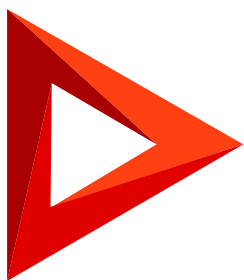


Чаты

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	4
Интеграция с мессенджером	4
Прием сообщений	4
Сервисы для работы с чатами в Creatio	5
Добавить новый провайдер канала	5
Добавить новый провайдер канала	5
1. Добавить новый провайдер в Creatio	6
2. Реализовать хранение данных нового канала	6
3. Зарегистрировать новый канал в базе данных	6
4. Создать веб-сервис для приема сообщений	7
5. Реализовать конвертацию входящего сообщения в универсальный формат Creatio	10
6. Реализовать получение данных профиля пользователя	12
7. Реализовать загрузку вложений	15
8. Реализовать отправку сообщений	17
9. Выполнить связывание интерфейсов	19
Добавить механизм маршрутизации чатов	21
1. Добавить новое правило маршрутизации чатов	22
2. Создать класс, который реализует интерфейс IOperatorRoutingRule	23
3. Связать интерфейс и код правила из справочника	26
4. Перезапустите приложение в IIS	28
Интеграция с каналами чатов	29
Интеграция с каналом Facebook Messenger	29
Интеграция с каналом WhatsApp	30
Интеграция с каналом Telegram	31

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 новый провайдер канала

 **Сложный**

Пример. В on-site приложении Creatio создать новый провайдер чата `Test`. В качестве зависимости пользовательского [пакета](#) указать пакет `OmnichannelMessaging`.

1. Добавить новый провайдер в Creatio

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 [*Провайдер канала*] ([*Channel provider*]).
4. Откройте наполнение справочника и добавьте значение .

2. Реализовать хранение данных нового канала

В базе данных приложения создайте таблицу [TestMsgSettings] (имя таблицы следует формировать по правилу [*<ИмяПровайдера>MsgSettings*]).

Структура таблицы зависит от конкретного мессенджера и должна содержать данные, необходимые системе для выполнения отправки и приема сообщений. Как правило мессенджеры для отправки используют авторизационный токен.

Пример скрипта, 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
```

Postgres

```
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 ''::character v
    "UserName" character varying(250) COLLATE pg_catalog."default" NOT NULL DEFAULT ''::characte
    CONSTRAINT "PK_TestMsgSettings_Id" PRIMARY KEY ("Id")
);
```

3. Зарегистрировать новый канал в базе данных

Для регистрации нового канала добавьте запись в таблицу `[Channel]`.

Описание полей таблицы `[Channel]`

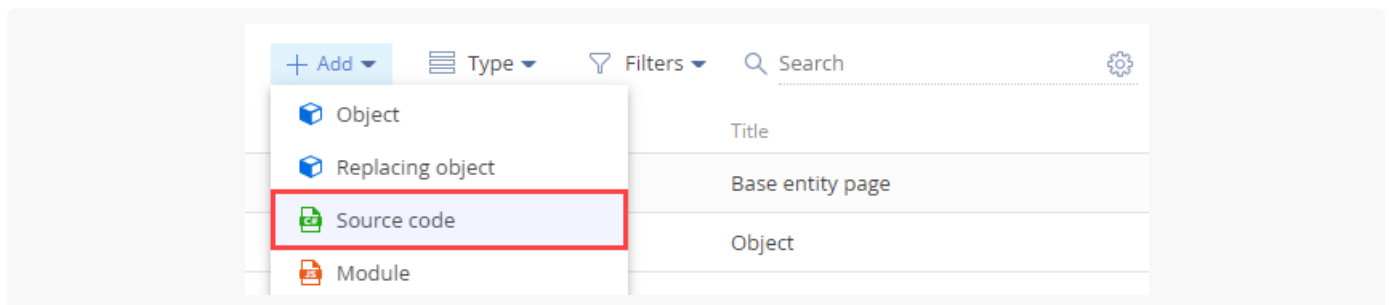
Поле	Описание
<code>[Name]</code>	Название канала.
<code>[ProviderId]</code>	Идентификатор добавленного провайдера.
<code>[MsgSettingsId]</code>	Идентификатор записи в таблице <code>[TestMsgSettings]</code> .
<code>[Source]</code>	Идентификатор канала внутри мессенджера, например идентификатор страницы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

4. Создать веб-сервис для приема сообщений

В примере реализуем вариант, при котором мессенджер присылает входящие сообщения на указанный endpoint. Созданный веб-сервис должен быть анонимным и доступным извне. Наследовать веб-сервис следует от базового веб-сервиса `OmnichannelMessagingService`, поскольку он содержит набор методов, позволяющих сохранить сообщение, создать чат, контакт и т. д.

Чтобы **создать веб-сервис**:

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 дизайнера схем заполните **свойства схемы**:

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

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

4. В дизайнера схем добавьте исходный код.

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

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

        #region Constructors: Public

        public TestOmnichannelMessagingService() : base() {
```



```

    }

    /// <summary>
    /// Initialize a new instance of <see cref="TestOmnichannelMessagingService"/>.
    /// </summary>
    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

    /// <summary>
    /// Receive messages from the integration API.
    /// </summary>
    /// <param name="message">Test provider message.</param>
    [OperationContract]
    [WebInvoke(UriTemplate = "receive", Method = "POST", RequestFormat = WebMessageFormat
        ResponseFormat = WebMessageFormat.Json)]
    public void ReceiveMessage(TestIncomingMessage message) {
        MessagingMessage messagingMessage = new MessagingMessage(TestIncomingMessageConve
        GetChannelAndQueueBySource(messagingMessage);
        InternalReceive(messagingMessage);
    }

    #endregion

}

#endregion

}

