# Outbound Contact Deployment Guide

## Calling List Object

5/9/2025

# Contents

# Calling List Object

Calling Lists are tables that contain customer contact information, including telephone numbers to be called during an outbound campaign. Calling Lists are also configuration objects in Genesys Administrator. Define a Calling List object after you have created the Format and Table Access objects. If you need user-defined fields in your Calling List, create those fields and add them to the Format object before you create the Table Access object. When a format is associated with a Table Access object, you cannot add or delete fields in that format. You can create Filter and Treatment objects before, while, or after you define the Calling List object and add or remove these objects at any time.

| | |
|---|---|
| Note: | Only calling lists based on a 7.1 or higher format can be migrated to Outbound Contact 8.x.<br><br>Warning: Calling lists that are associated with formats that do not contain the switch_id, group_id, and treatments fields (such as those from version 7.0 or earlier), or that contain the phone and phone_type fields instead of the contact_info and contact_info_type fields (respectively) must be migrated to Outbound Contact solution 7.1 first, and then migrated to release 8.x. See the *Genesys Migration Guide* for more details. |

Create the Calling List object *before* the Calling List table exists in the relational database. This approach enables Genesys Administrator to create the Calling List table according to the properties of the Calling List configuration object. If the table does not exist yet, OCM displays this message The table [table name] does not exist. Create it? Click Yes to create the Calling List table.

If the Calling List table already exists in the database, the properties of the Calling List object must exactly match the structure of the table (as defined by its Format and Table Access objects) in the database.

## Creating a New Calling List Object

**Start**

1. In Genesys Administrator, select Provisioning > Outbound Contact > Calling List.
2. In the Details pane, click New.
3. Define the field values in the associated tabs, including the Configuration tab, the Options tab, the Campaigns tab, the Format tab, and the Treatments tab.

   | | |
   |---|---|
   | Note: | For other tabs, see the *Framework Genesys Administrator Help.* |

4. Click Save and Close.

**End**

## Calling List Object--Configuration Tab Fields

### General Section

#### Name

Required; the default value is `[Calling List]`. The name of the calling list. This value must be unique within the tenant.

#### Description

Optional; the default value is `[Blank]`. A brief description of the calling list.

#### Tenant

Automatically populated by the system.

#### State Enabled

Required; default is checked. A check box indicating that a customer interaction can be directed to this target. You can find a more complete description in *Framework Genesys Administrator Help*.

#### Calling Time From

The default value is `[8:00:00 AM]`. Not used at the list level, at this time.

#### Calling Time To

The default value is `[6:00:00 PM]`. Not used at the list level, at this time.

#### Maximum Attempts

Required; the default value is `[10]`. The maximum number of attempts to call a single record in this Calling List during one campaign. You cannot set this value to zero.

| | |
|---|---|
| Note: | This parameter is taken into account by Outbound Contact Server when applying treatments only. If the record is being processed again as a Personal or Campaign Callback, this parameter is not used by OCS. |

#### Script

The Script Property in the Campaign, Calling List, and Campaign Group defines the Script object, which contains all of the attributes that are required by Agent Scripting. For more information, see the section, "Attaching Script Information to OCS User Events and Telephony Events", in the Outbound Contact Reference Manual.

| | |
|---|---|
| Note: | For descriptions of the fields in the Data Base and Query sections of the `Configuration` tab, see *Framework Genesys Administrator Help.* |

### Format

The Format for the Calling List.

**Data Base and Query Sections/Calling List object**

These entries are associated with using SQL queries to create a Calling List.

### Table Name

The name of the table in the database.

### Data Access Point

Required; the default value is `[None]`. Specifies the `Database Access Point` through which the table is accessed. Enter a value by locating an existing `Database Access Point` object.

### DB Server

Required; the default value is `[None]`. Specifies the name of the DB Server associated with the database.

### SQL Server

Required; the default value is `[None]`. Specifies the name of the SQL Server used for running the SQL query.

### Type

Required; the default value is `[None]`. Specifies the type of database: MSSQL, Informix, DB2, Oracle, or Sybase.

### DB Name

Required; the default value is `[None]`. Specifies the name of the database in which the table exists.

### Login

Required; the default value is `[None]`. Specifies the login name for the database.

### Password

Required; the default value is `[None]`. Specifies the password for the database.

