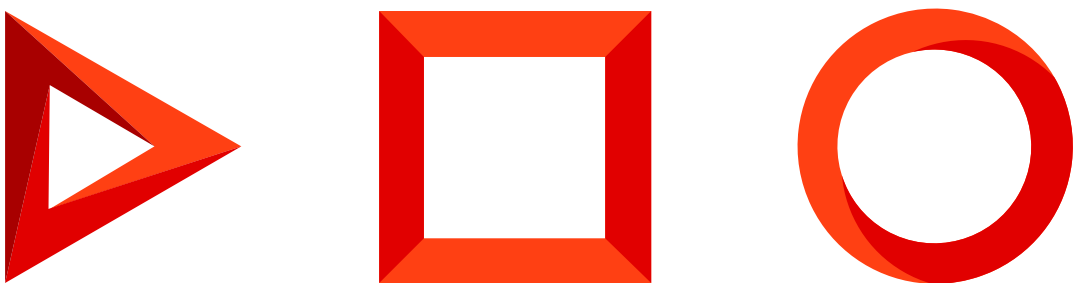
Creatio Academy

Чаты

API для работы с чатами

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

API для работы с чатами

Средний

Базовая функциональность работы с чатами находится в предустановленном пакете `OmnichannelMessaging`.

Описание настройки интеграции с мессенджерами содержится в блоке статей Настройка чатов.

Интеграция с мессенджером

Библиотеки, в которых находится основная часть функциональности интеграции с мессенджером:

- `OmnichannelMessaging.dll`.
- `OmnichannelProviders.dll`.

Возможности интеграции с мессенджером предоставляет класс `MessageManager`.

Основные **методы** класса `MessageManager`:

- `Receive` — принимает входящие сообщения. Для сообщения используется универсальный формат (`UnifiedMessage` — унифицированный класс сообщения) или строка, которая в дальнейшем будет конвертирована в универсальный формат. При необходимости использования конвертации метод использует класс, который реализует интерфейс `IIncomeMessageWorker`. Для каждого мессенджера используется собственная реализация данного интерфейса. Также выполняется сохранение сообщения в системе.
- `Send` — отправляет исходящие сообщения. Система подбирает подходящий класс-отправитель сообщения. Класс должен реализовывать интерфейс `IOutcomeMessageWorker`. Также выполняется сохранение сообщения в системе.
- `Register` — добавляет канал при настройке интеграции с Facebook Messenger. Метод использует класс, который реализует интерфейс `IMessengerRegistrationWorker`. Используется для мессенджеров, которые требуют выполнения дополнительных действий при регистрации, например, получение токена.
- `GetMessagesByChatId` — получает сообщения, которые относятся к чату. Для сообщения используется формат `IEnumerable<UnifiedMessage>`. Чат передается в параметрах метода.

Прием сообщений

Специфичные для мессенджеров **сервисы** приема сообщений:

- `FacebookOmnichannelMessagingService` — сервис приема сообщений от Facebook Messenger.
- `TelegramOmnichannelMessagingService` — сервис приема сообщений от Telegram.

Для обоих сервисов базовым является сервис `OmnichannelMessagingService`, который содержит набор общих для мессенджеров методов.

`InternalReceive` — основной **метод** сервиса `OmnichannelMessagingService`. Принимает сообщения.

Сервисы для работы с чатами в Creatio

Сервисы для работы с чатами в Creatio:

- `OmnichannelChatService` — сервис для работы с чатами, который позволяет получить историю, чаты оператора и т. д.
- `OmnichannelOperatorService` — сервис операторов, который позволяет получить и изменить статус.

Основные **методы** сервиса `OmnichannelChatService`:

- `AcceptChat` — закрепляет чат за текущим пользователем.
- `GetUnreadChatsCount` — получает количество непрочитанных чатов.
- `MarkChatAsRead` — помечает все сообщения в указанном чате как прочитанные.
- `GetConversation` — получает сообщения чата для отображения в коммуникационной панели.
- `CloseActiveChat` — закрывает указанный чат.
- `GetChats` — получает все чаты для указанного оператора.
- `GetChatActions` — получает список действий, доступных для очереди указанного чата.
- `GetUnreadMessagesCount` — получает количество непрочитанных сообщений во всех чатах оператора.

Добавить новый провайдер канала

1. Добавьте новый провайдер в справочник [*Провайдер канала*] ([*Channel provider*]).
2. Реализуйте хранение данных нового канала в базе данных.
3. Зарегистрируйте новый канал в базе данных.
4. Создайте веб-сервис для приема сообщений.
5. Реализуйте конвертацию входящего сообщения в универсальный формат Creatio.
6. Реализуйте получение данных профиля пользователя в классе, который реализует интерфейс `IProfileDataProvider`.
7. Реализуйте загрузку вложений.
8. Реализуйте отправку сообщений в классе, который реализует интерфейс `IOutcomeMessageWorker`.
9. Выполните связывание интерфейсов.

Добавить пользовательский провайдер канала


Сложный

Пример. В on-site приложении Creatio создать пользовательский провайдер канала `Test`.

1. Добавить пользовательский провайдер канала в Creatio

1. Перейдите в дизайнер системы по кнопке .
2. В блоке [*Настройка системы*] ([*System setup*]) перейдите по ссылке [*Справочники*] ([*Lookups*]).
3. Откройте справочник [*Провайдер канала*] ([*Channel provider*]) и добавьте в него значение "`Test`".

2. Настроить хранение данных пользовательского канала

1. Создайте таблицу базы данных и определите ее структуру. Структура таблицы зависит от провайдера и содержит данные для отправки и приема сообщений. Как правило, для отправки сообщений провайдеры используют авторизационный токен.Для этого выполните SQL-скрипт, который приведен ниже.

SQL-скрипт, который создает таблицу `[TestMsgSettings]`

MSSQL

```
IF OBJECT_ID('TestMsgSettings') IS NULL
    BEGIN
    CREATE TABLE TestMsgSettings(
        Id uniqueidentifier NOT NULL DEFAULT (newid()),
        Token nvarchar(250) NOT NULL DEFAULT (''),
        UserName nvarchar(250) NOT NULL DEFAULT (''),
        CONSTRAINT PK_TestMsgSettings_Id PRIMARY KEY CLUSTERED (Id)
    )
END
```

PostgreSQL

```
CREATE TABLE IF NOT EXISTS public."TestMsgSettings" (
    "Id" uuid NOT NULL DEFAULT uuid_generate_v4(),
    "Token" character varying(250) COLLATE pg_catalog."default" NOT NULL DEFAULT ''::characte
    "UserName" character varying(250) COLLATE pg_catalog."default" NOT NULL DEFAULT ''::chara
    CONSTRAINT "PK_TestMsgSettings_Id" PRIMARY KEY ("Id")
);
```

`[TestMsgSettings]` — таблица для хранения данных пользовательского канала. Для названия таблицы используйте шаблон `[SomeChannelProviderNameMsgSettings]`, где `SomeChannelProviderName` — имя

провайдера канала.

