.4.

Но мы не сможем получать данные из наших базовых представлений «\_U», не изменив сначала consumption view. Это ограничение возникает из-за того, что модель программирования ABAP RESTful не позволяет обычным consumption view извлекать из представлений CDS, связанных с моделью программирования ABAP RESTful. Таким образом, нам нужно преобразовать наши текущие consumption view в projection view, как показано в листинге 4.37. *Представления проекции (Projection Views)* — это особый тип представления, разработанный для модели программирования ABAP RESTful, который может повторно использовать представления транзакций, созданные с помощью модели программирования ABAP RESTful. Представления проекции не имеют связанных с ними таблиц SQL, потому что они являются проекциями фактических базовых представлений CDS. Проекционные представления имеют некоторые ограничения; например, нельзя использовать операторы case. Следовательно, нам нужно перенести эти операторы в наши представления «\_U».

Представление проекции можно определить с помощью синтаксиса: определите сущность корневого представления *<view\_name> as projection on <underlying\_transactional\_view\_name>*, как показано в листинге 4.37 и листинге 4.38.

@EndUserText.label: 'Purchase Document'

@AccessControl.authorizationCheck: #NOT\_REQUIRED

@Search.searchable: true

@Metadata.allowExtensions:true

@VDM.viewType: #CONSUMPTION

**define root view entity** Z\_C\_PurchaseDocumentLrp

**as projection on** Z\_I\_PurchaseDocument\_U

{

@EndUserText.label: 'Purchase Document'

@Consumption.semanticObject: 'PurchasingDocument'

**key** PurchaseDocument,

@EndUserText.label: 'Overall Price'

OverallPrice,

@EndUserText.label: 'Approval Required'

@ObjectModel.foreignKey.association: '\_IsApprovalRequired'

@Consumption.valueHelpDefinition:

[{entity:{name:'I\_Indicator' , element: 'IndicatorValue'}}]

IsApprovalRequired,

OverallPriceCriticality,

@EndUserText.label: 'Status'

@Consumption.valueHelpDefinition:

[{entity:{name:'Z\_C\_StatusVH' , element: 'Status'}}]

Status,

@EndUserText.label: 'Priority'

@Consumption.valueHelpDefinition:

[{entity:{name:'Z\_C\_PriorityVH' , element: 'Priority'}}]

Priority,

@Search.defaultSearchElement : true

@Search.fuzzinessThreshold : 0.8

@Semantics.text: true

@EndUserText.label: 'Description'

Description,

@EndUserText.label: 'Purchasing Organization'

@Consumption.valueHelpDefinition:

[{entity:{name:'Z\_I\_PurchasingOrganization' ,

element: 'PurchasingOrganization'}}]

PurchasingOrganization,

@EndUserText.label: 'Currency'

Currency,

@EndUserText.label: 'Created at'

@Consumption.filter.hidden: true

crea\_date\_time,

@EndUserText.label: 'Created by'

crea\_uname,

@EndUserText.label: 'Last changed at'

@Consumption.filter.hidden: true

lchg\_date\_time,

@EndUserText.label: 'Last changed by'

lchg\_uname,

@EndUserText.label: 'Image'

@Consumption.filter.hidden: true

PurchaseDocumentImageURL,

/\* Associations \*/

\_PurchaseDocumentItem :

**redirected to composition child** Z\_C\_PurchaseDocumentItemLrp,

\_Currency,

\_IsApprovalRequired,

\_Priority,

\_Status,

\_PurchasingOrganization

}

Листинг 4.37. Представление проекции Z\_C\_PurchaseDocumentLRP

Представление проекции Z\_C\_PurchaseDocumentItemLRP также необходимо изменить, чтобы оно могло получать данные из представления Z\_I\_PurchaseDocumentItem\_U, как показано в листинге 4.38. В следующем разделе мы добавим поле виртуального элемента в это представление CDS для расчета рейтинга поставщика. Добавление виртуального поля в представление проекции также немного отличается от обычного вида CDS.