### Case Conversion

Case conversion method for key names of key-value lists coming from DB Server. This value specifies whether and how a client application converts the field names of a database table when it receives data from DB Server. If you select upper, field names are converted into uppercase; if you select lower, field names are converted into lowercase; and if you select any, field names are not converted. This value corresponds to the `dbcase` option in a configuration file.

| | |
|---|---|
| Notes: | • This setting does not affect the values of key-value lists coming from DB Server. That is, actual data is presented exactly as it appears in the database tables.<br>• Use the default value (any) unless directed to do otherwise by Genesys Technical Support. |

### SELECT FROM

Required; the default value is `[None]`. Specifies the table name from which to select records for the Calling List.

### WHERE

Required; default value is `[None]`. Specifies the SQL query to identify the records.

### ORDER BY

Required; the default value is `[None]`. Specifies how to order the records identified, based on the query.

## Calling List Advanced Entries

If you use the Calling List Advanced object instead of the Calling List object, you create lists based on specified Table Access and Filter objects rather than a SQL query.

### Table Access

Required; the default value is `[None]`. The Table Access object to which the Calling List refers. The Table Access object defines the format of the Calling List and identifies the DB Access Point used to access the Calling List table. Use the Browse button to locate an existing value.

### Log Table Access

The default value is `[None]`. Not used at this time.

### Filter

Optional; The default value is `[None]`. The filter to be applied to this calling list. A filter defines which call records within the Calling List table will be dialed by the campaign.

| | |
|---|---|
| Note: | The Filter field becomes enabled only when the Table Access field is populated. |

### Calling List Object--Options Tab

Use this tab to define Outbound Calling List--related options.

### Calling List Object--Campaigns Tab

This tab lists the Campaigns that include this Calling List.

### Calling List Object--Format Tab

On the Calling List object only (not Calling List Advanced object), use this tab to add the fields that define the Format for this calling list.

### Calling List Object--Treatments Tab Fields

You can add an existing treatment to the Calling List by clicking the Add button and selecting an available treatment. You can delete an applied treatment by selecting a displayed treatment and clicking Remove. Click Save and Close to the changes. This tab displays the following information for applied treatments.

#### Name

Name of applied treatment.

#### State

Indicates whether the Treatment is enabled or disabled. See Treatment Object for more information about setting up treatments for unsuccessful calls.

## Changing the Structure of a Calling List

The database administrator can make changes directly to the Calling List table using the database management system (DBMS) tools; however, Genesys Administrator will no longer be able to access this table. Genesys recommends that you create a new Calling List object, which will match the new physical structure of the table, as described on Creating a New Calling List Object.

## Fine-Tuning a Calling List

When Genesys Administrator creates a physical table for the Calling List, It also creates appropriate indexes on the table; however, if the structures of the dialing filters require additional indexes, the customer must create them. The customer also must perform tune-up procedures, such as updating

index statistics and maintaining transaction logs. To assign a List Weight to a calling list, see List Weight (in Configuration Manager); Share (in Genesys Administrator).

## Record Cancellation Requests and Customer ID

In addition to making record cancellation requests according to the record handle and the phone number, they can now be made according to the customer ID, which is currently leveraged by the Do Not Call feature.

Both the Desktop protocol and the 3rd party protocol have been extended to include (optionally) the `GSW_CUSTOMER_ID` attribute in the `RequestRecordCancel` and the `CM_ReqCancelRecord` requests respectively.

With this ability, record cancellation requests can identify sets of records not only by the record handle and the telephone number, but also by the customer ID. If more than one record identifier is included in the same request, the identifiers are prioritized as follows: record handle (highest), telephone, and customer ID (lowest). OCS processes `RequestRecordCancel` by customer ID in the same way that the other types of `RequestRecordCancel` are processed. The only difference is that the records that are subject to cancellation are determined by the value of the customer ID field in the Calling List table.