2. Зарегистрировать пользовательский канал в базе данных. Для этого добавьте запись в таблицу `[Channel]`. Поля, которые необходимо заполнить, приведены в таблице ниже.

Поля таблицы `[Channel]`

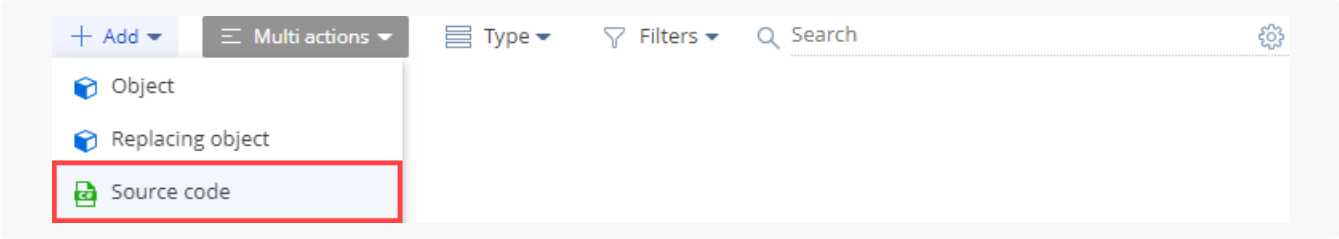
Поле	Описание
`[Name]`	Название канала.
`[ProviderId]`	Идентификатор пользовательского провайдера.
`[MsgSettingsId]`	Идентификатор записи в таблице `[TestMsgSettings]`.
`[Source]`	Идентификатор канала внутри мессенджера, например, идентификатор страницы на Facebook или идентификатор клиента в Telegram. Позволяет определить получателя по сообщению от мессенджера.

3. Создать веб-сервис для приема сообщений

1. Создайте схему [*Исходный код*] ([*Source code*]).

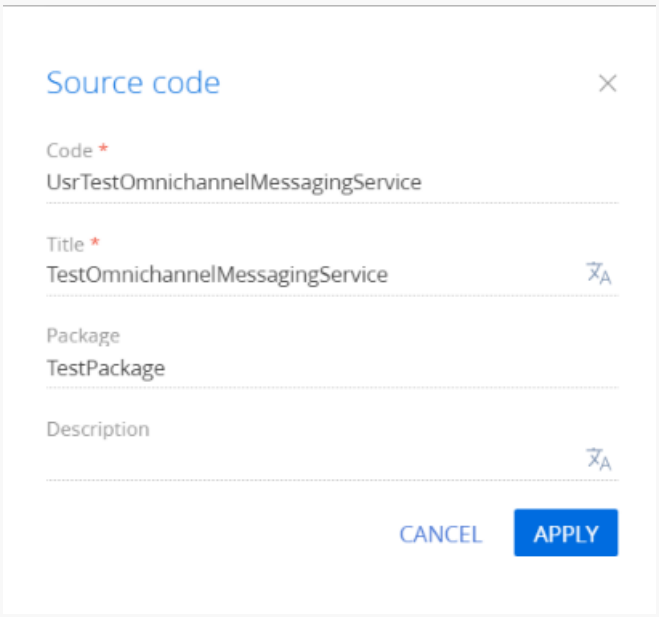
   a. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.

   b. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).



   c. В дизайнере исходного кода заполните **свойства схемы**:

   - [*Код*] ([*Code*]) — "`UsrTestOmnichannelMessagingService`".
   - [*Заголовок*] ([*Title*]) — "`TestOmnichannelMessagingService`".



Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).

2. Создайте класс веб-сервиса.
   a. В дизайнере схем добавьте пространство имен `Terrasoft.Configuration.Omnichannel.Messaging`.
   b. С помощью директивы `using` добавьте пространства имен, типы данных которых задействованы в классе.
   c. Добавьте название класса, которое соответствует названию схемы (свойство [*Код*] ([*Code*])).
   d. В качестве родительского класса укажите класс `OmnichannelMessagingService`, который содержит базовые методы.
   e. Для класса добавьте атрибуты `[ServiceContract]` и `[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)]`.
3. Реализуйте метод класса (т. е., конечную точку) веб-сервиса. Для этого в дизайнере исходного кода добавьте в класс метод `public void ReceiveMessage(TestIncomingMessage message)`. Мессенджер присылает сообщение на конечную точку `receive`.

   Исходный код пользовательского веб-сервиса `TestOmnichannelMessagingService` представлен ниже.

`TestOmnichannelMessagingService`

```
namespace Terrasoft.Configuration.Omnichannel.Messaging
{
    using System;
    using System.ServiceModel;
    using System.ServiceModel.Activation;
    using System.ServiceModel.Web;
    using Terrasoft.Common;
    using Terrasoft.Core;
    using Terrasoft.Core.DB;
```

```
using Terrasoft.Web.Common.ServiceRouting;

#region Class: TestOmnichannelMessagingService

/* The service that sends and receives messages from the messaging integration API. */
[ServiceContract]
[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.R
[DefaultServiceRoute]
public class TestOmnichannelMessagingService : OmnichannelMessagingService
{

    #region Constructors: Public

    public TestOmnichannelMessagingService() : base() {
    }

    /* Initialize a new instance of TestOmnichannelMessagingService. */
    public TestOmnichannelMessagingService(UserConnection userConnection) : base(userConn
    }

    #endregion

    #region Methods: Private

    private void GetChannelAndQueueBySource(MessagingMessage message) {
        Select channelSelect = new Select(UserConnection)
            .Top(1).Column("Id")
            .Column("ChatQueueId")
            .From("Channel")
            .Where("Source").IsEqual(Column.Parameter(message.Recipient))
            .And("IsActive").IsEqual(Column.Parameter(true)) as Select;
        channelSelect.ExecuteReader(reader => {
            message.ChannelId = reader.GetColumnValue<Guid>("Id").ToString();
            message.ChannelQueueId = reader.GetColumnValue<Guid>("ChatQueueId");
        });
    }

    #endregion

    #region Methods: Public

    /* Receive messages from the integration API. */
    [OperationContract]
    [WebInvoke(UriTemplate = "receive", Method = "POST", RequestFormat = WebMessageFormat

    /* message is a test provider message. */
    public void ReceiveMessage(TestIncomingMessage message) {

        /* Convert the message. */
```

```
            MessagingMessage messagingMessage = new MessagingMessage(TestIncomingMessageConve

            /* Identify channel using the [Source] field. */
            GetChannelAndQueueBySource(messagingMessage);

            /* Create a new chat if an open chat with this client is not found, or add a mess
            Create a contact before first communication with a client and fill out their cred
            InternalReceive(messagingMessage);
        }

        #endregion

    }

    #endregion

}
```