```

В примере мессенджер присылает входящее сообщение на endpoint `receive`.

В методе `ReceiveMessage` выполняется:

- Конвертация сообщения (метод `Convert` класса `TestIncomingMessageConverter`, который будет создан на следующем шаге).
- Идентификация канала по полю `Source` (метод `GetChannelAndQueueBySource`).
- Вызов метода `InternalReceive`, который выполнит следующие действия:
 - Создаст новый чат, если открытый чат с этим клиентом не будет найден, или добавит сообщение к существующему.
 - Создаст контакт, если это первое общение с клиентом, и выполнит заполнение его контактных данных, или привяжет чат к существующему контакту.

5. Сохраните и опубликуйте схему.

5. Реализовать конвертацию входящего сообщения в универсальный формат Creatio

После приема сообщения от мессенджера необходимо выполнить конвертацию входящего сообщения в универсальный формат Creatio (класс `MessagingMessage`). В примере эта задача выполняется классом `TestIncomingMessageConverter`, который конвертирует входящее сообщение с типом `TestIncomingMessage` (сообщение в формате мессенджера) в формат Creatio.

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 дизайнера схем заполните **свойства схемы**:
 - [*Код*] ([*Code*]) — "UsrTestIncomingMessageConverter".
 - [*Заголовок*] ([*Title*]) — "TestIncomingMessageConverter".

Source code

Code *
UsrTestIncomingMessageConverter

Title *
TestIncomingMessageConverter

Package
TestPackage

Description

CANCEL APPLY

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

4. В дизайнера схем добавьте исходный код.

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) {
        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

}

}

```

Блок `Attachments` заполняется в зависимости от формата доступа к файлам, который предоставляет мессенджер. В примере это ссылка для загрузки. Возможен вариант с передачей в сообщении `FileId` для последующего скачивания файла.

5. Сохраните и опубликуйте схему.

6. Реализовать получение данных профиля пользователя

Мессенджеры предоставляют API для получения данных клиентов, которые отправляют сообщения. Чтобы получить эти данные необходимо создать класс, реализующий 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 дизайнера схем заполните **свойства схемы**:
 - `[Код]` (`[Code]`) — `"UsrTestProfileDataProvider"`.
 - `[Заголовок]` (`[Title]`) — `"TestProfileDataProvider"`.

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

4. В дизайнера схем добавьте исходный код.

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

    /// <summary>
    /// Retrieve the profile data from the Test provider.
    /// </summary>
    public class TestProfileDataProvider : IProfileDataProvider
    {
        #region Properties: Private

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

```

#endregion

#region Constructors: Public

/// <summary>
/// Initialize a new instance of <see cref="FacebookProfileDataProvider"/>
/// </summary>
/// <param name="userConnection">User connection.</param>
public TestProfileDataProvider(UserConnection userConnection) {
}