### Configuring Objects to Identify Records by Customer ID for Record Cancellation Requests

- To configure object to enable identification of record cancellation requests by customer ID.

| | |
|---|---|
| Note: | If you have already use the Customer ID for Do Not Call requests, you do not need to configure anything else. |

**Start**

1. Create a new user-defined field object. On the `Configuration` tab, define the fields as follows:
   - `Name = <user-specific name>`
   - `Data Type = varchar`
   - `Length = 64`
   - `Field Type = User-Defined`

   Also select the `Nullable` and `State Enabled` options.
2. On the `Options` tab for the `Field` object, assign the `send_attribute` to it by adding a `default` section. On the `Option` tab, define the fields as follows:
   - `Option Name = send_attribute`
   - `Option Value =GSW_CUSTOMER_ID`
3. Designate the new user-defined field as the `customer_id` option in the OCS Application object. In Genesys Administrator > `Provisioning > Environment > Applications > Outbound folder >`

OCS application > Options > OCServer section, create and define the customer_id option. Use the name of the new user-defined field as the value of customer_id.

- Option Name = customer_id
- Option Value = <name of new user-defined field>

4. Add the new field (defined in Step 1) to a new Format object.
   In Genesys Administrator > Provisioning > <Tenant> > Format object, create a new format for a Calling List table that will include the new user-defined field.
5. Create a Calling List object with the new format. In Genesys Administrator > Provisioning > <Tenant> > Calling Lists, configure a Calling List Advanced object with the following:
   In the Configuration tab, specify the following:

   - Table Access: <New Calling List>

   This is a new Calling List, formatted with the customer_id field.

**End**

## Customized Stored Procedures for Calling List and Campaign Objects

OCS uses stored procedures to get the values of different counters from the calling list database table (for example, the number of retrieved records), and to enable end users to customize them by defining or modifying the procedures code.

Customizing your stored procedures enables you to fine-tune a stored procedure for the following purposes:

- To adjust statistical calculations for pre-defined statistical counters.
- To define up to five custom database-related counters.

Starting in release 8.1.2, you can develop your own customized stored procedures by completing the following steps:

- Determine if the default procedure can be extended or should be completely rewritten.
- Create and debug code for the customized stored procedure, by using third-party DBMS tools.
- Store the body of the customized stored procedure in the Genesys configuration, by using Genesys Administrator.
- Configure OCS to completely replace the standard procedure code (using the instead option) or extend the code of the standard procedure with the customized procedure (using the before or after options).

Although you are limited by the predefined names, and input and output parameters in the custom procedure, there are no limitations within the body of the custom procedure.

## Configuring Custom Procedures

Custom procedures are configured in Genesys Administrator, in the Annex (Options) section of Campaign and Calling List objects. The section names for OCS options are standard and the calling list level options take precedence over the Campaign level options.

Reporting procedures are defined by configuring this pair of options: `report-procedure-body` and `report-procedure-location`.

Genesys Administrator has a simple, built-in text editor that enables you to define the body of the customized procedure. After the code has been defined and the location object for the code determined, Genesys Administrator stores the custom procedure code in the option value field as a binary string.

Define custom procedures in Genesys Administrator by navigating to one of the following locations in the interface:

- `Provisioning > Outbound Contact > Campaigns > Campaign object name > Configuration tab > Stored Procedures`
- `Provisioning > Outbound Contact > Calling Lists > Calling list object name > Configuration > Stored Procedures`

### Custom Reporting Procedures

Each DBMS has its own requirements for stored procedures. The following table describes the requirements for customization of stored reporting procedures for each DBMS, whether you are adding custom counters to existing procedures, or creating completely new user-defined stored procedures.

**Customizing Reporting Procedures**

| If Using Before/After for Custom Counters | If Using for Reporting Procedure, Use this Template for Procedure Body |
|---|---|
| MS SQL, Sybase | |
| Use variables `@CustomCounter01Num` - `@CustomCounter05Num` predefined as `NUMERIC` in procedure body for counters. Then use variables `@CustomCounter01` - `@CustomCounter05` defined as `VARCHAR(20)` to return values.<br><br>For example:<br><br>`SELECT @CustomCounter01Num = (SELECT COUNT(distinct record_id) FROM $list_tbl_name WHERE $dial_filter_where)`<br><br>`SELECT @CustomCounter01 = CONVERT(VARCHAR(20), ISNULL(@CustomCounter01Num, 0))` | `CREATE PROCEDURE $sr_proc_name(`<br>`@CTime VARCHAR(20),`<br>`@ReadyCount VARCHAR(20) OUTPUT,`<br>`@RetrievedCount VARCHAR(20) OUTPUT,`<br>`@FinalCount VARCHAR(20) OUTPUT,`<br>`@TotalCount VARCHAR(20) OUTPUT,`<br>`@TotalRecsCount VARCHAR(20) OUTPUT,`<br>`@ReadyRecsCount VARCHAR(20) OUTPUT,`<br>`@CustomCounter01 VARCHAR(20) OUTPUT,`<br>`@CustomCounter02 VARCHAR(20) OUTPUT,`<br>`@CustomCounter03 VARCHAR(20) OUTPUT,`<br>`@CustomCounter04 VARCHAR(20) OUTPUT,`<br>`@CustomCounter05 VARCHAR(20) OUTPUT)`<br>`AS`<br>`BEGIN`<br>`-- TODO: Add custom code here`<br>`END` |
| Oracle | |
| Use variables `v_CustomCounter01` - | `CREATE OR REPLACE PROCEDURE $sr_proc_name(` |

