# Newsletter - Developer Guide

2017-02-10

Orckestra, Europe
Nygårdsvej 16
DK-2100 Copenhagen
Phone +45 3915 7600

www.orckestra.com

C1 CMS

# Contents


**1 INTRODUCTION ..... 3**

1.1 Before You Start 3

**2 WRITING CUSTOM MAILING LIST MEMBER PROVIDERS ..... 4**

2.1 Default Mailing List Member Providers 4
2.2 Custom Mailing List Member Providers 5

*2.2.1 Sample Code 5*

2.3 Creating Class Library 6

*2.3.1 Adding References 6*

2.4 Creating Required Classes 6

*2.4.1 Step 1: Create Mailing List Definition Class 7*
*2.4.2 Step 2: Create Mailing List Member Class 9*
*2.4.3 Step 3: Create Mailing List Provider Class 11*

2.5 Building and Deploying 13

*2.5.1 Automatic Deployment 13*
*2.5.2 Manual Deployment 13*

2.6 Using Custom Mailing List 14

**3 LOCALIZING NEWSLETTER ..... 16**

3.1 Localizing Subscribe and Unsubscribe Forms 16
3.2 Localizing Newsletter GUI 17

*3.2.1 Creating Localization Files 17*
*3.2.2 Translating Newsletter GUI strings 17*
*3.2.3 Switching the CMS Console GUI Language 18*

3.3 Localizing Mailing Lists, Newsletters and Templates 18

*3.3.1 Localizing Mailing Lists 18*
*3.3.2 Localizing Newsletter Templates 18*
*3.3.3 Localizing Newsletters 19*

**4 GETTING NEWSLETTER ID ..... 20**


C1 CMS

# 1 Introduction

This guide is intended for Web and C# developers and administrators who want to extend or localize the Newsletter add-on.

The Newsletter add-on allows users to create mailing lists and create and send newsletters directly from C1 CMS.

For information about using the Newsletter add-on, please refer to the Newsletter User Guide.

In this guide you will learn how to create custom mailing list member providers by integrating them with external databases.

For creating custom member providers, you should be proficient in C#/.NET programming and know the API of a 3rd-party database application you want to integrate mailing lists with.

Besides, you will learn how to localize Newsletter's GUI, forms and content elements. For localization tasks, you should have a general idea of how localization works in C1 CMS and know how to edit XML-formatted files.

## 1.1 Before You Start

You should make sure that you have the following prerequisites in place before you start the guide:

- You have installed and configured the latest version of C1 CMS
- You use a test C1 CMS installation that you can experiment with.
- You have installed and configured the latest version of the Newsletter add-on on this test C1 CMS environment.

**Important:** You should *never use the production environment* for learning purposes while trying the examples and following the instructions in this guide.

# 2 Writing Custom Mailing List Member Providers

The mailing lists in the Newsletter add-on are filled with members by mailing list member providers.

However, you are not limited to the sources existing within the Newsletter add-on or C1 CMS itself for member lists. You can create your own mailing list member providers to get their member lists from external sources such as databases and database-like files.

## 2.1 Default Mailing List Member Providers

The Newsletter add-on comes with two default providers:

- SubjectBasedMailingListProvider
- DataTypeBasedMailingListProvider

As it follows from their names, they handle subject-based and datatype-based mailing lists respectively.

Two corresponding assemblies contain the code for these providers:

- Composite.Community.Newsletter.DataTypeBased.dll
- Composite.Community.Newsletter.SubjectBased.dll

These assemblies are installed in the Bin folder of the C1 CMS-based website.

The providers are plugged in via the C1 CMS configuration file (*\\<website>\App_Data\Composite\Composite.config*) in the <**Composite.Community.Newsletter.Plugins.MailingListProviderConfiguration**> section:

```
<Composite.Community.Newsletter.Plugins.MailingListProviderConfiguration>
       <MailingListProviderPlugins>
              <add name="Composite.Community.Newsletter.SubjectBased"
type="Composite.Community.Newsletter.SubjectBased.StandardPlugins.MailingLi
stProvider.SubjectBasedMailingListProvider,
Composite.Community.Newsletter.SubjectBased" />
              <add name="Composite.Community.Newsletter.DataTypeBased"
type="Composite.Community.Newsletter.DataTypeBased.StandardPlugins.MailingL
istProvider.DataTypeBasedMailingListProvider,
Composite.Community.Newsletter.DataTypeBased" />
       </MailingListProviderPlugins>
</Composite.Community.Newsletter.Plugins.MailingListProviderConfiguration>
```

Listing 1: Default providers plugged in

Each <**add**> element stands for one mailing list member provider and has two mandatory attributes:

- name
- type

The **name** attribute specifies the name of the mailing list member provider.