#endregion

#region Methods: Public

/// <summary>
/// Retrieve the profile data from Facebook.
/// <param name="profileId">The Facebook profile ID.</param>
/// <param name="channelId">The channel from which to send the request.</param>
/// <returns>The contact ID.</returns>
/// </summary>
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>(
                        sr.ReadToEnd());
                    return new ProfileData {
                        FirstName = testProfile.first_name,
                        LastName = testProfile.last_name,
                    };
                }
            }
        }
    } catch {
        return new ProfileData();
    }
}

#endregion

}

#endregion
}

```

В примере класс отправляет запрос на URL `https://graph.test.com/` для получения данных в формате `TestProfileData` (предполагаемый формат мессенджера) и конвертирует полученный ответ во внутренний формат `ProfileData`. При создании контакта к нему будут добавлены полученные данные (например, имя, фамилия, фото). Если запрос не будет успешным или данные будут некорректными, то контакт будет создан с именем [*<Новый контакт><Имя канала>-<идентификатор клиента в мессенджере>*].

5. Сохраните и опубликуйте схему.

7. Реализовать загрузку вложений

Для загрузки вложений создайте класс, реализующий 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 дизайнера схем заполните **свойства схемы**:
 - [*Код*] ([*Code*]) — "UsrTestAttachmentLoadWorker".
 - [*Заголовок*] ([*Title*]) — "TestAttachmentLoadWorker".

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

4. В дизайнера схем добавьте исходный код.

TestAttachmentLoadWorker

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

```

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

    #region Class: TestAttachmentLoadWorker

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

        #region Properties: Protected

        protected UserConnection UserConnection;
        protected AttachmentsDownloader AttachmentsDownloader;

        #endregion

        #region Constructors: Public

        /// <summary>
        /// Initialize a new instance of the <see cref="TestAttachmentLoadWorker" /> class.
        /// <param name="userConnection">Instance of the <see cref="UserConnection"/> class.<
        /// </summary>
        public TestAttachmentLoadWorker(UserConnection userConnection) {
            UserConnection = userConnection;
            AttachmentsDownloader = new AttachmentsDownloader(userConnection);
        }

        #endregion

        #region Methods: Public

        /// <summary>
        /// Load the attachments.
        /// </summary>
        /// <param name="incomeAttachment">The attachment from the messenger.</param>
        /// <param name="message">The source message.</param>
        public void Load(MessageAttachment incomeAttachment, UnifiedMessage message) {
            incomeAttachment.FileName = FileUtilities.GetFileNameFromUrl(incomeAttachment.Up1
            incomeAttachment.FileId = AttachmentsDownloader.Load(incomeAttachment);
        }

        #endregion
    }
}

```



```
#endregion
}
```

Для загрузки вложений используется внутренний класс `AttachmentsDownloader`, который выполняет загрузку и сохранение файла по ссылке. Сохранение файла происходит в таблицу `[OmnichannelMessageFile]`.

5. Сохраните и опубликуйте схему.

8. Реализовать отправку сообщений

Для отправки сообщений **создайте класс**, реализующий 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 дизайнера схем заполните **свойства схемы**:
 - [*Код*] ([*Code*]) — "UsrTestOutcomeMessageWorker".
 - [*Заголовок*] ([*Title*]) — "TestOutcomeMessageWorker".

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

4. В дизайнера схем добавьте исходный код.

```
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

/// <summary>
/// The class that sends messages to the Test provider.
/// </summary>
public class TestOutcomeMessageWorker : IOOutcomeMessageWorker
{

    #region Properties: Protected

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

    #region Constructors: Public

    /// <summary>
    /// Initialize a new instance of the <see cref="TestOutcomeMessageWorker" /> class.
    /// <param name="userConnection">Instance of the <see cref="UserConnection"/> class.<
    /// </summary>
    public TestOutcomeMessageWorker(UserConnection userConnection) {
        UserConnection = userConnection;
    }

    #endregion

    #region Methods: Public

    /// <summary>
    /// Send the message to Test provider.
    /// </summary>
    /// <param name="message">The UnifiedMessage message.</param>
    public string SendMessage(UnifiedMessage unifiedMessage) {
        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;
        return result;
    }