| | |
|---|---|
| v_CustomCounter05 predefined as NUMBER in procedure body for counters. Then use variables p_CustomCounter01 - p_CustomCounter05 defined as VARCHAR2 to return values.<br><br>For example:<br><br>`SELECT COUNT(*) INTO v_CustomCounter01 FROM (select DISTINCT record_id FROM $list_tbl_name WHERE $dial_filter_where);`<br><br>`p_CustomCounter01 := TO_CHAR(NVL(v_CustomCounter01, 0));` | `p_CTime VARCHAR2,`<br>`p_ReadyCount OUT VARCHAR2,`<br>`p_RetrievedCount OUT VARCHAR2,`<br>`p_FinalCount OUT VARCHAR2,`<br>`p_TotalCount OUT VARCHAR2,`<br>`p_TotalRecsCount OUT VARCHAR2,`<br>`p_ReadyRecsCount OUT VARCHAR2,`<br>`p_CustomCounter01 OUT VARCHAR2,`<br>`p_CustomCounter02 OUT VARCHAR2,`<br>`p_CustomCounter03 OUT VARCHAR2,`<br>`p_CustomCounter04 OUT VARCHAR2,`<br>`p_CustomCounter05 OUT VARCHAR2)`<br>`AS`<br>`BEGIN`<br>`-- TODO: Add custom code here`<br>`END;`<br>`END $sr_proc_name;` |
| DB2 | |
| Use variables iCustomCounter01 - iCustomCounter05 predefined as INTEGER in procedure body for counters. Then use variables CustomCounter01 - CustomCounter05 defined as CHAR(32) to return values.<br><br>For example:<br><br>`SELECT COUNT(DISTINCT record_id) INTO iCustomCounter01 FROM $list_tbl_name WHERE $dial_filter_where;`<br><br>`SET CustomCounter01= CHAR(VALUE(iCustomCounter01, 0));` | `CREATE PROCEDURE $sr_proc_name(`<br>`IN CTime INTEGER,`<br>`OUT ReadyCount CHAR(32),`<br>`OUT RetrievedCount CHAR(32),`<br>`OUT FinalCount CHAR(32),`<br>`OUT TotalCount CHAR(32),`<br>`OUT TotalRecsCount CHAR(32),`<br>`OUT ReadyRecsCount CHAR(32),`<br>`OUT CustomCounter01 CHAR(32),`<br>`OUT CustomCounter02 CHAR(32),`<br>`OUT CustomCounter03 CHAR(32),`<br>`OUT CustomCounter04 CHAR(32),`<br>`OUT CustomCounter05 CHAR(32))`<br>`LANGUAGE SQL`<br>`BEGIN`<br>`-- TODO: Add custom code here`<br>`END` |
| Informix | |
| Use variables v_CustomCounter01 - v_CustomCounter05 predefined as INT in the procedure body for counters. Procedure returns these variables in the RETURN statement.<br><br>For example:<br><br>`SELECT COUNT(DISTINCT record_id) INTO v_CustomCounter01 FROM $list_tbl_name WHERE $dial_filter_where;` | `CREATE PROCEDURE $sr_proc_name(`<br>`p_CTime CHAR(32))`<br>`RETURNING INT, INT, INT, INT, INT, INT,`<br>`INT, INT, INT, INT, INT;`<br>`-- TODO: Add custom code here`<br>`RETURN v_ReadyCount, v_RetrievedCount,`<br>`v_FinalCount, v_TotalCount,`<br>`v_TotalRecsCount, v_ReadyRecsCount,`<br>`v_CustomCounter01, v_CustomCounter02,`<br>`v_CustomCounter03, v_CustomCounter04,`<br>`v_CustomCounter05;`<br>`END PROCEDURE` |
| PostgreSQL | |
| Use variables v_custom01 - v_custom05 declared as INTEGER in procedure body for counters. Then use variables p_CustomCounter01 - p_CustomCounter05 declared as VARCHAR(32) to return values. | `CREATE OR REPLACE FUNCTION $sr_proc_name(`<br>`c_time numeric(18,0),`<br>`OUT p_ReadyCount varchar(32),`<br>`OUT p_RetrievedCount varchar(32),`<br>`OUT p_FinalCount varchar(32),` |