The **type** attribute specifies the name of the mailing list member provider class followed by the provider's assembly separated by a comma.

C1 CMS

In the default configuration, these values are as follows:

**Name:** Composite.Community.Newsletter.SubjectBased

**Provider:**
Composite.Community.Newsletter.SubjectBased.StandardPlugins.MailingListProvider.SubjectBasedMailingListProvider

**Assembly:** Composite.Community.Newsletter.DataTypeBased

**Name:** Composite.Community.Newsletter.DataTypeBased

**Provider:**
Composite.Community.Newsletter.SubjectBased.StandardPlugins.MailingListProvider.DataTypeBasedMailingListProvider

**Assembly:** Composite.Community.Newsletter.SubjectBased

The **SubjectBasedMailingListProvider** makes use of members manually added to the subject-based mailing lists. They are centrally stored in the built-in global datatype **Composite.Community.Newsletter.SubjectBased.Member** that comes with the Newsletter add-on.

The **DataTypeBasedMailingListProvider** gets its member list from any existing global datatype that has the *Email* field.

## 2.2 Custom Mailing List Member Providers

Using the plug-in model of the Newsletter add-on, you can create custom mailing list member providers and integrate them into the Newsletter add-on.

These providers get their member lists from external databases and database-like files, for example, a CRM system, an SQL database, an Excel spreadsheet or even a plain-text file (flat-file database).

Creating a custom provider means creating an assembly (similar to the default two), which implements the plug-in model-related classes and interfaces and plug it into the Newsletter.

The steps to create a custom mailing list member provider include:

1. Creating a class library project and adding required references to it.
2. Creating required classes by implementing the Newsletter plug-in model.
3. Building and deploying the provider on the website.

In the following few sections, you will learn more about these steps.

### 2.2.1 Sample Code

For illustration, we will create a sample custom provider.

To simplify the sample code, instead of using 3rd-party application APIs to import member lists, we will use a flat-file database stored in the *MailingList.txt* file in the root folder of the target website.

The member list will only consist of one field, “Email”. Each email address will be kept in the *MailingList.txt* file on a new line.

## 2.3 Creating Class Library

Each custom mailing list provider is represented by a class library assembly. So you should start by creating a class library project by using a Class Library project template in Visual Studio 2008 (Visual C#, Windows, Class Library).

For our sample we will create the project called **Composite.Community.Newsletter.FileBased**

### 2.3.1 Adding References

To be able to use C1 CMS, the Newsletter add-on and other functionality, you should add a number of references to the project.

Normally, you should add references to the assemblies located in the *Bin* folder of your website with the Newsletter add-on already installed.

The following references must be added to the project:

- Composite
- Composite.Community.Newsletter
- Composite.Generated
- Composite.Workflows
- ICSharpCode.SharpZipLib
- Microsoft.Practices.EnterpriseLibrary.Common
- Microsoft.Practices.EnterpriseLibrary.Configuration.Design
- Microsoft.Practices.EnterpriseLibrary.ExceptionHandling
- Microsoft.Practices.EnterpriseLibrary.Logging
- Microsoft.Practices.EnterpriseLibrary.Validation
- Microsoft.Practices.ObjectBuilder
- System.Configuration
- TidyNet

Once you have created a Class Library project and added all required references, go on to create a number of classes for your mailing list member provider.

## 2.4 Creating Required Classes

To integrate your mailing list member provider into the Newsletter add-on, you should create 3 classes, which are implementation of two abstract classes available in **Composite.Community.Newsletter** and one interface available in **Composite.Community.Newsletter.Plugins.MailingListProvider**:

- Mailing List Definition Class
- Mailing List Member Class
- Mailing List Provider Class

In the following subsections, you will learn more about these abstract classes and interface and see the sample code that implements them.