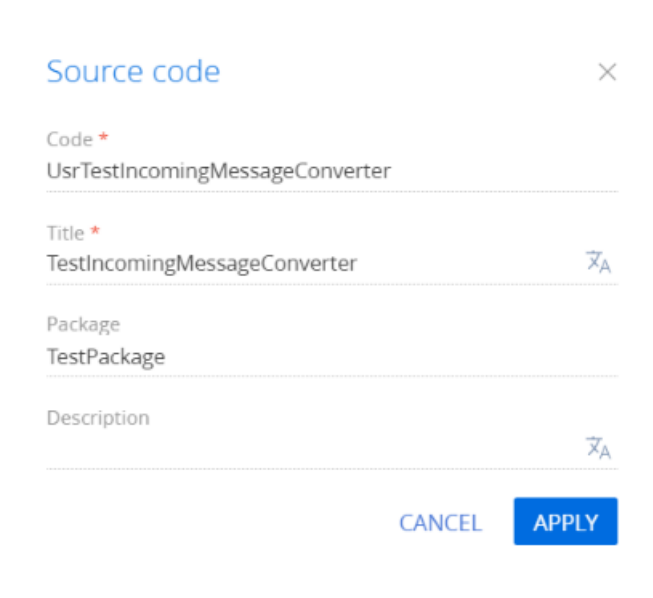
4. На панели инструментов дизайнера исходного кода нажмите [*Опубликовать*] ([*Publish*]) для выполнения изменений на уровне базы данных.

4. Реализовать конвертацию входящего сообщения

Для обработки входящего сообщения реализуйте класс `TestIncomingMessageConverter`, который конвертирует сообщение с типом `TestIncomingMessage` (сообщение в формате мессенджера) в универсальный формат Creatio (класс `MessagingMessage`).

Чтобы **создать класс-конвертер**:

1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).
3. В дизайнере схем заполните **свойства схемы**:
   - [*Код*] ([*Code*]) — "`UsrTestIncomingMessageConverter`".
   - [*Заголовок*] ([*Title*]) — "`TestIncomingMessageConverter`".



Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).

4. В дизайнере схем добавьте исходный код. Исходный код класса-конвертера `TestIncomingMessageConverter` представлен ниже.

TestIncomingMessageConverter

```
namespace Terrasoft.Configuration.Omnichannel.Messaging
{
    using OmnichannelProviders.Domain.Entities;
    using System;
    using System.Collections.Generic;

    public class TestIncomingMessage : UnifiedMessage
    { }

    public static class TestIncomingMessageConverter
    {
        #region Methods: Public

        public static MessagingMessage Convert(TestIncomingMessage message) {
            var messageType = MessageType.Text;
            var messageId = Guid.NewGuid();
            var result = new MessagingMessage {
                Id = messageId,
                Message = message.Message,
                /* Получатель сообщения – идентификатор страницы, либо клиента, добавленный в
                Recipient = message.Recipient,
                /* Идентификатор отправителя сообщения внутри мессенджера. Будет связан с кон
                Sender = message.Sender,
                Timestamp = message.Timestamp,
```

```
                /* Идентификатор канала внутри системы, поле MsgSettingsid из объекта. */
                ChannelId = message.ChannelId,
                MessageDirection = MessageDirection.Incoming,
                MessageType = messageType,
                /* Указывает на источник канала (сторонние разработчики). */
                Source = ChannelType.ThirdParty,
                /* Имя провайдера. В дальнейшем будет использоваться как идентификатор провай
                ChannelName = "Test"
            };
            if (messageType != MessageType.Text) {

                /* Блок [Attachments] заполняется в зависимости от формата доступа к файлам,
                result.Attachments = new List<MessageAttachment>();
                foreach (var attachment in message.Attachments){
                    result.Attachments.Add(new MessageAttachment {
                        MessageId = messageId,
                        UploadUrl = attachment.UploadUrl,
                        FileType = attachment.FileType
                    });
                }
            }
            return result;
        }

        #endregion

    }
}
```

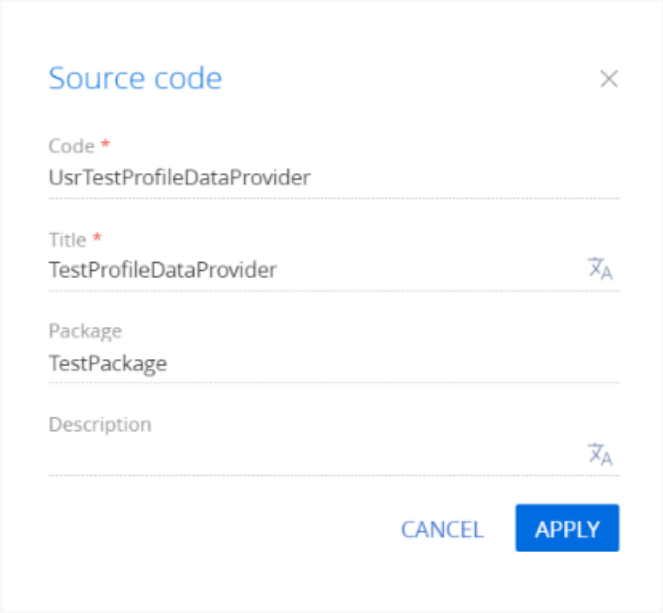
5. На панели инструментов дизайнера исходного кода нажмите [*Опубликовать*] ([*Publish*]) для выполнения изменений на уровне базы данных.