@EndUserText.label: 'Purchase Document'

@AccessControl.authorizationCheck: #NOT\_REQUIRED

@Search.searchable: true

@Metadata.allowExtensions:true

@VDM.viewType: #CONSUMPTION

**define view entity** Z\_C\_PurchaseDocumentItemLrp

**as projection on** Z\_I\_PurchaseDocumentItem\_U

{

@EndUserText.label: 'Purchase Document Item'

@Search: {defaultSearchElement: true, ranking: #HIGH, fuzzinessThreshold: 0.8}

**key** PurchaseDocumentItem,

@ObjectModel.foreignKey.association: '\_PurchaseDocument'

@EndUserText.label: 'Purchase Document'

**key** PurchaseDocument,

@EndUserText.label: 'Price'

Price,

@EndUserText.label: 'Quantity'

Quantity,

@EndUserText.label: 'Overall Item Price'

OverallItemPrice,

@Search: {defaultSearchElement: true, ranking: #HIGH, fuzzinessThreshold: 0.8}

@EndUserText.label: 'Vendor Name'

Vendor,

@EndUserText.label: 'Vendor Type'

@Consumption.valueHelpDefinition:

[{entity:{name:'Z\_C\_VendorTypeVH' , element: 'VendorType'}}]

VendorType,

@Search: {defaultSearchElement: true, ranking: #HIGH, fuzzinessThreshold: 0.8}

@EndUserText.label: 'Item Description'

Description,

@Consumption.valueHelpDefinition:

[{entity:{name:'I\_Currency' , element: 'Currency'}}]

Currency,

@Consumption.valueHelpDefinition:

[{entity:{name:'I\_UnitOfMeasure' , element: 'UnitOfMeasure'}}]

QuantityUnit,

@EndUserText.label: 'Image'

@Consumption.filter.hidden: true

PurchaseDocumentItemImageURL,

@EndUserText.label: 'Created at'

crea\_date\_time,

@EndUserText.label: 'Created by'

crea\_uname,

@EndUserText.label: 'Last changed at'

lchg\_date\_time,

@EndUserText.label: 'Last changed by'

lchg\_uname,

/\* Associations \*/

\_PurchaseDocument : **redirected to parent** Z\_C\_PurchaseDocumentLrp,

\_QuantityUnitOfMeasure,

\_VendorType

}

Листинг 4.38. Проекционное представление Z\_C\_PurchaseDocumentItemLRP

После преобразования наших представлений потребления в представления проекций окончательный VDM нашего приложения показан на рис. 4.51. Составной слой (composite) потребляется представлениями проекций, и, следовательно, транзакционные функции из составных представлений теперь передаются этим многоразовым слоям представления потребления.

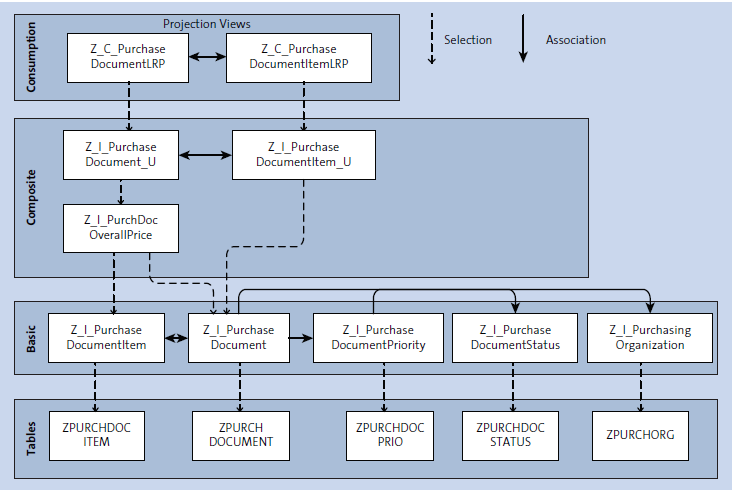


Рисунок 4.51 Окончательный VDM приложения

### 4.2.11 Создание (Projection Behavior Definition) проекции определения поведения

То же правило для создания представления проекции для использования из бизнес-объектов с поддержкой модели программирования ABAP RESTful применимо и к определениям поведения.

Поскольку для определения поведения разрешено только одно представление, нашему новому Projection View также требуется собственная проекция определения поведения. Шаги для создания проекции определения поведения такие же, как и для определения нормального поведения.