For example:

```
DECLARE v_custom01 INTEGER := 0;

SELECT COUNT(DISTINCT record_id) INTO
v_custom01 FROM $list_tbl_name WHERE
($dial_filter_where);
p_CustomCounter01 :=
CAST(COALESCE(v_custom01, 0) AS VARCHAR);
```

```
OUT p_TotalCount varchar(32),
OUT p_TotalRecsCount varchar(32),
OUT p_ReadyRecsCount varchar(32),
OUT p_CustomCounter01 varchar(32),
OUT p_CustomCounter02 varchar(32),
OUT p_CustomCounter03 varchar(32),
OUT p_CustomCounter04 varchar(32),
OUT p_CustomCounter05 varchar(32))
AS $$
--TODO: Declare variables here
BEGIN
--TODO: Add custom code here
RETURN;
END;
$$ LANGUAGE plpgsql;
```

## Customized Stored Procedures Macro Expressions

To simplify the creation of customized procedures, OCS supports the use of macro expressions in custom procedures. OCS substitutes this macro expression for the actual values when the SQL code for the custom procedure is generated, depending on the calling list for which the code generation is executed. By using macro expressions, OCS enables you to define a custom procedures template on a Campaign level that can be generated for each calling list that is configured under the Campaign. See the **Macro Expressions for Custom Stored Procedures** table for macro expressions for custom stored procedures. Refer also to the **OCS Supported Macro Expressions** table that also contains other macro expressions supported by OCS.

**Macro Expressions for Custom Stored Procedures**

| Parameter name | To be submitted for: | Usage example in the custom procedure body |
|---|---|---|
| `$list_tbl_name` | Name of the calling list table | `UPDATE $list_tbl_name SET ...` |
| `$dial_filter_where` | WHERE clause of the dialing filter | `UPDATE $list_tbl_name SET ..." WHERE $dial_filter_where` |
| `$dial_filter_order_by` | ORDER BY clause of the dialing filter | `SELECT TOP 10 chain_id FROM $list_tbl_name ORDER BY $dial_filter_order_by` |
| `$list_dbid` | DBID of the Calling List | |
| `$camp_dbid` | DBID of the Campaign | |
| `$group_dbid` | DBID of the Group | |
| `$camp_group_dbid` | DBID of the Campaign Group (session) | |

## Monitoring Dynamic Modification to Custom Procedures

OCS monitors dynamic modifications to the dialing filter and recreates customized procedures if the filter is updated, if the procedure contains the `$dial_filter_where` and/or `$dial_filter_order_by` macro expression.

OCS also monitors the options that define the body and location of the customized procedure and regenerates these procedures if these options are modified.

### Implementation of Custom Procedures

There are three possible configurations:

1. Before--A call to the custom code is executed before any code execution in the hosting procedure, immediately following the BEGIN clause.
2. After--A call to the custom code is executed after all of the code in the hosting procedure is executed.
3. Instead--The body of the standard stored code is fully replaced with the custom procedure code.

#### Logging and Debugging

To assist in debugging custom procedures, OCS logs all SQL code for custom procedures when it is created and any errors in the custom procedures when they are created.

OCS logs the SQL code for procedures by using the `DEBUG` log level. Genesys recommends that any errors that occur when procedures are created be logged with the `TRACE` log message severity to ensure the proper alarms on Message Server are configured.

## Omitting Verification of the Campaign DBID for Callbacks and Rescheduled Records Retrieval

Stored procedures take the Campaign DBID (and optionally the Group DBID as configured in the OCS `callback-observe-group` option) into account when any record type, other than General, is retrieved.

Use the `callback-observe-campaign` option to control whether Campaign DBID is considered by the OCS record-retrieval mechanism.