5. Реализовать получение данных профиля пользователя

Мессенджеры предоставляют API для получения данных клиентов, которые отправляют сообщения. Чтобы получить данные профиля пользователя, создайте класс, который реализует интерфейс `IProfileDataProvider`.

Чтобы **создать класс для получения данных профиля пользователя**:

1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).
3. В дизайнере схем заполните **свойства схемы**:
   - [*Код*] ([*Code*]) — `"UsrTestProfileDataProvider"`.
   - [*Заголовок*] ([*Title*]) — `"TestProfileDataProvider"`.



Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).

4. В дизайнере схем добавьте исходный код.

- Класс отправляет запрос на адрес `https://graph.test.com/`, чтобы получить данные в формате `TestProfileData` (предполагаемый формат мессенджера).
- Класс конвертирует полученный ответ во внутренний формат `ProfileData`.
- При создании контакта класс добавляет полученные данные (например, имя, фамилия, фото) ко внутреннему формату. Если запрос неуспешный или данные некорректны, то класс создает контакт с именем `[Новый контакт][Имя канала]-[Идентификатор клиента в мессенджере]`.

Исходный код класса `TestProfileDataProvider` представлен ниже.

`TestProfileDataProvider`

```
namespace Terrasoft.Configuration.Omnichannel.Messaging
{
    using Newtonsoft.Json;
    using OmnichannelProviders.Domain.Entities;
    using OmnichannelProviders.Interfaces;
    using System.IO;
    using System.Net;
    using Terrasoft.Core;

    public class TestProfileData {
        public string first_name { get; set; }
        public string last_name { get; set; }
    }

    #region Class: TestProfileDataProvider
```

```
/* Retrieve the profile data from the Test provider. */
public class TestProfileDataProvider : IProfileDataProvider
{
    #region Properties: Private

    private readonly string _testProviderApiUrl = "https://graph.test.com/";

    #endregion

    #region Constructors: Public

    /* Initialize a new instance of FacebookProfileDataProvider. */
    public TestProfileDataProvider(UserConnection userConnection) {
    }

    #endregion

    #region Methods: Public

    /* Retrieve the profile data from Facebook.
    profileId is the Facebook profile ID.
    channelId is the channel from which to send the request.
    Return the contact ID. */
    public ProfileData GetProfileDataByProfileId(string profileId, string channelId)    {
        var requestUrl = string.Concat(_testProviderApiUrl, profileId);
        WebRequest request = WebRequest.Create(requestUrl);
        try {
            using (var response = request.GetResponse()) {
                using (Stream stream = response.GetResponseStream()) {
                    using (StreamReader sr = new StreamReader(stream)) {
                        var testProfile = JsonConvert.DeserializeObject<TestProfileData>(
                        return new ProfileData {
                            FirstName = testProfile.first_name,
                            LastName = testProfile.last_name,
                        };
                    }
                }
            }
        }
        catch {
            return new ProfileData();
        }
    }

    #endregion

}
```

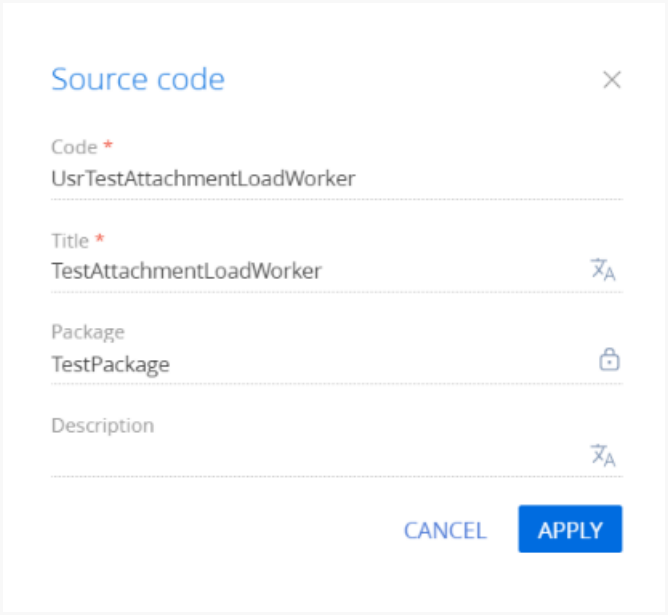
```
    #endregion
}
```

5. На панели инструментов дизайнера исходного кода нажмите [*Опубликовать*] ([*Publish*]) для выполнения изменений на уровне базы данных.

6. Реализовать загрузку вложений

Для загрузки вложений создайте класс, который реализует интерфейс `IAttachmentsLoadWorker`:

1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).
3. В дизайнере схем заполните **свойства схемы**:
   - [*Код*] ([*Code*]) — "`UsrTestAttachmentLoadWorker`".
   - [*Заголовок*] ([*Title*]) — "`TestAttachmentLoadWorker`".



   Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).
4. В дизайнере схем добавьте исходный код. Внутренний класс `AttachmentsDownloader` загружает вложение по ссылке и сохраняет его в таблицу `[OmnichannelMessageFile]` базы данных.

   Исходный код класса `TestAttachmentLoadWorker` представлен ниже.

`TestAttachmentLoadWorker`

```
namespace Terrasoft.Configuration.Omnichannel.Messaging
{
```

