

Carousel Email API Developer's Guide

by Noesis Srl
<http://www.noesis-research.com>

Version 1.0



CAROUSEL

Table of Contents

<i>Carousel e-mail API Developer's Guide</i>	2
Technical Notes	2
Configuration	2
Creating the profiles	2
Campaign Configuration	3
E-mail message Creation	7
Creating constraints	9
Bid creation	11
Checking the accuracy of the configuration	13
Optimization Process	14
The GetPlanningRules – SetFeedback loop	14
The SetFeedback method	16

Carousel e-mail API Developer's Guide

This document describes how to use the Carousel e-mail API to programmatically access the Carousel Optimization Engine through a web service (Carousel Service).

Technical Notes

LANGUAGE: The Carousel e-mail API *Web Service* interface supports a growing number of languages - including Java, .NET, Perl, PHP, Python, OCAML, Ruby and XML. The sample code provided in this guide is in C#. The Java version is also provided at the end of the document. You can easily translate the code in any language of your choice.

HTTP/SOAP ERRORS: It must be stressed that all the procedures described below are concerned with the calls to a *Web Service*, and so they must always be encapsulated within a *try-catch* block which can manage any exceptions generated both by possible communications problems and by parameters being passed on incorrectly.

Configuration

To be able to use the service there is a preliminary configuration stage to create the various entities which describe the activity of the user. These will be the basis upon which the rules for the visualization of the e-mail message will be created.

Creating the profiles



Using the method

AddProfile(string userName, string password, Profile profile)

it is possible to add various *Profiles*. It will be the user's job to create the object that it describes by specifying a name (*Name*), a category (*Category*) and a tag list (*Tags*).

The category defines a similarity group: the profiles which are a part of the same category should have similar characteristics and defining a

series of categories/groups coherently makes it possible to propagate knowledge between them. *Profiles* with the same category share information.

The field *Tags* is not compulsory and covers the tag list which may be necessary to qualify the profile.

The field *Name* is mandatory while the field *Category* not; the field *Id* is the identification assigned automatically by the application. Whenever one is specified by the user for the profile, this will in any case be ignored. The user can retrieve a given profile at any time or consult the list of all existing profiles, deciding to cancel or update them.

Example of a code – creating a new profile

```
// This code sample creates a new profile.
public void AddProfile()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

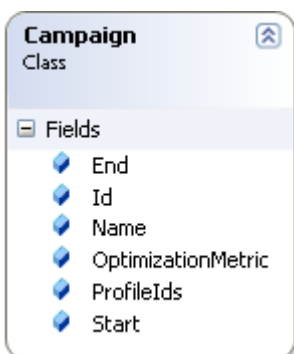
    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    // Create new profile structure
    Profile newProfile = new Profile();
    newProfile.Name = "NewProfile";
    newProfile.Category = "Category_for_Profile";
    // this field is not mandatory
    newProfile.Tags = new ArrayOfString { "tag1", "tag2", "tag3" }

    try
    {
        // Add the new profile if there are no error
        Profile addedProfile = service.AddProfile(user, password,
                                                    newProfile);

        // Display new profile
        Console.WriteLine("New profile of " + addedProfile.Category +
                           " category and id " + addedProfile.Id +
                           " was created.");
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine("New profile was not created due to the following
                           error: " + ex);
    }
}
```

Campaign Configuration



Each user can run a campaign made up of one or more *E-mail message*. Using the method

AddCampaign(string userName, string password, Campaign campaign)

it is possible to add various *Campaigns*. It will be the user's job to create the object that it describes by specifying a name (*Name*), a start and an end (*Start, End*), a list of the IDs of the profiles that the campaign will work with (*ProfileIds*) and the type of optimization metric that will be used (*OptimizationMetric*).

The field *Start* is compulsory, whereas *End* is not; the end date "31/12/2050" has been entered as the default setting, in effect indicating that the campaign is infinite.

The field *ProfileIds* is compulsory. A campaign without particular profiles defined can't exist. If they are defined, the profiles listed must exist and have been previously created. It should be noted that the same profile can be affected by different campaigns. The field *OptimizationMetric* is not

compulsory; it indicates what the objective function of the algorithm will be based on, and can have one of the following values:

Clicks = 0,

Revenue = 1

The default setting is 1.