}

```

```

        #endregion
    }

    #endregion
}

```

Класс `TestOutcomeMessageWorker` переводит сообщение в формат мессенджера и выполняет его отправку, используя API мессенджера. Для выполнения отправки может понадобится токен, который следует хранить в таблице `[TestMsgSettings]`. Доступ к таблице можно получить через переданный в конструкторе `UserConnection`. В примере выполняется отправка сообщения на URL `https://graph.test.com/`, используя внутренний класс `HttpRequestSender`.

5. Сохраните и опубликуйте схему.

9. Выполнить связывание интерфейсов

Чтобы выполнить связывание интерфейсов **создайте класс** наследник `AppEventListenerBase`:

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 дизайнера схем заполните **свойства схемы**:
 - `[Код]` (`[Code]`) — `"UsrTestAppEventListener"`.
 - `[Заголовок]` (`[Title]`) — `"TestAppEventListener"`.

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

4. В дизайнера схем добавьте исходный код.

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

    /// <summary>
    /// The class that runs prerequisites for OmnichannelMessaging on 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 ArgumentNullException("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

    /// <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

}

```

Созданные классы связываются по тегу "Test". Этот же тег должен быть указан при конвертации сообщения (шаг 4) в поле `ChannelName`. Таким образом система определяет какие файлы необходимо использовать для получения данных профиля, загрузки вложений и отправки сообщений.

5. Сохраните и опубликуйте схему.

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.