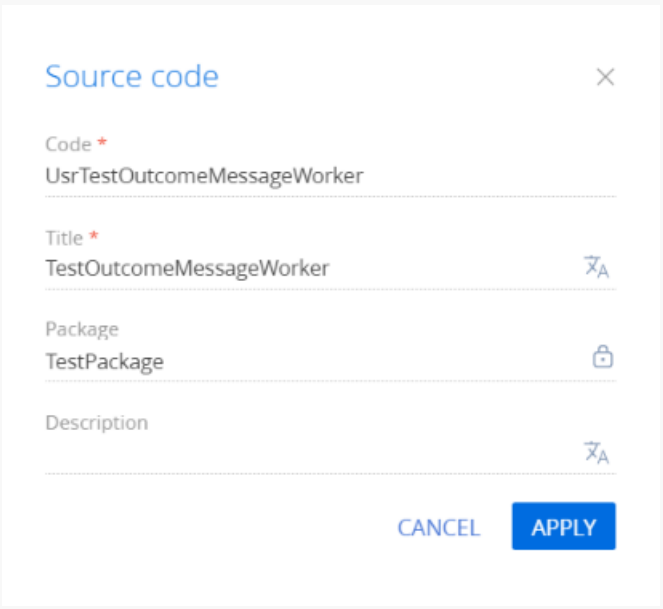
```
    using OmnichannelProviders;
    using OmnichannelProviders.Domain.Entities;
    using OmnichannelProviders.MessageConverters;
    using Terrasoft.Core;

    #region Class: TestAttachmentLoadWorker

    /* The class that loads attachments from the Test provider. */
    public class TestAttachmentLoadWorker : IAttachmentsLoadWorker
    {

        #region Properties: Protected

        protected UserConnection UserConnection;
        protected AttachmentsDownloader AttachmentsDownloader;

        #endregion

        #region Constructors: Public

        /* Initialize a new instance of the TestAttachmentLoadWorker class. */
        public TestAttachmentLoadWorker(UserConnection userConnection) {
            UserConnection = userConnection;
            AttachmentsDownloader = new AttachmentsDownloader(userConnection);
        }

        #endregion

        #region Methods: Public

        /* Load the attachments.
        incomeAttachment is the attachment from the messenger.
        message is the source message. */
        public void Load(MessageAttachment incomeAttachment, UnifiedMessage message) {
            incomeAttachment.FileName = FileUtilities.GetFileNameFromUrl(incomeAttachment.Upl
            incomeAttachment.FileId = AttachmentsDownloader.Load(incomeAttachment);
        }

        #endregion
    }

    #endregion
}
```

5. На панели инструментов дизайнера исходного кода нажмите [*Опубликовать*] ([*Publish*]) для выполнения изменений на уровне базы данных.

7. Реализовать отправку сообщений

Для отправки сообщений **создайте класс**, который реализует интерфейс `IOutcomeMessageWorker`:

1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).
3. В дизайнере схем заполните **свойства схемы**:
   - [*Код*] ([*Code*]) — "`UsrTestOutcomeMessageWorker`".
   - [*Заголовок*] ([*Title*]) — "`TestOutcomeMessageWorker`".



   Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).
4. В дизайнере схем добавьте исходный код. Класс `TestOutcomeMessageWorker` переводит сообщение в формат мессенджера и отправляет его, используя API мессенджера. Для выполнения отправки может использоваться токен, который необходимо хранить в таблице `[TestMsgSettings]`. Доступ к таблице предоставляет переданный в конструкторе `UserConnection`. Класс отправляет сообщение на адрес `https://graph.test.com/`, используя внутренний класс `HttpRequestSender`.

   Исходный код класса `TestOutcomeMessageWorker` представлен ниже.

`TestOutcomeMessageWorker`

```
namespace Terrasoft.Configuration.Omnichannel.Messaging
{
    using Newtonsoft.Json;
    using Newtonsoft.Json.Serialization;
    using OmnichannelProviders.Application.Http;
    using OmnichannelProviders.Domain.Entities;
```

```
    using OmnichannelProviders.MessageWorkers;
    using Terrasoft.Core;

    #region Class: TestOutcomeMessageWorker

    /* The class that sends messages to the Test provider. */
    public class TestOutcomeMessageWorker : IOutcomeMessageWorker
    {

        #region Properties: Protected

        protected UserConnection UserConnection;
        private readonly string _testProviderApiUrl = "https://graph.test.com/";

        #endregion

        #region Constructors: Public

        /* Initialize a new instance of the TestOutcomeMessageWorker class. */
        public TestOutcomeMessageWorker(UserConnection userConnection) {
            UserConnection = userConnection;
        }

        #endregion

        #region Methods: Public

        /* Send the message to Test provider.
        message is the UnifiedMessage message. */
        public string SendMessage(UnifiedMessage unifiedMessage, out bool success) {
            var serializerSettings = new JsonSerializerSettings();
            serializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver(
            var json = JsonConvert.SerializeObject(unifiedMessage, serializerSettings);
            var requestUrl = string.Concat(_testProviderApiUrl, json);
            var result = new HttpRequestSender().PostAsync(requestUrl, json).Result;
   success = true;
            return result;
        }

  public string PassControlToPrimaryReceiver(UnifiedMessage message) {
   return string.Empty;
  }

        #endregion
    }

    #endregion
}
```

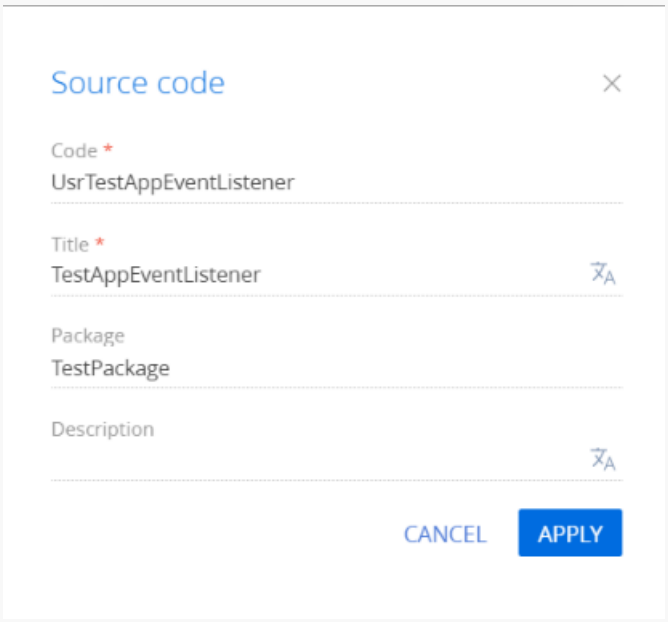
5. На панели инструментов дизайнера исходного кода нажмите [*Опубликовать*] ([*Publish*]) для выполнения изменений на уровне базы данных.

8. Связать интерфейсы

Чтобы **связать интерфейсы**, создайте класс.

Чтобы **создать класс**:

1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).
3. В дизайнере схем заполните **свойства схемы**:
   - [*Код*] ([*Code*]) — "`UsrTestAppEventListener`".
   - [*Заголовок*] ([*Title*]) — "`TestAppEventListener`".



   Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).
4. В дизайнере схем добавьте исходный код. В качестве родительского класса укажите класс `AppEventListenerBase`. Созданные классы связываются по тегу `Test`. Укажите тег в поле `ChannelName` на шаге 4. Приложение использует тег, чтобы определить файлы, которые необходимо использовать для получения данных профиля, загрузки вложений и отправки сообщений.