The field *Name* is compulsory.

The field *Id* is the identification assigned automatically by the application. Whenever one is specified by the user for the campaign, this will in any case be ignored. The user can retrieve a given campaign at any time or consult the list of all existing campaigns, deciding to cancel or update them; it is also possible to know the current state of a campaign, using the following code system:

Initialized = 0 *if the campaign has already been created but not started yet*

Paused = 1 *if the campaign has been temporarily paused*

Started = 2 *if the campaign is running*

Stopped = 3 *if the campaign has finished.*

Example of a code – creating a new campaign

```
// This code sample creates a campaign with a specified duration, for a set of
// profiles. To create profiles see the procedure AddProfile.
// To create e add e-mail message to campaign see the procedure AddEmailMessage.
public void AddCampaign()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    // Create new campaign structure
    Campaign newCampaign = new Campaign();
    newCampaign.Name = "NewCampaign";
    newCampaign.Start = DateTime.Now;
    // end time must not be specified if the campaign is infinite
    newCampaign.End = DateTime.Now.AddDays(3);

    // this field has a value chosen from above:
    // 0 if we want to optimize the number of click,
    // 1 if we want to optimize the revenue
    newCampaign.OptimizationMetric = 0;

    // profile ids must be exist
    // this field is mandatory
    newCampaign.ProfileIds = new ArrayOfInt() { 1, 2, 3 };

    try
    {
        // Add the new campaign if there are no error
        Campaign addedCampaign = service.AddCampaign(user, password,
                                                    newCampaign);

        // Display new campaign
        Console.WriteLine("New campaign with name " + addedCampaign.Name +
```

```

        " and id " + addedCampaign.Id + " was created.");
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine("New campaign was not created due to the following
            error: " + ex);
    }
}

```

Example of a code – retrieving the list of existing campaigns

```

// This code sample retrieves information about all campaigns that belongs to
the customer issuing the request.
public void GetAllCampaigns()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    try
    {
        // Get all campaigns
        Campaign[] campaigns = service.GetCampaigns(user, password);

        // Display campaign info
        foreach (Campaign c in campaigns)
        {
            Console.WriteLine("Campaign name is " + c.Name +
                " and id is " + c.Id);
        }
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine("Campaigns info no retrieved due to the following
            error: " + ex);
    }
}

```

Example of a code – updating a campaign

```

// This code sample change a campaign updating only the field specified and not
null.
public void UpdateCampaign()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    // Create the updated campaign structure

```

```

// In this campaign only the name changes,
//all the other field are set to null
Campaign campaign1 = new Campaign();
campaign1.Id = 3; // Campaign with ID=3 must exist
campaign1.Name = "UpdatedName";
campaign1.Start = null;
campaign1.End = null;
campaign1.OptimizationMetric = null;
campaign1.ProfileIds = null;

// In this campaign start date and optimization metric are changed
Campaign campaign2 = new Campaign();
campaign2.Id = 7; // Campaign with ID=7 must exist
campaign2.Name = null;
campaign2.Start = DateTime.Now.AddDays(5);
campaign2.End = null;
campaign2.OptimizationMetric = 1;
campaign2.ProfileIds = null;

// In this campaign profile list are changed
Campaign campaign3 = new Campaign();
campaign3.Id = 9; // Campaign with ID=9 must exist
campaign3.Name = null;
campaign3.Start = null;
campaign3.End = null;
campaign3.OptimizationMetric = null;
campaign3.ProfileIds = new ArrayOfInt(){ 1, 2 };

try
{
    Campaign updateCampaign1 = service.UpdateCampaign(user, password,
                                                    campaign1);
    Campaign updateCampaign2 = service.UpdateCampaign(user, password,
                                                    campaign2);
    Campaign updateCampaign3 = service.UpdateCampaign(user, password,
                                                    campaign3);

    // Display result
    Console.WriteLine("Campaigns were updated.");
}
catch (ArgumentException ex)
{
    Console.WriteLine("Campaigns was not updated due to the following
                    error: " + ex);
}
}

```

Example of a code – checking and modifying the state of a campaign

```

// This code sample retrieves information about the current status of a
// specified campaign, resume a campaign and change status of another one pausing
// it.
public void CampaignStatus()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";
}