All agents Creatio

APPLY CANCEL

Name *
All agents

Routing rule *
To all agents


Chat completion timeout, minutes
2

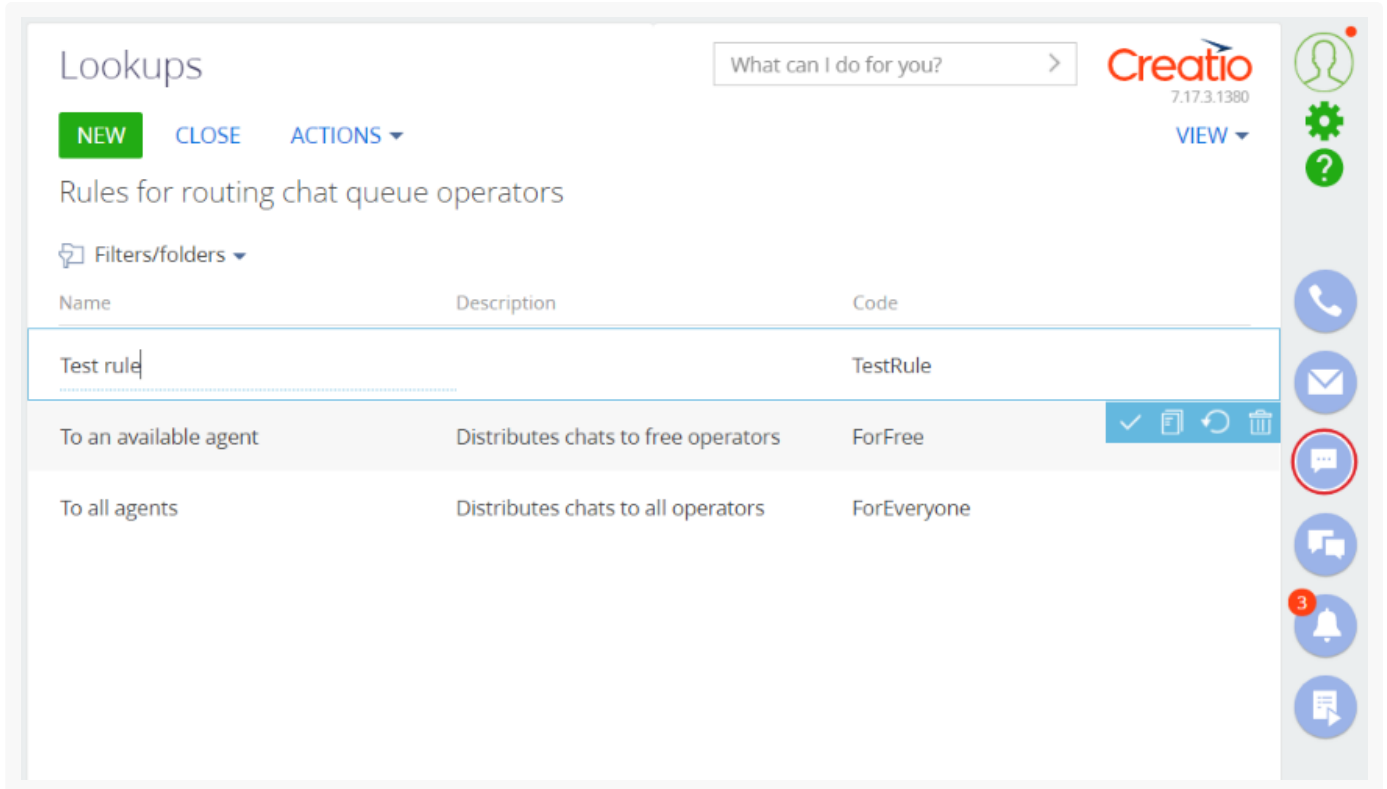
Supervisor

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`

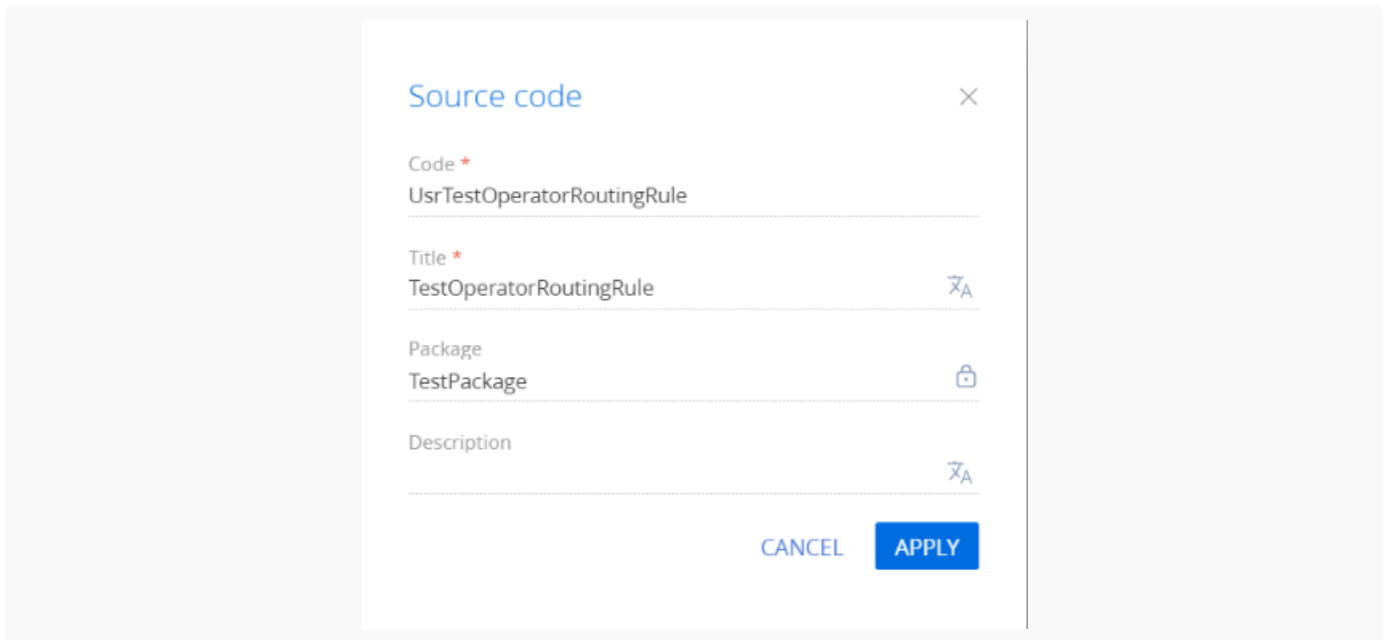
```
public List <Guid> GetOperatorIds(string chatId, Guid queueId) {
    var parsedChatId = Guid.Parse(chatId);
    var chatOperator = GetChatOperator(parsedChatId);
    return chatOperator.IsNotEmpty() ? new List <Guid> {
        chatOperator
    } : PickupFreeQueueOperators(parsedChatId, queueId);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

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 получают 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 дизайнера схем заполните **свойства схемы**:
 - `[Код]` (`[Code]`) — `"UsrTestOperatorRoutingRule"`.
 - `[Заголовок]` (`[Title]`) — `"TestOperatorRoutingRule"`.



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

4. В дизайнерах 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
}

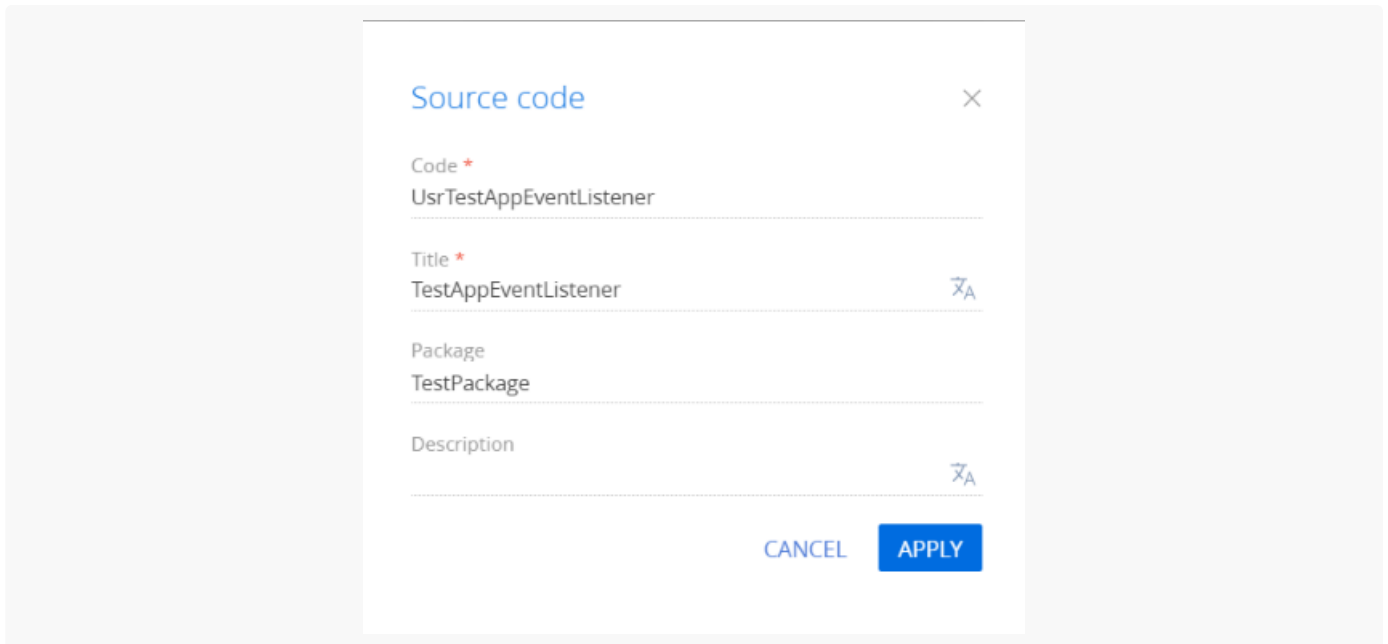
```

