EngageOne Connect

Development Guide for Non Delivery Actions

NDR Action Plugin – Development Guide

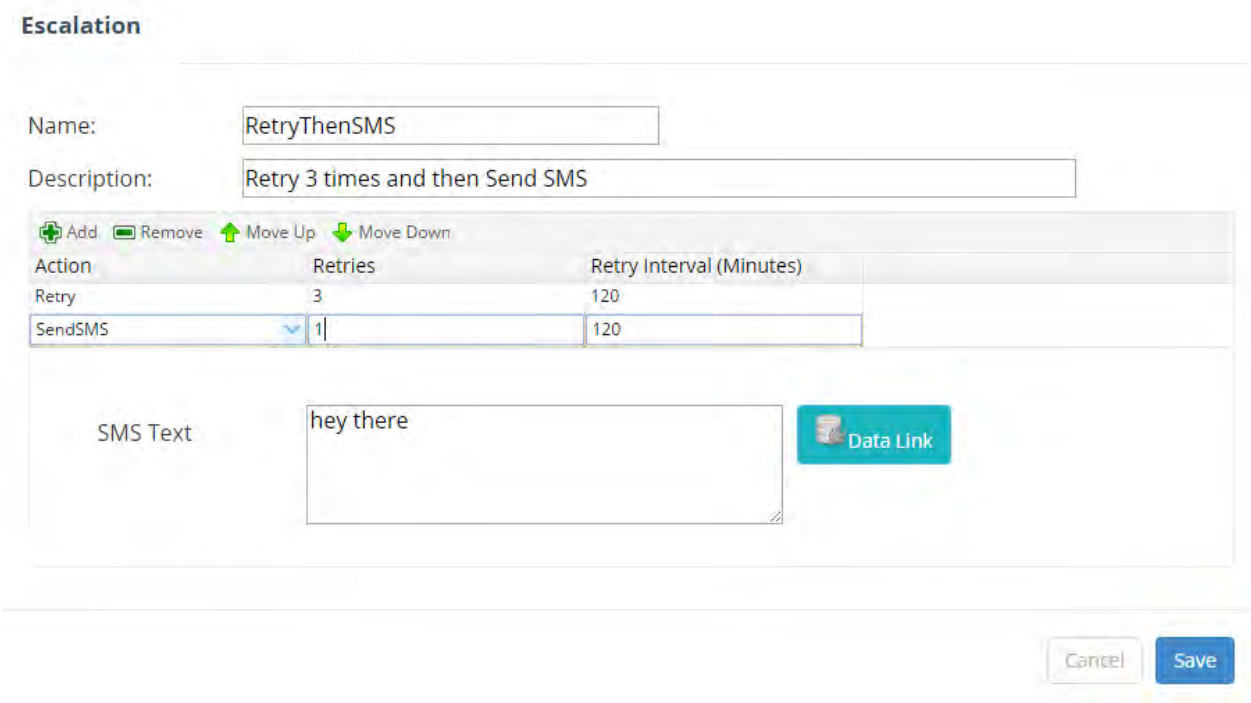
NDR Escalations Overview

The purpose of the NDR Escalation mechanism is to allow the system to automatically respond to a delivery failure.

Functionality

The system comes with a set of pre-built actions (each represented by an NDR Action Plugin – more on that later).

The user defines escalations, where a single escalation consists of a list of NDR Actions and their parameters.



Once defined, the user can attach an escalation to a group of NDR codes, meaning, if this NDR code is returned, the escalation will be executed and will run the first action. If an NDR (bounce back) happens again, the second action will be executed and so forth.

In the example above, the first 3 times the NDR will occur, the email will be retried. If the NDR happened a fourth time, an SMS with the "Hey there" text will be sent instead. If the NDR happened a fifth time, nothing will happen.

The user defines the default mapping between NDR Codes and escalations, and then can modify this configuration for a specific campaign.

For a more comprehensive guide how to define and use NDR Escalations and Actions please refer to our main User's Guide.

NDR Action Plugin

Technical Implementation

The Incoming Data Processing service is responsible for handling escalation logic.

In it, there's the *APIncomingDataProcessing.Common.Schedulers.ExecuterScheduler* class, which is responsible for basic escalation logic and activating the relevant plugins.

It consumes data from two tables in the database -

- *FailedMessagesForNDRHandling* - this is basically a queue of ndr failures. The scheduler periodically reads from this table, handles whatever ndrs it finds, and then deletes it.
- *NDREscalationsForCampaign* – This table stores the relevant escalation configuration for each correlation. The data is copied here from *NDREscalations* and *NDREscalationActions* tables whenever a new campaign correlation is created.

As part of its execution, the scheduler stores data in the *NDREscalationState* table, which stores the last action that was executed for each user (identified by an *AccessId*), so that it knows the next action to execute when a new NDR arrives for each user.

<u>The basic logic of the scheduler:</u>

For each new NDR

Read the relevant state (by access id)

Read the relevant configuration (by correlation id)

If a relevant escalation exists for NDR code

If state exists

Select next relevant action in escalation

Else

Select first action in escalation

Group all selected actions by type, and execute each action once, passing in a list of all relevant NDRs

Save new State for each NSR

Delete all NDRs from database

Implementing a new plugin

Two things need to be done, to integrate a new plugin into the system:

1. Define a class that implements the *Columba.DeliverySDK.CollectorsData.IExecuter* interface. Alternatively, one can extend the *APIncomingDataProcessing.BuiltinPlugins.Common.AlternateSendMetadataPlugin* class, which handles some of the logic common to many plugins, which is to send something to the same user through the delivery engine. Both the *SendSMS* and the *Retry* plugin inherit from this class. The assembly containing the implemented class should be placed under {incoming-data-processing-service-install-dir}\CollectorsPlugins for the service to find it.
2. Add a record into the *Actions* table. The *PluginName* column must have the name of the class.

Sample Code – Send SMS NDR Action

```
class SendSMS : AlternateSendMetadataPlugin
  {
    protected override ReadyToSend CreateRTS(Guid correlationId, MessageToEscalate message)
    {
      return new ReadyToSend
      {
        CorrelationId = correlationId,
        LastModified = DateTimeOffset.UtcNow,
        ToUserId = message.Message.SES_USER_UNIQUE_ID,
        BodySMS = CreateSMSText(message),
        AccessId = message.Message.AccessId.ToString(),
        DeliveryCounter = 0,
        Status = 0,
```

```
            PhoneNumber = message.Message.Phone
        };
    }

    protected override short MessageType
    {
        get { return 5; }
    }

    private byte[] CreateSMSText(MessageToEscalate message)
    {
        var link = string.IsNullOrEmpty(message.Message.icpACCESS_ID) ? "" : string.Format("\n\n{0}",
Uri.EscapeUriString(message.Message.ICPlink.Replace("${accessId}", message.Message.icpACCESS_ID)));
        var messageText = string.Format("{0}{1}", message.Params.Value<string>("body"), link);
        return CompressHelper.EncryptAndCompress(Encoding.UTF8.GetBytes(messageText));
    }
}
```

If you want to send some message through delivery engine (to the same user the NDR comes from) you need to derive from the abstract class AlternateSendMetadataPlugin and override CreateRTS which creates readyToSend item (the message itself - in this case SMS)

In any case of some other functionality you need to implement IExecuter (AlternateSendMetadataPlugin derives from AlternateSendPlugin which implement IExecuter so this code example does not cover it) and implement ExecuteAction method to do whatever you want.