### 2.4.1 Step 1: Create Mailing List Definition Class

First, you should create a class that will represent a custom mailing list, that is, a mailing list definition class. This class must supply the following information to the Newsletter code:

- GUID
- Title
- Description
- Flag to indicate whether members are culture-specific
- Entity token

The **GUID** uniquely identifies the mailing list in the Newsletter add-on.

The **Title** and **Description** are the GUI elements that the end user will identify the mailing list by.

The **flag** indicates whether the members are culture-specific. If true, different mailing list members can be available for each locale on C1 CMS. If false, the member list will be available only for the default locale.

The **Entity token** is used by the C1 CMS Security model to place elements in the tree structures such as those representing pages in the *Content* perspective.

You should create the mailing list definition class by inheriting it from the abstract **MailingListDefinition** class (**Composite.Community.Newsletter.MailingListDefinition**) and overriding its properties:

```
public abstract class MailingListDefinition
{
	protected MailingListDefinition();
	public abstract string Description { get; }
	public Guid Id { get; protected set; }
	public abstract EntityToken ListEntityToken { get; }
	public abstract bool MembersAreCultureSpecific { get; }
	public abstract string Title { get; }
}
```

Listing 2: Abstract MailingListDefinition class

In the following example, we have created the **FileBasedMailingListDefinition** class for our sample mailing list member provider:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Composite.Security;
using Composite.Community.Newsletter.ElementProvider;

namespace
Composite.Community.Newsletter.FileBased.Plugins.MailingListProvider
{
	public class FileBasedMailingListDefinition : MailingListDefinition
	{
		private EntityToken _listEntityToken;

		public FileBasedMailingListDefinition(Guid id)
		{
			this.Id = id;
			this._listEntityToken = new MailingListEntityToken(id,
"FlatFile");
		}

		public override string Description
		{
			get { return "File-based mailing list"; }
		}

		public override EntityToken ListEntityToken
		{
			get { return _listEntityToken; }
		}

		public override bool MembersAreCultureSpecific
		{
			get { return false; }
		}

		public override string Title
		{
			get { return "File-based mailing list"; }
		}
	}
}
```

Listing 3: Sample MailingListDefinition implementation

As you can see in the example above:

1. In the constructor, we initialize the class's **ID** property with the GUID passed to the constructor.
2. We also create a new instance of the **MailingListEntityToken** class (**Composite.Community.Newsletter.ElementProvider.MailingListEntityToken**) and initialize the private **ListEntityToken** variable with it.
3. Next, we provide the title and description with hard-coded strings.
4. Then, we provide the **EntityToken** property we have initialized in Step 2.
5. Finally, we indicate that the members of this list are not culture-specific.

### 2.4.2 Step 2: Create Mailing List Member Class

Now you should create a class that will represent a member of the custom mailing list. Each member can be of one or more types. Each type may include one or more fields. All these fields will be available to the user when he or she creates a newsletter. By using types you can combine sets of fields for a member object in your mailing list member provider.

The class you are about to create must be initialized with a string that contains an email address and supply a member as an object to the Newsletter code.

You should create the mailing list member class by inheriting it from the abstract **MailingListMember** class (**Composite.Community.Newsletter.MailingListMember**) and overriding its **GetMemberObject** method:

```
public abstract class MailingListMember
{
        protected string _email;
        public MailingListMember(string email);
        public string Email { get; }
        public virtual object GetMemberObject(Type type);
}
```

Listing 4: Abstract MailingListMember class

C1 CMS

In the following example, we have created the **FileBasedMailingListMember** class for our sample mailing list member provider:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace
Composite.Community.Newsletter.FileBased.Plugins.MailingListProvider
{

	public class Member
	{
		public Member(string email)
		{
			this.Email = email;
		}

		public string Email { get; private set; }
	}

	public class FileBasedMailingListMember : MailingListMember
	{
		private Member _data = null;

		public FileBasedMailingListMember(string email) : base(email)
		{
			_data = new Member(email);
		}

		public override object GetMemberObject(Type type)
		{
			if (type == typeof(Member))
			{
				return _data;
			}
			throw new InvalidOperationException("Unknown type...");
		}
	}
}
```