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 дизайнера схем заполните **свойства схемы**:
 - [*Код*] ([*Code*]) — "UsrTestAppEventListener".
 - [*Заголовок*] ([*Title*]) — "TestAppEventListener".



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

4. В дизайнера 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 ArgumentNullException("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.

Queue


CLOSE

Name *
New queue for Supervisor

Routing rule * Chat completion timeout, minutes

Test rule

To all agents

To an available agent

Интеграция с каналами чатов

 Средний

Назначение чатов — обработка операторами контакт-центра сообщений из популярных мессенджеров непосредственно в приложении Creatio. **Канал чата** в Creatio — это источник, из которого в приложение добавляются сообщения клиентов. Общий порядок действий по настройке обработки чатов описан в статье [Настроить обработку чатов](#).

Каналы чатов, интеграцию с которыми позволяет настроить Creatio:

- Facebook Messenger.
- WhatsApp.
- Telegram.

Интеграция с каналом Facebook Messenger

Этапы предварительной подготовки к настройке интеграции с каналом Facebook Messenger:

1. Проверить наличие доступа к облачным сервисам Creatio.
2. Проверить наличие доступа к сервисам Facebook.

Инструкция по настройке интеграции с Facebook Messenger содержится в статье [Настроить интеграцию с Facebook Messenger](#).