Единственная разница заключается в том, чтобы указать имя проекционного вида в поле Root View name в окне мастера создания определения поведения. Проекция определения поведения будет выглядеть так, как показано в листинге 4.39. Если вы проверите код, первая строка показывает, что это проекция, а операции CRUD и действия определяются с использованием ключевого слова *use* перед фактическими именами операций/действий.

projection;

define behavior for Z\_C\_PurchaseDocumentLrp //alias <alias\_name>

{

use create;

use update;

use delete;

use action Approve\_Order as Approve;

use action Reject\_Order as Reject;

use association \_PURCHASEDOCUMENTITEM { create; }

}

define behavior for Z\_C\_PurchaseDocumentItemLrp //alias <alias\_name>

{

use create;

use update;

use delete;

}

Листинг 4.39. Определение проекции поведения

## 4.3 Виртуальные элементы в Core Data Services

В разделе 4.1.4 вы узнали, что выразительный DDL данных CDS позволяет нам передавать вычисления в SAP HANA, что сокращает объем вычислений на сервере ABAP и способствует выталкиванию кода и парадигме Code-to-Data для использования базы данных SAP HANA и её возможности. Однако в некоторых случаях вам все равно может потребоваться вызвать некоторый код ABAP для вычисления значений, которые не являются частью постоянной модели, например, устаревшей логики ABAP, которую нельзя легко перенести в представления CDS. В таких случаях в игру могут вступить виртуальные элементы, которые определяются на уровне представления потребления CDS, но рассчитываются с использованием выходов кода ABAP.

Аннотации, необходимые для определения поля CDS как виртуального элемента, показаны в листинге 4.40. На уровне CDS поле виртуального элемента — это просто пустое поле. Однако SADL будет анализировать аннотации и, когда представление CDS запрашивается через соответствующий объект OData, делегирует вычисление данных выходному классу ABAP (***@ObjectModel.virtualElementCalculatedBy***).

Если вы также можете фильтровать и сортировать предоставленные сущностью OData данные по полю виртуального элемента, необходимо реализовать фильтр (***@ObjectModel.filter.transformedBy***) и сортировку (***@ObjectModel.sort.transformedBy***) аннотации и выходы.

@ObjectModel.virtualElementCalculatedBy: 'ABAP:<ABAP\_class>'

@ObjectModel.filter.transformedBy: 'ABAP:<ABAP\_class>'

@ObjectModel.sort.transformedBy: 'ABAP:<ABAP\_class>'

Virtual <field\_name> : <datatype>

Листинг 4.40. Объявление поля в качестве виртуального элемента в списке выбора полей CDS

### 4.3.2 Реализация кода ABAP для заполнения виртуального элемента

Для расчета, сортировки и фильтрации виртуальных элементов соответствующие классы выхода ABAP должны реализовывать интерфейсы и методы, перечисленные в таблице 4.2.

Для вычисления виртуального элемента метод *GET\_CALCULATION\_INFO* вызывается перед выполнением фактической базы данных SELECT. Предполагается, что этот метод возвращает все поля, необходимые для расчета поля виртуального элемента. Затем SADL позаботится о том, чтобы эти поля также были выбраны из базы данных с помощью своего универсального механизма запросов. В методе *CALCULATE*, который вызывается после SELECT базы данных, должен быть реализован расчет значения поля виртуального элемента, и он может использовать ранее указанные и выбранные поля. Соответствующие методы сопоставления выходов фильтра (MAP\_ATOM) и сортировки (MAP\_ELEMENT) вызываются до фактического выполнения SELECT базы данных. В этом случае условия сортировки или фильтрации, определенные для виртуального элемента, должны быть сопоставлены с другими полями, доступными в постоянной таблице базы данных, и могут быть включены в оператор SELECT базы данных, который генерирует механизм запросов SADL.

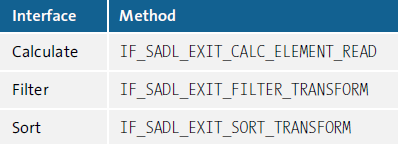


Таблица 4.2 Интерфейсы, которые необходимо реализовать для различных выходов виртуальных элементов

В нашем представлении CDS мы будем использовать эту функцию виртуального элемента для определения рейтинга поставщиков на лету. В предыдущей модели программирования ABAP для SAP Fiori эти виртуальные поля были просто частью аннотаций, но теперь они интегрированы в язык ABAP. Виртуальные поля можно добавлять в представление проекции, как показано в листинге 4.41. Логика расчета поставщика реализована в классе *ZCL\_VENDOR\_RATING\_CALC\_EXIT*.