Listing 5: Sample MailingListMember implementation

As you can see in the example above:

1. First, we create a class called “Member”, which will serve as our member object type.
2. Then in the **FileBasedMailingListMember**’s constructor we create a new **Member** object using the email address passed to the constructor and initialize the private Member variable using this object.
3. In the **GetMemberObject** method, we return the **Member** object created in step 2 if the type passed to this method is of the Member type; otherwise, we throw an invalid operation exception indicating that the input parameter is of an unknown type.

C1 CMS

### 2.4.3 Step 3: Create Mailing List Provider Class

Finally, you have to create a class that represents the mailing list provider.

The class should:

- Retrieve a list of member object types available in a specific mailing list
- Retrieve a list of mailing list definitions
- Retrieve a list of members in a specific mailing list (being able to limit the number of members to get and skip members until a specific email is found on the list)
- Build the unsubscribe link that will be inserted in newsletters

You should create a mailing list provider class by implementing the **IMailingListProvider** interface (**Composite.Community.Newsletter.Plugins.MailingListProvider.IMailingListProvider**):

```
public interface IMailingListProvider
{
	string BuildUnsubscribePathAndQuery(string memberEmail, Guid mailingListId);
	IEnumerable<Type> GetAvailableMemberObjectTypes(Guid mailingListId);
	IEnumerable<MailingListDefinition> GetMailingListDefinitions();
	IEnumerable<MailingListMember> GetMemberChunk(Guid mailingListId, int maxMembersToGet, string skipUntilEmail);
}
```

Listing 6: IMailingListProvider interface

C1 CMS

In the following example, we have created the **FileBasedMailingListProvider** for our sample solution:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Practices.EnterpriseLibrary.Common.Configuration;
using Composite.Community.Newsletter.Plugins.MailingListProvider;
using Composite.IO;
using System.IO;
namespace
Composite.Community.Newsletter.FileBased.Plugins.MailingListProvider
{
	[ConfigurationElementType(typeof(NonConfigurableMailingListProvider)
)]
	public sealed class FileBasedMailingListProvider :
IMailingListProvider
	{

		public IEnumerable<Type> GetAvailableMemberObjectTypes(Guid
mailingListId)
		{
			yield return typeof(Member);
		}

		public IEnumerable<MailingListDefinition>
GetMailingListDefinitions()
		{
			yield return new FileBasedMailingListDefinition(new
Guid("{E9A9A3CB-B338-4a5f-8D25-6C6CC2B8B9A8}"));
		}

		public IEnumerable<MailingListMember> GetMemberChunk(Guid
mailingListId, int maxMembersToGet, string skipUntilEmail)
		{

			var memberlist =
File.ReadAllLines(PathUtil.BaseDirectory + "Mailinglist.txt").Select(email
=> (MailingListMember)new FileBasedMailingListMember(email));
			if (string.IsNullOrEmpty(skipUntilEmail))
			{
				return memberlist.Take(maxMembersToGet);
			}
			return memberlist.SkipWhile(d => d.Email !=
skipUntilEmail).Skip(1).Take(maxMembersToGet);
		}
	}
}
```

Listing 7: Sample IMailingListProvider implementation

As you can see in the example above:

1. For the member object types we return our type **Member**.

2. For the mailing list definitions, we create and return our **FileBasedMailingListDefiniton** object.

3. For the mailing list member objects, we read our file which serves as a flat-file database where members listed each on a new line and return this list after verifying a number of conditions against values passed to this method. The path to the file and its name are hard-coded as “MailingList.txt” in the root folder of the website.

C1 CMS

Once you have finished creating all the required classes, you should proceed to build and deploy your mailing list member provider.

## 2.5 Building and Deploying

Once you have built your solution, you are ready to deploy it and use it on the website.

To deploy the solution, you can follow one of the two approaches:

- Automatic
- Manual

For **automatic** deployment, you should build an add-on for your mailing list member provider and then install it on your C1 CMS via its Packages system.

For **manual** deployment, you should copy the assembly you have just built to a specific folder on your website and plug it in via the C1 CMS configuration file.

### 2.5.1 Automatic Deployment

For automatic deployment, you should build an add-on for your mailing list member provider following the standard add-on-building procedure.

In the *Install.xsl* file, you should specify the name of your custom provider, its class and its assembly.

The following is the sample for the **FileBasedMailingListProvider** we have created:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
	<xsl:template match="@* | node()">
		<xsl:copy>
			<xsl:apply-templates select="@* | node()" />
		</xsl:copy>
	</xsl:template>
	<xsl:template
match="/configuration/Composite.Community.Newsletter.Plugins.MailingListPro
viderConfiguration/MailingListProviderPlugins">
		<xsl:copy>
			<xsl:apply-templates select="@* | node()" />
			<xsl:if
test="count(add[@name='Composite.Community.Newsletter.FileBased'])=0">
				<add
name="Composite.Community.Newsletter.FileBased"
type="Composite.Community.Newsletter.FileBased.Plugins.MailingListProvider.
FileBasedMailingListProvider, Composite.Community.Newsletter.FileBased" />
			</xsl:if>
		</xsl:copy>
	</xsl:template>
</xsl:stylesheet>
```