```

```

// campaigns must exist
int campaignId1 = 1;
int campaignId2 = 2;
int campaignId3 = 3;

try
{
    // Get campaign status
    int status1 = service.GetCampaignStatus(user,password,campaignId1);

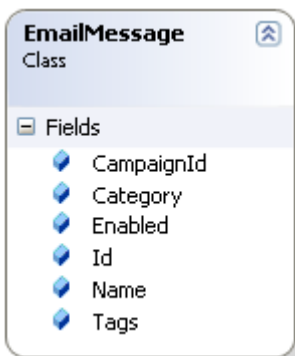
    // Pause campaign 2
    int status2 = service.PauseCampaign(user, password, campaignId2);

    // Resume campaign 3
    int status3 = service.PauseCampaign(user, password, campaignId3);

    // Display campaign status
    Console.WriteLine("Campaign with Id: "+campaignId1+" is "+status1);
    Console.WriteLine("Campaign with Id: "+campaignId2+" is "+status2);
    Console.WriteLine("Campaign with Id: "+campaignId3+" is "+status3);
}
catch (ArgumentException ex)
{
    Console.WriteLine("Campaign status no retrieved due to the following
        error: " + ex);
}
}

```

E-mail message Creation



Every campaign is made up of one or more *e-mail message*. Using the method

AddEmailMessage(string userName, string password, int campaignId, EmailMessage emailMessage)

it is possible to add an *e-mail message* to a given campaign identified by means of parameter. It will be the user's job to create the object which will describe them, specifying a name (*Name*), a category (*Category*) and a tag list (*Tags*).

The user can also decide if the message created is enabled or not, setting a boolean value for the field *Enabled*; note that only the messages enabled are used by the algorithm to create the rules. The default value for *Enabled* is TRUE.

As for the *Profiles*, the category identifies a similarity group: messages which are a part of the same category should have similar characteristics and defining a series of categories/groups coherently makes it possible to propagate knowledge between them. *E-mail message* with the same category share information.

The field *Tags* is not compulsory and represents any tag list which may be necessary for an *e-mail message*.

The field *Name* is mandatory while the field *Category* not; *CampaignId* is the ID of the campaign that the ad belongs to.

The field *Id* is the ID assigned automatically by the application and whenever one is specified by the user for the message, this will in any case be ignored. The user can retrieve a given *e-mail message* at any time or consult the list of all existing messages, or only those for a single campaign, deciding to cancel or update them.

Example of a code – creating a new *e-mail message*

```
// This code sample creates a new email message for a campaign.
public static void AddEmailMessage()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    int campaignId = 1; // campaign with ID=1 must exist

    // Create new email message structure
    EmailMessage newEmailMessage = new EmailMessage();
    newEmailMessage.Name = "NewEmailMessage";
    newEmailMessage.Category = "Category_for_EmailMessage";
    newEmailMessage.Enabled = TRUE;
    newEmailMessage.Tags = new ArrayOfString { "tag1", "tag2", "tag3" };
    // this field is not mandatory

    try
    {
        // Add the new email message if there are no error
        EmailMessage addedEmailMessage = service.AddEmailMessage(user,
            password, campaignId, newEmailMessage);

        // Display new email message
        Console.WriteLine("New EmailMessage of " +
            addedEmailMessage.Category + " category and id " +
            addedEmailMessage.Id + " was created for the
            campaign " + addedEmailMessage.CampaignId);
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine("New EmailMessage was not created due to the
            following error: " + ex);
    }
}
```

Example of a code – retrieving the list of e-mail messages for a given campaign

```
// This code sample retrieves information about EmailMessages of a particular
campaign.
public static void GetEmailMessagesByCampaign()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    int campaignId = 1; // campaign with ID=1 must be exist

    try
    {
```

```

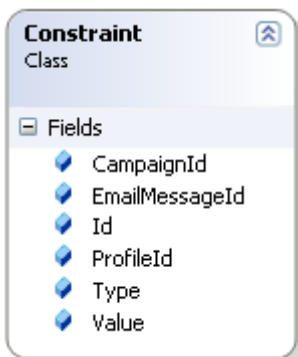
// Get EmailMessages
EmailMessage[] EmailMessages =
    service.GetEmailMessagesByCampaign(user, password, campaignId);

// Display EmailMessages info
Console.WriteLine("EmailMessages for campaign with Id: " +
    campaignId + " are ");
foreach (EmailMessage a in EmailMessages)
{
    Console.WriteLine("EmailMessage of " + a.Category +
        " category and Id " + a.Id);
}
}
catch (ArgumentException ex)
{
    Console.WriteLine("EmailMessages info no retrieved due to the
        following error: " + ex);
}
}