Взаимодействие со страницами Facebook выполняется от имени приложения Creatio Social, которое обращается к облачным сервисам. На уровне облачных сервисов выполняется:

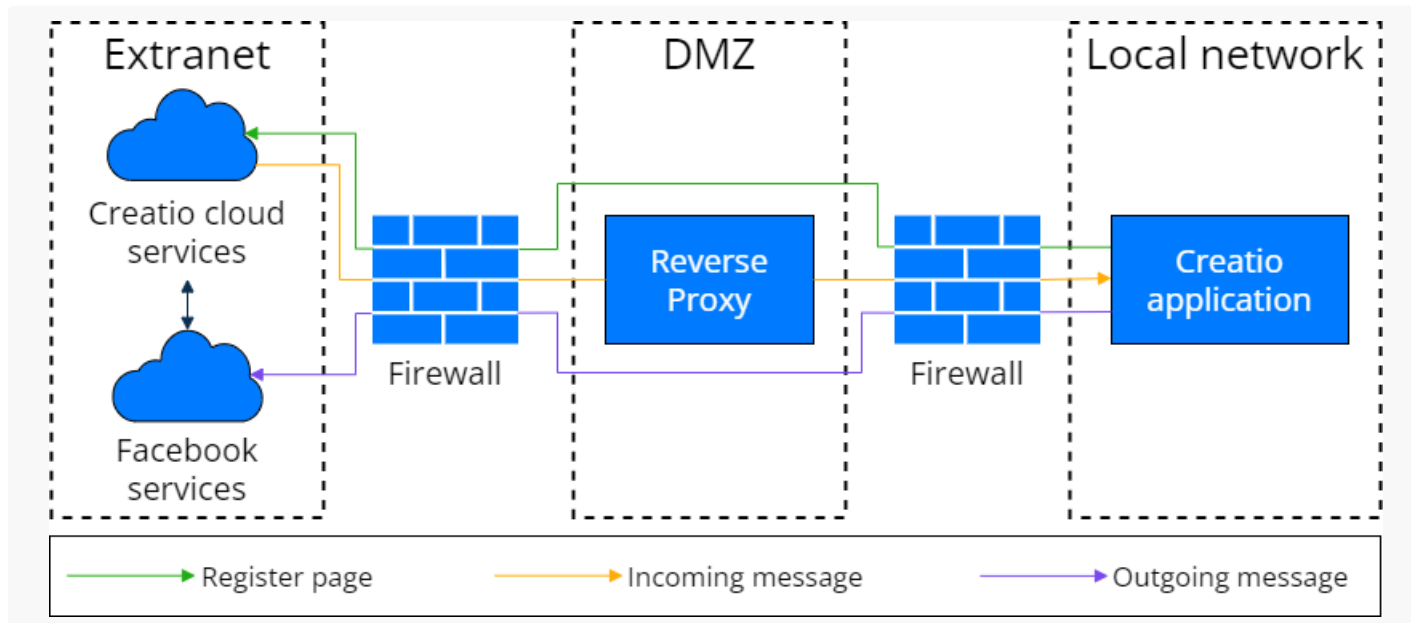
- Подписка на новые входящие сообщения страницы Facebook.
- Создание привязки "подписка-сайт Creatio".

После получения **входящих сообщений** на страницу Facebook облачные сервисы Creatio отправляют сообщение в клиентское приложение Creatio. Если сайт Creatio недоступен, то сообщение помещается в

очередь и будет отправлено повторно.

Отправка **исходящих сообщений** выполняется напрямую из приложения Creatio без использования облачных сервисов.

Схема взаимодействия on-site приложения Creatio с каналом Facebook Messenger представлена на рисунке ниже.



Интеграция с каналом WhatsApp

Возможность интеграции с каналом WhatsApp доступна для приложений Creatio версии 7.18.0 и выше.