Listing 8: Sample Install.xsl for FileBasedMailingListProvider

In the similar way, you should modify the *Uninstall.xsl*.

Once you have built the add-on, you can install it via the Packages system in the CMS Console (**System** > **Packages** > **Installed Packages** > **Local Packages** > **Install local package**).

### 2.5.2 Manual Deployment

To deploy the custom mailing list member provider manually:

1. Copy the mailing list member provider assembly to the *Bin* subfolder in the root folder of your website.
2. Open the C1 CMS configuration file found at *\\<website>\App_Data\Composite\Composite.config*
3. Locate the <**Composite.Community.Newsletter.Plugins.MailingListProviderConfiguration**> section.
4. Under the <**MailingListProviderPlugins**> element add the name of your provider, its class and its assembly.

   The following is the sample for the **FileBasedMailingListProvider** we have created:

```
<add name="Composite.Community.Newsletter.FileBased"
type="Composite.Community.Newsletter.FileBased.Plugins.MailingListPr
ovider.FileBasedMailingListProvider,
Composite.Community.Newsletter.FileBased" />
```

   Listing 9: Sample of plugging in `FileBasedMailingListProvider`

5. Now restart the server and then refresh the browser window (or tab) in which you have your CMS Console running.

Now that you have deployed the custom mailing list member provider, you can start using it.

## 2.6 Using Custom Mailing List

Once you have deployed your custom mailing list member provider, it will appear in the Content perspective as another mailing list.

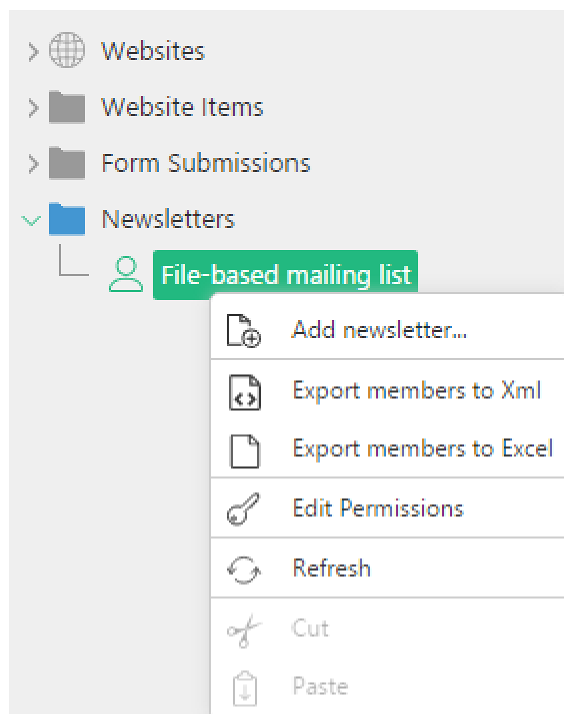


Figure 1: Custom file-based mailing list

This mailing list will use the list of members retrieved from your database or database-like file.