```

When the various entities necessary to describe the user’s activity have also been defined, the final step is to define the additional characteristics which allow us to specify a series of details for carrying out the campaign. At this point, therefore, we will move on defining the constraints and the bids.

Creating constraints



It is possible to identify a series of constraints which can be applied to one e-mail message or to an entire campaign, either a given profile, or also a combination of more of these.

Using the method

AddConstraint(string userName, string password, Constraint constraint)

it is possible to add a *Constraint*; it is up to the user to create the object which it describes, specifying the entity which it is concerned with, which aspect it will concern (*Type*) and the value (*Value*).

The fields that identify the entity (*EmailMessageId*, *CampaignId* and *ProfileId*) can be filled in either in full or in part according to how restrictive the constraint needs to be. For example, if a constraint is required that always applies for an entire campaign, then only the field *CampaignId* needs to be specified. If, on the contrary, a constraint is required which is in effect for only a certain e-mail message, then only the field *EmailMessageId* need to be completed. The fields *Value* and *Type* are compulsory. Based on what the constraint is to affect, a different option in the field *Type* needs to be specified, in particular:

Clicks = 0 *if it concerns the number of clicks*
Impressions = 1 *if it concerns the number of impressions*
Cost = 2 *if it concerns the cost*

We can consider the *Constraints* as the relationships “<=” on the aspect that we wish to constrain; for example, to express that “we do not want to obtain more than 100,000 impressions for message #3” we would need to define an object like:

```

Constraint constraint = new Constraint();
    constraint.EmailMessageId = 3;
    constraint.CampaignId = null;
    constraint.ProfileId = null;
    constraint.Type = 1;
    constraint.Value = 100000;

```

When the fixed threshold has been reached, message #3 will no longer be displayed and will not be in competition with the others in the other campaigns which are still active. The same can be done for all the other constrainable entities, for example, a campaign which has reached one of the fixed thresholds on which it will no longer be active and the relevant messages will not be displayed.

CampaignId is the ID of the campaign that the ad belongs to.

The field *Id* is the ID assigned automatically by the application and whenever one is specified by the user for the message, this will in any case be ignored.

The user can retrieve a given *Constraint* at any time or consult the list of all existing *Constraints*, or only those for a single campaign, deciding to cancel or update them.

Example of a code – creating a new constraint

```

// This code sample create some constraints.
// Note that all the fields are not mandatory.
public void AddConstraints()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    // Create new Constraints structure
    // the emailMessage, campaign and profile indicated must exist

    // The user specify all the fields to indicate the particular case and
    // entities for constraint.
    Constraint newConstraint1 = new Constraint();
    newConstraint1.EmailMessageId = 2;
    newConstraint1.CampaignId = 5;
    newConstraint1.ProfileId = 3;

    // this field has a value chosen from above:
    // 0 if we want to add a constraint over the number of click,
    // 1 if we want to add a constraint over the number of impression,
    // 2 if we want to add a constraint over the cost
    newConstraint1.Type = 0;
    newConstraint1.Value = Convert.ToDouble("INSERT_VALUE_HERE");

    // This constraint is valid wherever for all the messages of the campaign
    // with ID=10 of profile with ID=5
    Constraint newConstraint2 = new Constraint();
    newConstraint2.EmailMessageId = null;
    newConstraint2.CampaignId = 10;
    newConstraint2.ProfileId = 5;
    newConstraint2.Type = 1;
    newConstraint2.Value = Convert.ToDouble("INSERT_VALUE_HERE");