   Исходный код класса `TestAppEventListener` представлен ниже.

`TestAppEventListener`

```
namespace Terrasoft.Configuration.Omnichannel.Messaging
```

```
{
    using Common;
    using Core;
    using OmnichannelProviders;
    using OmnichannelProviders.Interfaces;
    using OmnichannelProviders.MessageWorkers;
    using Terrasoft.Core.Factories;
    using Web.Common;

    #region Class : TestAppEventListener

    /* The class that runs prerequisites for OmnichannelMessaging on application start. */
    public class TestAppEventListener : AppEventListenerBase
    {

        #region Fields : Protected

        protected UserConnection UserConnection {
            get;
            private set;
        }

        #endregion

        #region Methods : Protected

        /* Retrieve the user connection from the application event scope.
        context is the app event scope.
        Returns user connection. */
        protected UserConnection GetUserConnection(AppEventContext context) {
            var appConnection = context.Application["AppConnection"] as AppConnection;
            if (appConnection == null) {
                throw new ArgumentNullOrEmptyException("AppConnection");
            }
            return appConnection.SystemUserConnection;
        }

        protected void BindInterfaces() {
            ClassFactory.Bind<IAttachmentsLoadWorker, TestAttachmentLoadWorker>("Test");
            ClassFactory.Bind<IProfileDataProvider, TestProfileDataProvider>("Test");
            ClassFactory.Bind<IOutcomeMessageWorker, TestOutcomeMessageWorker>("Test");
        }

        #endregion

        #region Methods : Public

        /* Handle the app start.
        context is the app event scope. */
```

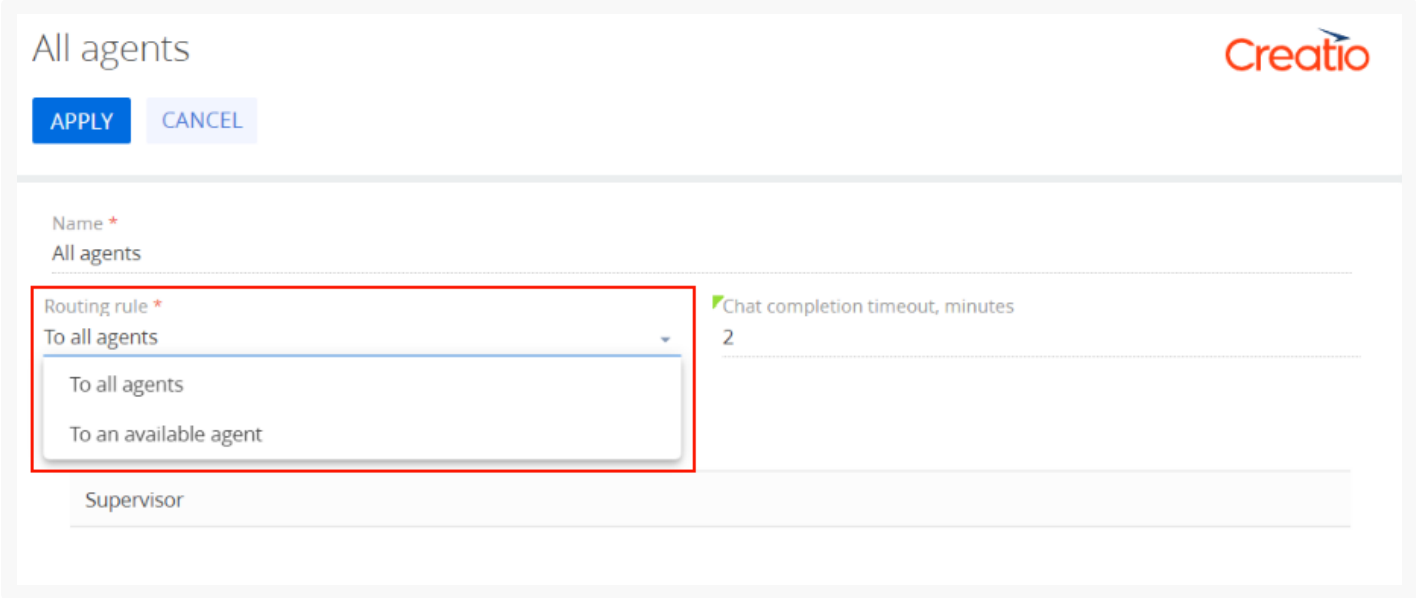
```
        public override void OnAppStart(AppEventContext context) {
            base.OnAppStart(context);
            UserConnection = GetUserConnection(context);
            BindInterfaces();
        }

        #endregion

    }

    #endregion

}
```

5. На панели инструментов дизайнера исходного кода нажмите [*Опубликовать*] ([*Publish*]) для выполнения изменений на уровне базы данных.

Добавить механизм маршрутизации чатов

Сложный

Для настройки маршрутизации чатов необходимо в выпадающем списке настройки очереди выбрать правило маршрутизации. Таким образом определяется механизм распределения чатов на операторов.