Этапы предварительной подготовки к настройке интеграции с каналом WhatsApp:

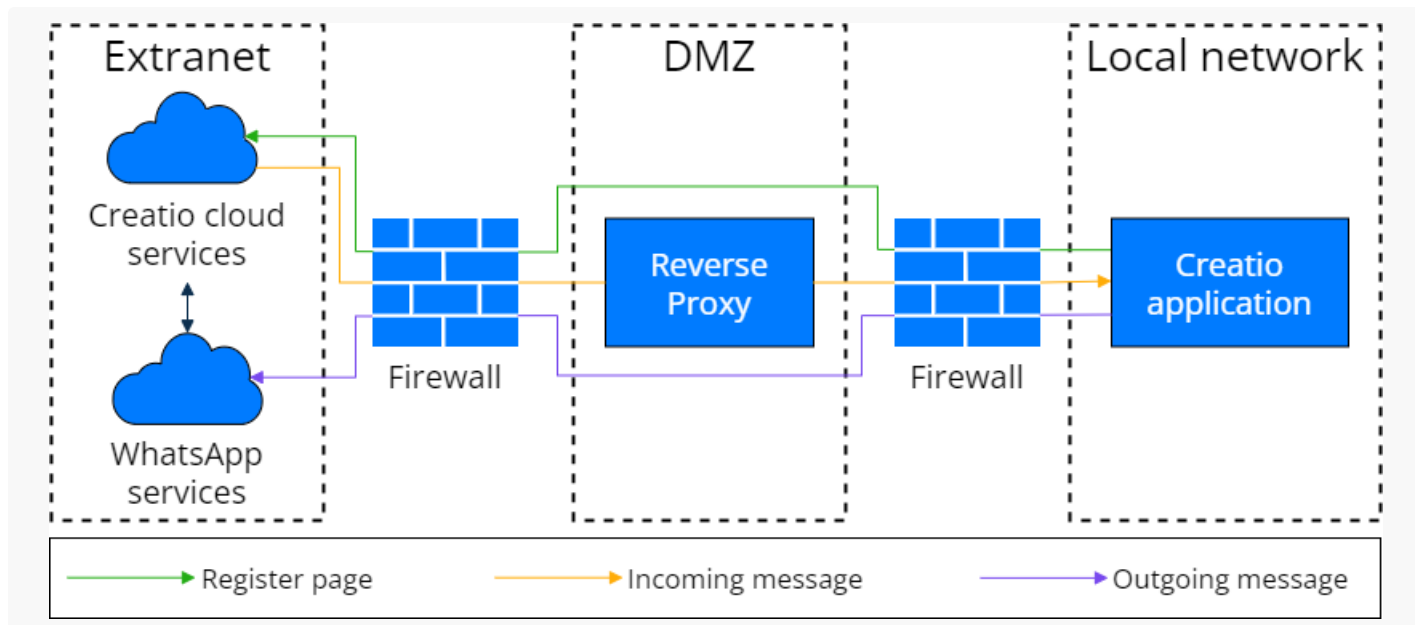
1. Проверить наличие доступа к облачным сервисам Creatio.
2. Создать учетную запись Twilio.
3. Приобрести номер телефона Twilio для получения входящих и отправки исходящих сообщений.

Инструкция по настройке интеграции с WhatsApp содержится в статье [Настроить интеграцию с WhatsApp](#).

После получения **входящих сообщений** на номер телефона Twilio облачные сервисы Creatio отправляют сообщение в клиентское приложение Creatio. Если сайт Creatio недоступен, то сообщение помещается в очередь и будет отправлено повторно.

Отправка **исходящих сообщений** выполняется напрямую из приложения Creatio без использования облачных сервисов.

Схема взаимодействия приложения Creatio, которое развернуто on-site, с каналом WhatsApp представлена на рисунке ниже.



Интеграция с каналом Telegram

Этапы предварительной подготовки к настройке интеграции с каналом Telegram:

1. Создать API для каждого чат-бота Telegram.
2. Указать созданное API в настройках приложения Creatio.

Для интеграции с Telegram промежуточные сервисы не используются. Инструкция по настройке интеграции с Telegram содержится в статье [Настроить интеграцию с Telegram](#).

Для получения **входящих сообщений** приложение Creatio использует [Long pooling технологию](#). Чтобы проверить наличие новых сообщений, приложение периодически отправляет запрос к Telegram API. Шаблон строки запроса к Telegram API представлен ниже.

Шаблон строки запроса к Telegram API

```
https://api.telegram.org/bot{token}/getUpdates
```

`token` — токен, который использовался при регистрации канала.

Отправка **исходящих сообщений** выполняется напрямую на адреса сервисов Telegram.

Схема взаимодействия приложения Creatio, которое развернуто on-site, с каналом Telegram представлена на рисунке ниже.