```

```

// This constraint is valid for all the messages of Campaign with ID=4
Constraint newConstraint3 = new Constraint();
newConstraint3.EmailMessageId = null;
newConstraint3.CampaignId = 4;
newConstraint3.ProfileId = null;
newConstraint3.Type = 0;
newConstraint3.Value = Convert.ToDouble("INSERT_VALUE_HERE");

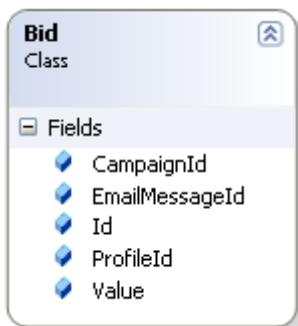
// This constraint is valid for all the messages of the campaign with ID=3
Constraint newConstraint4 = new Constraint();
newConstraint4.EmailMessageId = null;
newConstraint4.CampaignId = 3;
newConstraint4.ProfileId = null;
newConstraint4.Type = 2;
newConstraint4.Value = Convert.ToDouble("INSERT_VALUE_HERE");

try
{
    // Add new constraints if there are no error
    service.AddConstraint(user, password, newConstraint1);
    service.AddConstraint(user, password, newConstraint2);
    service.AddConstraint(user, password, newConstraint3);
    service.AddConstraint(user, password, newConstraint4);

    // Display new bid
    Console.WriteLine("New constraints were created.");
}
catch (ArgumentException ex)
{
    Console.WriteLine("New constraints were not created due to the
        following error: " + ex);
}
}

```

Bid creation



It is possible to define the cost for the user when a certain action goes into effect, particularly when a *e-mail message* is shown in a certain *campaign* for a specific *profile*.

As in the case of the *Constraints*, a *Bid* can be concerned with a specific *e-mail message* or an entire *campaign*, either a particular *profile*, or again a combination of more of these entities.

Using the method

AddBid(string userName, string password, Bid bid)

it is possible to add a *Bid*. It will be the user's job to create the object that describes it specifying the relevant entities and the value (*Value*).

The fields that identify the entities (*EmailMessageId*, *CampaignId* and *ProfileId*) can all be completed or less of them according to the extent of restriction required for the bid. For example, if a bid is required that always applies for an entire campaign, then only the field *CampaignId* needs to be specified. If, on the contrary, a bid is required which is in effect for only a certain *e-mail message*, then only the field *EmailMessageId* need to be completed.

The field *Id* is the ID assigned automatically by the application. Whenever one is specified by the user for the bid, this will in any case be ignored.

The user can retrieve a given *Bid* at any time or consult the list of all existing *Bids*, or only those for a single campaign, deciding to cancel or update them.

Example of a code – creating new bids

```
// This code sample create some bid.
// Note that not all the fields are indicate, in this case the bid specify the
// value for a particular e-mail message, campaign or profile.
public static void AddBids()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    // Create new Bid structure
    // the EmailMessageId, campaign and profile indicated must exist

    // This bid is valid wherever for all the email message of the campaign
    // with ID=10 of profile with ID=5
    Bid newBid1 = new Bid();
    newBid1.EmailMessageId = null;
    newBid1.CampaignId = 10;
    newBid1.ProfileId = 5;
    newBid1.Value = Convert.ToDouble("INSERT_VALUE_HERE");

    // This bid is valid for all the e-mail messages of the user
    Bid newBid2 = new Bid();
    newBid2.EmailMessageId = null;
    newBid2.CampaignId = null;
    newBid2.ProfileId = null;
    newBid2.Value = Convert.ToDouble("INSERT_VALUE_HERE");

    // This bid is valid for all e-mail messages of the campaign with ID=3
    Bid newBid3 = new Bid();
    newBid3.EmailMessageId = null;
    newBid3.CampaignId = 3;
    newBid3.ProfileId = null;
    newBid3.Value = Convert.ToDouble("INSERT_VALUE_HERE");

    // This bid is valid for the e-mail message with ID=1 of the prifile with
    // ID=5 and of the campaign with ID=10
    Bid newBid = new Bid();
    newBid.EmailMessageId = 1;
    newBid.CampaignId = 10;
    newBid.ProfileId = 5;
    newBid.Value = Convert.ToDouble("INSERT_VALUE_HERE");

    try
    {
        // Add new bids if there are no error
        service.AddBid(user, password, newBid1);
        service.AddBid(user, password, newBid2);
        service.AddBid(user, password, newBid3);
        service.AddBid(user, password, newBid);

        // Display new bid
        Console.WriteLine("New bids were created.");
    }
}
```

```

    }
    catch (ArgumentException ex)
    {
        Console.WriteLine("New bids were not created due to the following
                           error: " + ex);
    }
}