Since the member list is based on the external source, you cannot add or remove members from the list as you can do with the subject-based mailing list.

For the custom mailing list, you can create and send newsletters as well as export members if necessary.

# 3 Localizing Newsletter

The default language used in the Newsletter add-on is English. However, as other add-ons and C1 CMS itself it can be localized.

Localization of the Newsletter add-on serves these purposes:

- Having the forms used on a website in the same language as that of the website
- Having the Newsletter GUI in the desirable language
- Having mailing lists, newsletter templates and newsletters localized

Each purpose implies its own approach, and the following few sections will cover them in detail.

## 3.1 Localizing Subscribe and Unsubscribe Forms

The strings use on the Subscribe and Unsubscribe forms are stored in a resource file (.resx) in the App_GlobalResources folder and available for English (US) out-of-the-box.

To localize the forms:

1. Make a copy of ~/App_GlobalResources/Composite/Community/Newsletter.resx in the same folder, adding the culture code of the target language to the filename (before the extension .resx). For example, for Danish: Newsletter.da-DK.resx
2. Edit the localization file in Visual Studio.
3. Translate the Value of each string key-value entry.

Here is the list of strings available for translation:

| Key | Value (in English by default) |
|---|---|
| MailingList | Mailing list: |
| MailingLists | Mailing lists: |
| MemberAlreadySubscribed | Please note that you were already signed up for the following mailing lists: |
| MemberEmail | Email |
| MemberName | Name |
| NeverMail_UnsubscribeSuccessfully | Successfully unsubscribed |
| NeverMail_ValidateEmail | Please enter a valid email address and try again. |
| SubjectBased_ConfirmMessage | A confirmation message has been sent to your email address |
| SubjectBased_SelectMailingList | Select a mailing list |
| SubjectBased_SuccessfullySubscribed | Thank you for your subscription |
| SubjectBased_Unsubscribe_YouAreNotSubscribed | You are not subscribed |
| SubjectBased_Unsubscribe_MemberDoesNotExists | This member does not exist |
| SubjectBased_Unsubscribe_UnsubscribeSuccessfully | Successfully unsubscribed |

| SubjectBased_ValidateEmail | Please enter a valid email address and try again. |
| --- | --- |
| SubjectBased_WrongConfirmUrl | Wrong confirm URL |
| Subscribe | Subscribe |
| Unsubscribe | Unsubscribe |
| UnsubscribeFromAll | Unsubscribe from all |

## 3.2 Localizing Newsletter GUI

By default, the Newsletter GUI is presented in English. To have the GUI strings in another language, you should:

1. Create localization files for the target language.
2. Translate the related strings in the corresponding localization file.
3. Switch the regional settings in C1 CMS.

### 3.2.1 Creating Localization Files

All the strings used in the Newsletter GUI visible in the CMS Console are stored in the Newsletter GUI localization files.

When the Newsletter add-on is installed, its GUI localization files are placed in the folder ~\Composite\InstalledPackages\localization\ as:

- Composite.Community.Newsletter.en-us.xml
- Composite.Community.Newsletter.SubjectBased.en-us.xml
- Composite.Community.Newsletter.DataTypeBased.en-us.xml
- Composite.Community.Newsletter.FunctionBased.en-us.xml

As their names suggest, all the default Newsletter localization files contain strings in English (“en-us”).

Copy each file replacing the culture code in its name with the proper one and place it in the same folder. For example, if you localize Newsletter to Danish, you should create these files by copying their English counterparts as follows:

- Composite.Community.Newsletter.da-dk.xml
- Composite.Community.Newsletter.SubjectBased.da-dk.xml
- Composite.Community.Newsletter.DataTypeBased.da-dk.xml
- Composite.Community.Newsletter.FunctionBased.da-dk.xml

### 3.2.2 Translating Newsletter GUI strings

Now you can edit each localization file of your target language and translate all the strings.

Each file is XML-formatted and you can edit it in any XML editor.

Its schema is standard: each string is represented with one <string> element under the **<strings>** root element. Each **<string>** element follows the key/value pattern represented with corresponding key and value attributes.

The value in the **key** attribute is referred internally from within the code. You cannot change the key attribute's value.