@ObjectModel.virtualElementCalculatedBy: 'ABAP:ZCL\_VENDOR\_RATING\_CALC\_EXIT'

virtual VendorRating :abap.int1 ( 0 ),

Листинг 4.41. Реализация виртуального элемента для рейтинга поставщиков

## 4.4 Добавление аннотаций пользовательского интерфейса к Projection Views (Представлениям проекции)

Пользовательский интерфейс нашего приложения будет основан на List Report SAP Fiori и шаблонах страниц объектов. В отличие от приложений свободного стиля SAP Fiori, макет приложений элементов SAP Fiori относительно фиксирован, а элементы управления пользовательского интерфейса нельзя размещать по желанию где-либо на странице. Однако при использовании приложений на основе шаблонов элементов SAP Fiori необходимо указать, какие поля будут отображаться в каких предопределенных позициях шаблона с помощью аннотаций *CDS @UI*. В следующих подразделах мы покажем вам, как указать аннотации пользовательского интерфейса, необходимые для нашего приложения. Кроме того, мы также представим некоторые аннотации, не относящиеся к пользовательскому интерфейсу, которые также повлияют на макет и функциональность пользовательского интерфейса, например, аннотации *@Search*.

### 4.4.1 Создание файла расширения метаданных

Вы можете добавить аннотации пользовательского интерфейса непосредственно в файл CDS DDL или создать файл расширения метаданных. Если вы создаете много аннотаций пользовательского интерфейса, что обычно имеет место при использовании приложений на основе шаблонов элементов SAP Fiori, мы рекомендуем переместить эти аннотации в отдельный файл, чтобы не загромождать основное представление CDS множеством аннотаций пользовательского интерфейса. Вы можете создать файл расширения метаданных, выполнив следующие действия:

1. Откройте перспективу ABAP в инструментах разработки ABAP в Eclipse.

2. В проекте ABAP выберите узел пакета, в котором вы хотите создать таблицу, с помощью представления Project Explorer.

3. Щелкните пакет правой кнопкой мыши и выберите «**Создать**» — «**Другой репозиторий ABAP**» — «**Основные службы данных**» — «**Расширение метаданных**».

4. Появится мастер создания расширения метаданных, где вы должны ввести имя и описание. Мы назовем расширение метаданных документа покупки Z\_C\_PurchaseDocumentLrp и расширение метаданных элементов документа покупки Z\_C\_PurchaseDocumentItemLrp. По соглашению файлы расширения метаданных будут иметь те же имена, что и их аннотированные источники DDL.

5. Нажмите Готово, и откроется редактор исходного кода.

В файле расширения метаданных вы должны указать, какое представление CDS вы хотите аннотировать. В нашем случае мы сначала хотим аннотировать представление документа покупки Z\_C\_PurchaseDocumentLRP, чтобы настроить отчет списка. Кроме того, необходимо указать приоритет расширения метаданных, назначив его слою. Уровень *#CORE* имеет самый низкий приоритет, тогда как уровень *#CUSTOMER* имеет самый высокий приоритет. Мы просто воспользуемся слоем *#CORE*, чтобы позже можно было перезаписать аннотации нашего пользовательского интерфейса без изменения основного файла расширения, как показано в листинге 4.42.

@Metadata.layer: #CORE

annotate view Z\_C\_PurchaseDocumentLRP with{…}

Листинг 4.42. Заголовок файла расширения метаданных для представления Z\_C\_PurchaseDocumentLRP CDS

Чтобы включить расширения метаданных со стороны CDS DDL, на уровне заголовка необходимо добавить следующую аннотацию: ***@Metadata.allowExtensions:true***. Все аннотации пользовательского интерфейса, которые мы представим в следующих разделах, будут добавлены к расширениям метаданных, которые мы только что создали для представления потребления CDS документа покупки и элемента документа покупки (Z\_C\_PurchaseDocumentLrp и Z\_C\_PurchaseDocumentItemLrp).