```

Checking the accuracy of the configuration

There are always the main checks for the accuracy of data inputted by the user; the most typical one is on whether something referred to by only its ID actually exists, for example, when not all *Profiles* have a category. At the end of configuration it is in any case a good idea to test the accuracy of the operations so far; in particular, by using the method

CheckConfiguration(string userName, string password)

it is possible to check the accuracy of some aspects.

The method carries out the tests and returns any applicable error messages; all messages are in the form of a string, and the fields are separated by “;”. The fields indicated are:

#	internalErrorNumber
tag	a tag describing the type of error: "warning", "error" or "info"
table	table name
field	field name
idsList	list of the IDs on record that have caused the error (a string of Ids separated by
commas)	
message	message

Example of an error message

```
105;error;Profiles;Category;10,20;Error in profiles
```

Example of a code – checking the configuration

```

// This code sample check the accuracy of the configuration done.
public void CheckConfiguration()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    try
    {
        ArrayOfString checks = service.CheckConfiguration(user, password);

        if (checks.Count == 0)
            Console.WriteLine("No error in configuration");
    }
}

```

```

else
{
    Console.WriteLine("Errors in configuration: ");
    foreach (string c in checks)
        Console.WriteLine(c);
}
}
catch (ArgumentException ex)
{
    Console.WriteLine("Check not done due to the following error: "+ex);
}
}

```

Once the entire system has been tested and the accuracy of it has been verified, it will be possible to start using the service by means of periodic calls which allow the user to retrieve the rules for the display of e-mail message and for saving feedback data.

Optimization Process

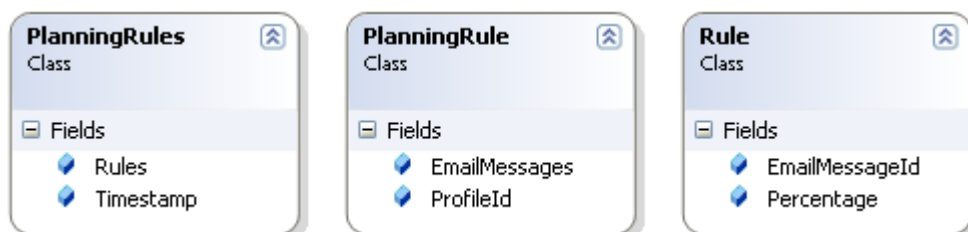
After campaigns have been configured, the E-Mail Server can begin a permanent communication process with Carousel Service, to receive the instructions to optimize the campaign in real time. As explained in the introduction, this communication process is a sequence of calls alternate to the methods *GetPlanningRules* and *SetFeedback*.

Carousel generates all the associations between profile and best e-mail message in terms of profitability. The algorithm takes all the e-mail messages in all the active campaigns which have been previously defined by the user into consideration.

The service identifies and returns the percentage for each available e-mail message that it will be shown.

The GetPlanningRules – SetFeedback loop

Carousel identifies the percentage chance for each available e-mail message that it will be shown in a given *situation (profile)*. Every so often the user requests these rules and the E-MailServer satisfies the requests on the different pages.



Using the method

GetPlanningRules(string userName, string password, string lastTimestamp)

the user requests a list of all the rules updated since the date indicated by the parameter and receives in return a *PlanningRules* object that is a list of *PlanningRule*.

Timestamp represents the creation date of the rules and is necessary to understand to what extent the information has been updated.

Each *PlanningRule* specify for each profile (*ProfileId*) the percentage chance for each available e-mail message that it will be shown (*EmailMessages*). Each *EmailMessage* is a *Rule* that specify the percentage chance (*Percentage*) for the e-mail message *EmailMessageId* that it will be shown.