The value in the **value** attribute is what the end user can see in the GUI or on a web form. This is the value you should edit when translating the string.

Once finished, you may need to restart the server (Tools | Restart Server).

### 3.2.3 Switching the CMS Console GUI Language

Please see "Switching GUI Language".

## 3.3 Localizing Mailing Lists, Newsletters and Templates

Localization of mailing lists, newsletter templates and newsletters is similar to localization of websites and web pages in C1 CMS. However, it has some differences you should be aware of while localizing.

### 3.3.1 Localizing Mailing Lists

Only subject-based mailing lists can be localized in Newsletter. The localization procedure is as simple as localizing a web page:

1. Switch to another language. All the subject-based mailing lists marked as "not translated" will appear under Newsletter in the Content Perspective.
2. Right-click the mailing list you want to translate.
3. In the shortcut menu, click **Translate**. The mailing list will change its icon and, if it has any newsletters in the main language, they will all appear below as "not translated".
4. Repeat Steps 2-3 for as many mailing lists as you need.

**Important:** The translated mailing list initially ***has no members***.

It does not inherit the member list from the main language. You should add members to the list manually.

The translated mailing list does inherit all the newsletters from the main language, though. Before using them, you should translate them, too.

You can further edit the mailing list properties such as the name and description. They will be specific to the current language and will not affect the properties of the mailing list in the main language.

### 3.3.2 Localizing Newsletter Templates

Unlike website templates, newsletter templates must be localized before you create newsletters based on them.

Besides, you will be able to translate any existing newsletters based on the templates in the main language only after you first translate the newsletter templates they are based on.

**To localize a newsletter template:**

1. Switch to another language. All the templates marked as "not translated" will appear under **Newsletter Templates** in the Layout Perspective.
2. Right-click the template you want to localize.

3. In the shortcut menu, click **Localize Newsletter Template**. A dialog box will appear with the properties set by default to those in the main language.
4. Change the values where necessary and click **OK**. The template will change its icon in the Layout Perspective.
5. Repeat Steps 2-4 for as many templates as you need.

### 3.3.3 Localizing Newsletters

**Important:** You can localize the newsletter only after you have localized a newsletter template it is based on.

**To localization a newsletter**:

1. Switch to another language.
2. Expand a mailing list with the newsletter you want to localize. All the newsletters marked as "not translated" will appear under the mailing list in the Content Perspective.
3. Right-click the newsletter you want to localize.
4. In the shortcut menu, click **Localize Newsletter**. The newsletter will change its icon.
5. Repeat Steps 2-4 for as many mailing lists as you need.

Now you can change the newsletter's properties and content, which will be only specific to this language and will not affect the properties and content of the newsletter in the main language.

# 4 Getting Newsletter ID

You can add a link in the newsletter to publicly show this newsletter on a page on your website. For this, you need to get the newsletter's ID via an ad-hoc CMS function.

Please note that you can only get the ID from within the newsletter. That is why this function must be inserted either in the newsletter itself or in the newsletter template in use.

Creating a function is up to you; however, you need to add an input parameter of a specific type and set up its default value as described below:

1. Create an XSLT function, for example, "Demo.Newsletter.ShowOnPage".
2. Add an input parameter of the String type to the function, for example, "Newsletter".
3. In the default value of this parameter, remove the "Composite.Constant.String" function and add the "Composite.Utils.GetInputParameter" function.
4. In its "Parameter name" parameter, type in "NewsletterId" (exactly as here).
5. On the "Template" tab, make use of this parameter's value (GUID), the way you need.
6. Insert this function in the newsletter or newsletter template.

When the recipient receives this newsletter, it will contain its ID.

You can implement Step 5 as you need accessing the newsletter ID as **/in:inputs/in:param[@name='Newsletter']** in your XSLT.

For illustration purposes, let's assume that you have a page **~/ViewNewsletter**. It expects a query string parameter "Newsletter" with a newsletter's ID and shows the newsletter's content by its newsletter ID.

In the template of your function, the markup may be similar to this:

```
<a
href="~/ShowNewsletter?Newsletter={/in:inputs/in:param[@name='Newsletter']}
">View Online</a>
```

C1 CMS