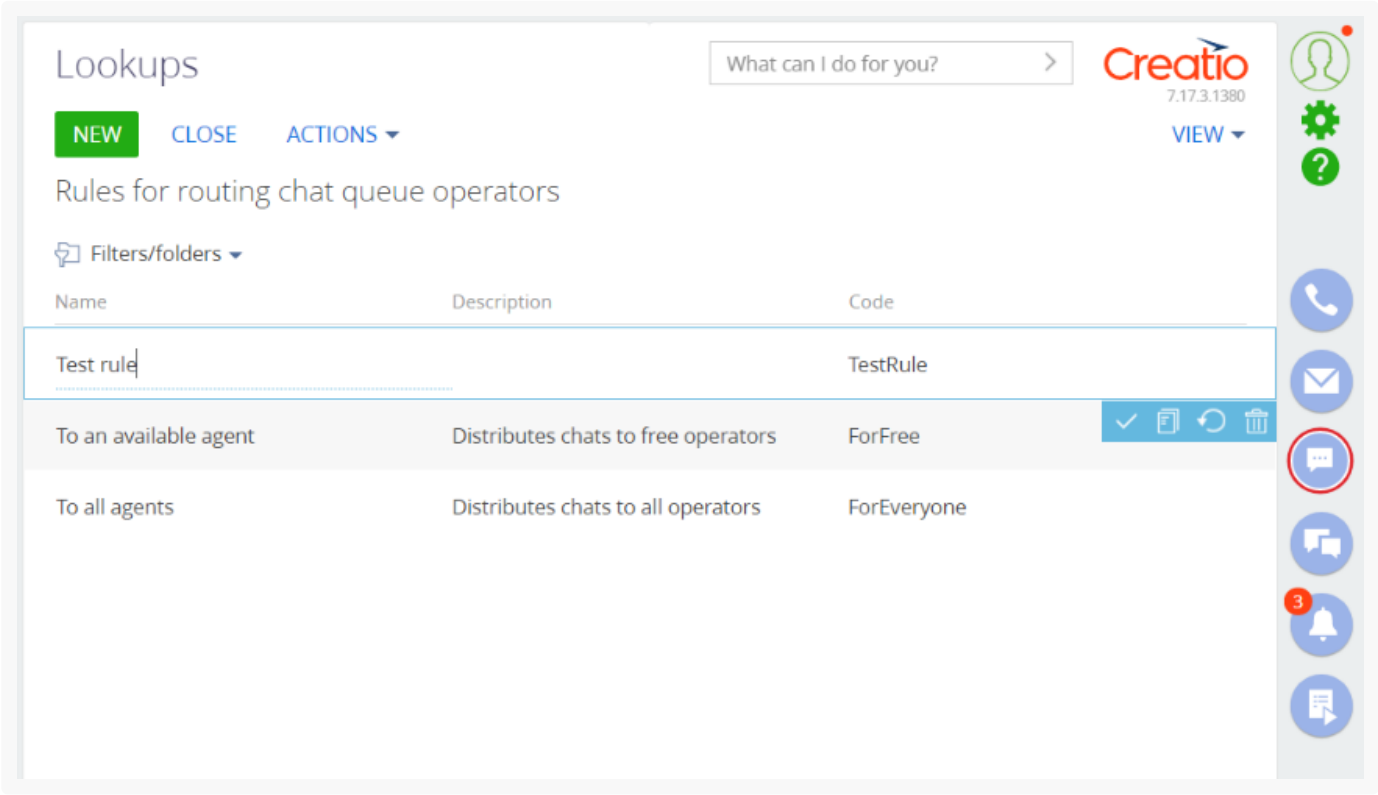
Рассмотрим пример добавления пользовательского механизма распределения.

Пример. Добавить пользовательский механизм распределения чатов. Системному пользователю (Supervisor) предоставить доступ к новым чатам. Оператору предоставить доступ к уже

назначенным чатам.

1. Добавить новое правило маршрутизации чатов

1. Перейдите в дизайнер системы по кнопке .
2. В блоке [*Настройка системы*] ([*System setup*]) перейдите по ссылке [*Справочники*] ([*Lookups*]).
3. Используя фильтр в верхней части страницы, найдите справочник "Правила маршрутизации чатов в очереди" ("Rules for routing chat queue operators").
4. Откройте наполнение справочника и добавьте **новое правило**:
   - [*Название*] ([*Name*]) — "`Test rule`".
   - [*Код*] ([*Code*]) — "`TestRule`".



2. Создать класс, который реализует интерфейс `IOperatorRoutingRule`

На этом шаге можно создать класс-наследник класса `BaseOperatorRoutingRule` или новый класс, реализующий интерфейс `IOperatorRoutingRule`.

Класс `BaseOperatorRoutingRule` содержит в себе абстрактные методы `PickUpFreeQueueOperators` и `GetChatOperator`, которые необходимо реализовать.

Абстрактные методы класса `BaseOperatorRoutingRule`

```
/// <summary>
/// Pick up and return queue agent IDs.
/// </summary>
/// <param name="chatId"><see cref="OmniChat"/> The agent ID.</param>
/// <param name="queueId"><see cref="ChatQueue"/> The instance ID.</param>
/// <returns>The queue agent IDs.</returns>
protected abstract List<Guid> PickUpFreeQueueOperators(Guid chatId, Guid queueId);

/// <summary>
/// Returns the <see cref="OmniChat"/> agent ID.
/// </summary>
/// <param name="chatId"><see cref="OmniChat"/> The agent ID.</param>
/// <returns>The chat agent.</returns>
protected abstract Guid GetChatOperator(Guid chatId);
```

При этом в класс `BaseOperatorRoutingRule` уже заложена реализация интерфейса `IOperatorRoutingRule`, с использованием логики, приведенной ниже.

Реализация интерфейса `IOperatorRoutingRule`

```
public List <Guid> GetOperatorIds(string chatId, Guid queueId) {
    var parsedChatId = Guid.Parse(chatId);
    var chatOperator = GetChatOperator(parsedChatId);
    return chatOperator.IsNotEmpty() ? new List <Guid> {
        chatOperator
    }: PickUpFreeQueueOperators(parsedChatId, queueId);
}
```

Если чату уже **назначен оператор** (в колонке `[OperatorId]` указан идентификатор пользователя), то необходимо выбрать этого оператора. Если чату **не назначен оператор**, то необходимо, используя метод `PickUpFreeQueueOperators` получить оператора или операторов в виде `List<Guid>`. В данном случае `Guid` это идентификаторы операторов из таблицы `[SysAdminUnit]`, которые в итоге и получат уведомления о новом чате, а также доступ к чату через коммуникационную панель.

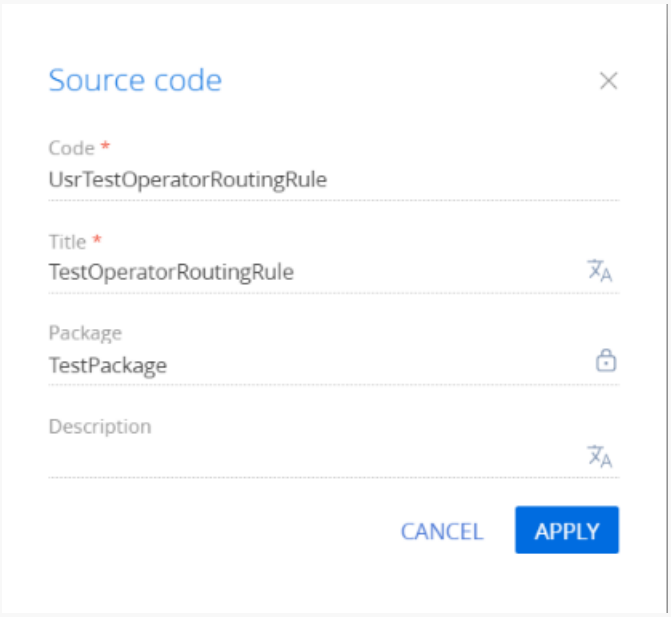
Создадим класс `TestOperatorRoutingRule` реализующий простейшую логику маршрутизации чата на системного пользователя (Supervisor).