### 4.4.2 Аннотации, относящиеся к пользовательскому интерфейсу, для List Report

В этом разделе мы рассмотрим наиболее важные аннотации пользовательского интерфейса шаблона List Report и настроим пользовательский интерфейс нашего List Report документов покупки. На рис. 4.52 показан обзор наиболее важных аннотаций пользовательского интерфейса CDS отчета списка и того, как они отражаются в пользовательском интерфейсе. К ним относятся:

1 @Search.searchable: true

2 @UI.selectionField

3 @UI.headerInfo

4 @UI.lineItem

Чтобы настроить пользовательский интерфейс отчета списка, мы должны аннотировать представление CDS Z\_C\_PurchaseDocumentItemLrp, поскольку соответствующий объект OData документа покупки будет привязан к экрану отчета списка.

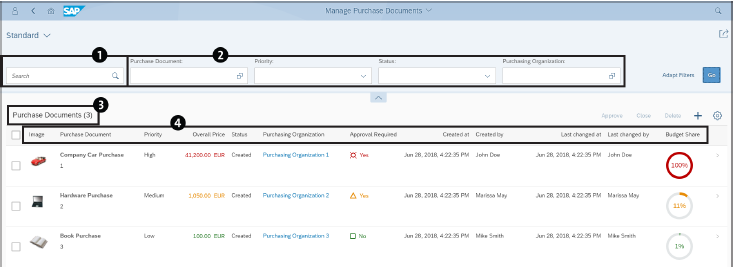


Рисунок 4.52 Наиболее важные аннотации пользовательского интерфейса CDS отчета о списках и то, как они будут отражены в шаблоне отчета о списках

#### Аннотация Line Item

Чтобы указать, какие поля объекта OData Z\_C\_PurchaseDocumentLRP изначально будут отображаться как столбцы в таблице отчета списка, мы должны аннотировать эти поля аннотацией ***@UI.lineItem***. Как это выглядит для поля PurchaseDocument, которое мы хотим отобразить вторым столбцом в таблице, показано в листинге 4.43.

@UI.lineItem: [{ importance: #HIGH, position: 20 }]

PurchaseDocument;

Листинг 4.43. Аннотация позиции для поля PurchaseDocument

Мы должны указать позицию поля, по которому сортируются столбцы таблицы. Кроме того, мы также можем определить важность поля. Поля с аннотацией важности: *#HIGH* всегда будет отображаться, даже, например, при отображении таблицы на маленьких дисплеях. Если важность не указана, позиция рассматривается как имеющая *#LOW* важность.

#### Header Info аннотация

Заголовок таблицы можно задать с помощью аннотации пользовательского интерфейса уровня заголовка ***@UI.headerInfo***. Для List Report значение, указанное для typeNamePlural, будет использоваться в качестве заголовка таблицы, как показано в листинге 4.44.

@UI.headerInfo: {

…

typeNamePlural: 'Purchase Documents',

… }

Листинг 4.44. Информация заголовка typeNamePlural для документов о покупке

#### Image аннотация

Аннотация ***@UI.headerInfo*** также содержит аннотацию ***imageUrl*** для определения поля, содержащего URL-адреса изображений, отображаемых в приложении. В нашем случае мы будем ссылаться на поле PurchaseDocumentImageURL, как показано в листинге 4.45.

@UI.headerInfo: {

…

imageUrl: 'PurchaseDocumentImageURL'

… }

Листинг 4.45. Аннотация imageUrl в заголовке для документов о покупке

Кроме того, поле PurchaseDocumentImageURL также должно быть снабжено аннотацией ***@Semantics.imageUrl: true***. Если затем мы добавим это поле в качестве первого столбца в нашу таблицу, используя аннотацию ***@UI.lineItem***, представленную ранее, шаблон автоматически отобразит HTML-тег *<img>* со свойством *src*, установленным на URL-адрес изображения, предоставленный значением поля.

#### Аннотация Criticality

Мы будем использовать значение поля *OverallPriceCriticality*, вычисленное на внутреннем уровне CDS, чтобы указать критичность столбца *OverallPrice*, как показано в листинге 4.46.