Example of an *Planning Rule*

```
ProfileId: P1
EmailMessages:
  EmailMessageId: Em1
  Percentage: 3

  EmailMessageId: Em13
  Percentage: 10

  EmailMessageId: Em2
  Percentage: 7
```

In this case in the *situation* P1 the 3% of the impression scheduled for the next cycle must be allocated for the message Em1, the 10% for Em13 and the 7% for Em2.

If there isn't any new rules, the method doesn't give an object but NULL.

Example of a code – retrieving the rules

```
// This code sample retrieves the planning rules which determine the e-mail
message to visualize during the next time.
public static void RequestPlanningRules()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    string lastRequestTime = DateTime.Now.ToString();

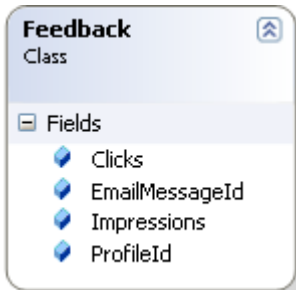
    try
    {
        PlanningRules rules = service.GetPlanningRules(user, password,
                                                    lastRequestTime);
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine("Planning Rules not retrieved due to the following
                            error: " + ex);
    }
}
```

The SetFeedback method

The feedback data represents what has happened on the E-Mail where Carousel's suggestions have been applied; in particular, it is concerned with the number of impressions and clicks received by the e-mail messages selected for the various profiles.

This data is of fundamental importance because it is by actually using it that the algorithm learns, evolves and generates ever more suitable and optimal rules; because of this, the more frequently that feedback is sent, the easier it will be for Carousel to improve the suggested rules.

Feedback that is sent on time and frequently ensures that Carousel will function at its best.



Using the method

```
SetFeedback(string userName, string password, Feedback[] feedback)
```

the user can periodically send a list of all the feedback gathered up until that point.

All *Feedback* must indicate the number of impressions and clicks (*Impressions*, *Clicks*) obtained by every e-mail message displayed

(*EmailMessageId*) to a particular user profile (*ProfileId*).

Every time the user sends feedback the application recalculates and updates the rules; for this reason, until new data has been sent, in memory we maintain the old rule list previously calculated with the usual value for the field *TimeStamp*.

Further requests for the rules will not give an object *PlanningRules* but NULL.

Note that recalculating and updating the rules are asynchronous and require some minutes.

Example of a code – sending feedback

```
// This code sample prepare and send the periodically list of feedback data that
the server needs to compute rules.
public void SetFeedback()
{
    // Set up service connection
    EmailServiceSoapClient service = new EmailServiceSoapClient();

    // Credentials
    string user = "INSERT_USERNAME_HERE";
    string password = "INSERT_PASSWORD_HERE";

    // Create the list of feedback
    // the advertisement and profile indicated must exist
    Feedback[] feedback = new Feedback[]
    {
        new Feedback
        {
            EmailMessageId = 1,
            ProfileId = 10,
            Impressions = 1500,
            Clicks = 75
        },
        new Feedback
        {
            EmailMessageId = 2,
            ProfileId = 5,
            Impressions = 2300,
```

```
        Clicks = 126
    }
};

try
{
    service.SetFeedback(user, password, feedback);
    Console.WriteLine("Feedback sent");
}
catch (ArgumentException ex)
{
    Console.WriteLine("Feedback not send due to the following error: "
        + ex);
}
}
```

About Noesis

Noesis is an Italian software house based in Pisa, Italy and has been founded by Academic professors of Pisa University.

Noesis' background assets include ideas, skills and technologies developed in research projects since 1997, involving large companies, small technology-oriented firms, and several universities. These projects were partially funded by the European Union and the Italian Government, partially self-financed by the industrial partners.

*The research projects developed technologies of artificial intelligence, adaptive systems, data mining, and text mining applied to **decision making, product/content recommendation, revenue management, and dynamic pricing.***

Noesis currently operates truly as a web company having the software development management in house and cooperating with external companies both in the Pisa area and on worldwide base for the engineering power and resources.

Noesis has established a network of partners both as developers and as distributors at a worldwide level for exploiting the solutions developed.

Learn more about Noesis at www.noesis-research.com

For additional information, please contact:

Noesis s.r.l.

*Corso Italia, 89
56125 Pisa
Italy*

Telephone +39 050 9911080

Fax +39 050 9911588

Email info@noesis-research.com

www.noesis-research.com