Чтобы **создать класс**:

1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).
3. В дизайнере схем заполните **свойства схемы**:

   - [*Код*] ([*Code*]) — "`UsrTestOperatorRoutingRule`".

- [*Заголовок*] ([*Title*]) — "TestOperatorRoutingRule".



Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).

4. В дизайнере схем добавьте исходный код.

TestOperatorRoutingRule

```
namespace Terrasoft.Configuration.Omnichannel.Messaging {
    using System;
    using System.Collections.Generic;
    using Terrasoft.Core;
    using Terrasoft.Core.DB;

    #region Class: ForEveryoneOperatorRoutingRule

    /// <summary>
    /// Retrieve chat agents.
    /// </summary>
    public class TestOperatorRoutingRule: BaseOperatorRoutingRule {

        #region Constructors: Public

        /// <summary>
        /// Initialize a new instance of <see cref="TestOperatorRoutingRule"/>.
        /// </summary>
        /// <param name="userConnection"><see cref="UserConnection"/> the instance.</param>
        public TestOperatorRoutingRule(UserConnection userConnection) : base(userConnection) {}

        #endregion
```

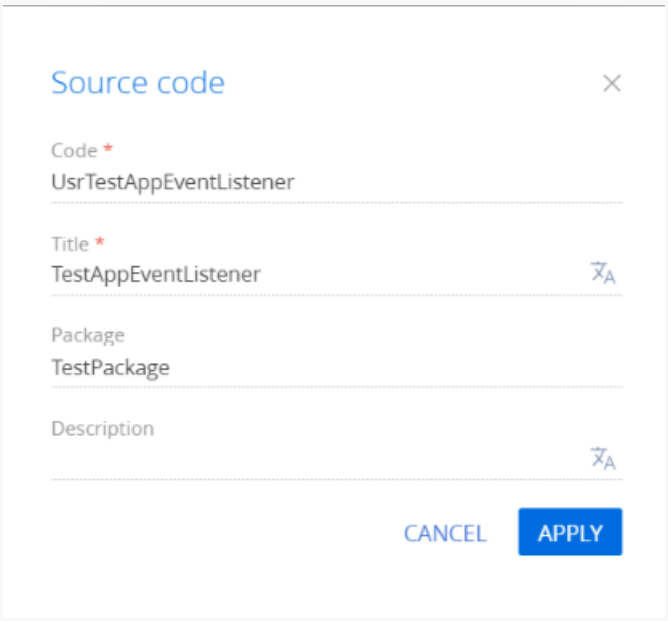
```
        #region Methods: Private

        #endregion

        #region Methods: Protected

        /// <inheritdoc cref="BaseOperatorRoutingRule.PickUpFreeQueueOperators(Guid, Guid)"/>
        protected override List < Guid > PickUpFreeQueueOperators(Guid chatId, Guid queueId) {
            return new List < Guid > {
                Guid.Parse("7F3B869F-34F3-4F20-AB4D-7480A5FDF647")
            };
        }

        /// <inheritdoc cref="BaseOperatorRoutingRule.GetChatOperator(Guid)"/>
        protected override Guid GetChatOperator(Guid chatId) {
            Guid operatorId = (new Select(UserConnection).Column("OperatorId").From("OmniChat",
            as Select).ExecuteScalar < Guid > ();
            return operatorId;
        }

        #endregion

    }

    #endregion
}
```

3. Связать интерфейс и код правила из справочника

Чтобы выполнить связывание реализации интерфейса и буквенного кода правила, указанного в справочнике на шаге 1, создайте класс `TestAppEventListener` (наследник класса `AppEventListenerBase`), в котором выполните связывание интерфейсов.

Чтобы **создать класс**:

1. Перейдите в раздел [*Конфигурация*] ([*Configuration*]) и выберите пользовательский пакет, в который будет добавлена схема.
2. На панели инструментов реестра раздела нажмите [*Добавить*] —> [*Исходный код*] ([*Add*] —> [*Source code*]).
3. В дизайнере схем заполните **свойства схемы**:
   - [*Код*] ([*Code*]) — "`UsrTestAppEventListener`".
   - [*Заголовок*] ([*Title*]) — "`TestAppEventListener`".



Для применения заданных свойств нажмите [*Применить*] ([*Apply*]).

4. В дизайнере схем добавьте исходный код.

TestAppEventListener

```
namespace Terrasoft.Configuration.Omnichannel.Messaging {
    using Common;
    using Core;
    using Terrasoft.Core.Factories;
    using Web.Common;

    #region Class: TestAppEventListener

    /// <summary>
    /// The class that runs prerequisites for OmnichannelMessaging on the application start.
    /// </summary>
    public class TestAppEventListener: AppEventListenerBase {

        #region Fields: Protected

        protected UserConnection UserConnection {
            get;
            private set;
        }

        #endregion

        #region Methods: Protected

        /// <summary>
        /// Retrieve the user connection from the application event scope.
```

```
        /// </summary>
        /// <param name="context">The application event scope.</param>
        /// <returns>User connection.</returns>
        protected UserConnection GetUserConnection(AppEventContext context) {
            var appConnection = context.Application["AppConnection"] as AppConnection;
            if (appConnection == null) {
                throw new ArgumentNullOrEmptyException("AppConnection");
            }
            return appConnection.SystemUserConnection;
        }

        protected void BindInterfaces() {
            ClassFactory.Bind < IOperatorRoutingRule,
            TestOperatorRoutingRule > ("TestRule");
        }

        #endregion

        #region Methods: Public

        /// <summary>
        /// Handle the application start.
        /// </summary>
        /// <param name="context">The application event scope.</param>
        public override void OnAppStart(AppEventContext context) {
            base.OnAppStart(context);
            UserConnection = GetUserConnection(context);
            BindInterfaces();
        }

        #endregion

    }

    #endregion

}
```

4. Перезапустите приложение в IIS

Для применения изменений перезапустите приложение в IIS. После перезапуска приложения и выбора в настройках очереди правила "`Test rule`", чаты будут распределяться по заданным условиям.