В зависимости от общего значения критичности цены общее значение поля цены будет отображаться определенным цветом, как указано в таблице 4.3. Аннотация критичности также является частью аннотации ***@UI.lineItem***.

@UI.lineItem: [{ importance: #HIGH,

position: 40,

criticality: 'OverallPriceCriticality',

value: 'OverallPrice' }]}

OverallPrice;

Листинг 4.46. Аннотация критичности позиции для поля OverallPrice

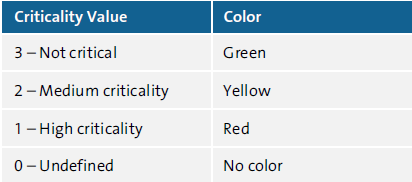


Таблица 4.3 Соответствие между значением критичности и цветом

Мы также используем значение поля *OverallPriceCriticality* таким же образом, чтобы указать критичность поля *IsApprovalRequired*.

#### Contact Quick View

Вы также можете отобразить контактную информацию в быстром представлении для поля, добавив аннотацию ***type: #AS\_CONTACT***, как показано в листинге 4.47, которая также является частью аннотации ***@UI.lineItem***, как показано на рисунке 4.53.

Данные быстрого представления, как показано в листинге 4.48, должны предоставляться через значение ассоциации: '\_PurchasingOrganization', где связанное представление должно содержать несколько полей с семантической аннотацией, как показано ранее на рис. 4.49.

@UI.lineItem: [{ importance: #MEDIUM,

position: 60,

type: #AS\_CONTACT,

value: '\_PurchasingOrganization',

label: 'Purchasing Organization' }]

PurchasingOrganization;

Листинг 4.47. Аннотация быстрого просмотра контактов для поля PurchasingOrganization

…

@Semantics.text: true

@Semantics.name.fullName: true

description as Description,

@Semantics: {

eMail.address: true,

eMail.type: [ #WORK ]

}

emailaddress as Email,

@Semantics.telephone.type: [ #WORK ]

phonenumber as Phone,

@Semantics.telephone.type: [ #FAX ]

faxnumber as Fax

…

Листинг 4.48. Обязательные аннотации для быстрого просмотра контактов в связанном представлении

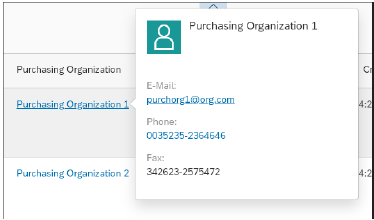


Рисунок 4.53 Быстрый просмотр контактов для закупочной организации

#### Аннотация Selection Field

Чтобы определенные поля отображались в расширенной области панели фильтров отчета списка как поля выбора, мы должны аннотировать эти поля аннотацией ***@UI.selectionField***. Как и в случае с аннотацией *LineItem*, мы также должны указать аннотацию *LineItem* для сортировки *selectionField*. Как показано в листинге 4.49, поля *PurchaseDocument* и *Priority* должны быть аннотированы, чтобы они отображались последовательно.

…

@UI.selectionField: [{ position: 10 }]

PurchaseDocument;

@UI.selectionField: [{ position: 20 }]

Priority;

…

Листинг 4.49. Аннотация поля выбора для панели фильтра

Другие поля выбора, показанные ранее на рис. 4.52, должны быть соответствующим образом аннотированы с увеличением значений позиции.

#### Аннотации Searchable

Аннотация уровня заголовка ***@Search.searchable***:***true*** помечает объект представления CDS, соответствующий сценариям поиска. Кроме того, при аннотации определенных полей с помощью ***@Search.defaultSearchElement: true*** эти элементы будут учитываться при свободном поиске и формировать область поиска. В нашем случае мы снабдим поле «Описание» документа покупки аннотациями поиска в главном представлении CDS. Мы сделаем это непосредственно в главном представлении, а не в расширении метаданных, так как мы хотим переместить аннотации пользовательского интерфейса только в файл расширения метаданных. Мы также введем ***@Search.fuzzinessThreshold***, чтобы сделать наш поиск более устойчивым к ошибкам. Значение может варьироваться от 0 до 1, где 1 означает, что при поиске будут возвращены только точные совпадения. Мы будем использовать рекомендуемое значение ***0,7***, которое вносит некоторую нечеткость, но не слишком большую:

…

@Search.defaultSearchElement : true

@Search.fuzzinessThreshold : 0.7

@Semantics.text: true

Description,

…

Аннотации *@Search* не относятся к категории аннотаций пользовательского интерфейса, но аннотирование сущности представления CDS документа покупки с помощью аннотаций *@Search* отобразит произвольное поле поиска в шаблоне отчета списка, как показано ранее на рис. 4.52. При создании документа метаданных службы OData SADL будет учитывать аннотации поиска и устанавливать для свойства *EntitySet sap:searchable* значение *true*, указывая пользовательскому интерфейсу или клиентам-потребителям, что объект поддерживает произвольный поиск. При отправке поискового запроса для определенного объекта SADL инициирует текстовый поиск SAP HANA (прозрачный для разработчика) с учетом индикатора нечеткости и области поиска, сформированной аннотированными полями ***@Search.defaultSearchElement: true*** представления CDS.

Запрос поиска OData *GET Z\_C\_PurchaseDocumentLrp?search=<SEARCH\_TERM>&…* будет преобразован SADL в инструкцию SELECT нечеткого текстового поиска SAP HANA с использованием предиката SAP HANA CONTAINS:

SELECT … FROM Z\_C\_PurchaseDocumentLRP

WHERE CONTAINS(Description, <SEARCH\_TERM>, FUZZY(0.7))…;

#### Value Helps

Справка по значению для поля должна быть определена с помощью аннотации ***@Consumption.valueHelpDefinition***. Если поле снабжено аннотацией

@Consumption.valueHelpDefinition: [ { entity:{name:'<valueHelp\_View\_Name>' ,

element: '<field\_name>'}}]

и поле используется в определении поведения, то во время активации привязки службы SADL примет во внимание эту информацию при создании службы OData и создаст аннотации справки по значениям для ключевого поля связанного представления. В листинге 4.50 показано определение справки по значению для *Status* в представлении *Z\_C\_PurchaseDocumentLRP* CDS.

@Consumption.valueHelpDefinition: [{ entity:{name:'Z\_C\_StatusVH' ,

element: 'Status'}}]

Status

Листинг 4.50. Справка по значению. Аннотация Odata для поля «Статус»

Соответствующее представление CDS, используемое в качестве представления справки по значениям, также требует определения нескольких аннотаций справки по значениям. Аннотация ***@ObjectModel.resultSet.sizeCategory: #XS*** сделает справку по значению для *Status* доступной через раскрывающийся список с множественным выбором в пользовательском интерфейсе, в отличие от справки по обычному значению. Аннотация ***@ObjectModel.dataCategory: #VALUE\_HELP*** классифицирует это представление CDS как представление справки по значениям, как показано в листинге 4.51.

…

@VDM.viewType: #CONSUMPTION

@ObjectModel.resultSet.sizeCategory: #XS

@ObjectModel.dataCategory: #VALUE\_HELP

**define view** Z\_C\_StatusVH **as select from** Z\_I\_PurchaseDocumentStatus {

@ObjectModel.text.element: ['StatusText']

key Status,

@Search: { defaultSearchElement: true,

ranking: #HIGH,

fuzzinessThreshold: 0.7 }

StatusText

}

Листинг 4.51. Значение Help CDS View for Status

#### Включение быстрых действий (actions)

Быстрые действия, которые мы добавили в корневой узел документа покупки (*Z\_I\_PurchaseDocument\_U*) наших неуправляемых бизнес-объектов, можно включить в пользовательском интерфейсе List Report, добавив аннотации ***@UI.lineItem***, показанные в листинге 4.52, в поле *PurchaseDocument*.

@UI: {

lineItem: [

…

{ type: #FOR\_ACTION, dataAction: 'Approve', position: 10, label: 'Approve' },

{ type: #FOR\_ACTION, dataAction: 'Reject', position: 20, label: 'Close' }]

…

]

}

PurchaseDocument;

Листинг 4.52. Аннотация позиции для включения быстрых действий BOPF в пользовательском интерфейсе отчета списка

4.4.3 246