

# 1. Extension Manager

## 1.1. Introduction

The extension manager is designed to solve many of the problems that occur when it comes to install applications or extensions. One of the main goals of the extension manager is to extend the Railo core with several functionalities. For example video processing with the tag CFVIDEO will be available as an extension that can easily be added to the core functionality.

In addition applications can be installed as well. In order to make that as easy as possible, Railo offers an extension API that allows you to create your own extension and offer a webservice that delivers several extensions.

The extension manager consists out of several elements that we will describe in this documentation. Basically the extension manager consists out of two parts:

- extension provider (webservice)  
provides extensions for the Railo administrators
- extension consumer (Railo administrators)  
This part is consuming the extension provider

The extension manager can be used for several purposes. You can use it to enhance Railo, install, update or remove certain applications. It is the global resource for you when you want to extend your Railo system.



## 2. Extension Provider (webservice)

Extensions are provided by the Extension Provider. A Extension Provider is a webservice that obeys a certain interface described below.

Extension Provider can easily be registered in the Railo Administrator. All you have to do is to enter and save the URL address pointing to it.

### Extension (experimental) - Providers

Please note, that only pages processed by Railo are aware of these mappings (cfm, cfml, cfc). If you want to use files not processed by Railo for these special mapping directories, you have to add virtual mappings to these directories to your application server.

<input type="checkbox"/> URL	Check
<input type="checkbox"/> <input type="text" value="http://www.railo-technologies.com/ExtensionProvider.cfc"/>	<input type="checkbox"/>
<input type="checkbox"/> <input type="text" value="http://localhost:8080/ext/ExtensionProvider.cfc"/>	<input type="checkbox"/>
<input type="checkbox"/> <input type="text" value="URL to the ExtensionProvider incl. path"/>	<input type="checkbox"/>

Let's have a look at the interface of the extension providers. As already said it is a webservice that can be programmed in any given technology. In our examples we are using a CFC (CF Component).

### 2.1. Interface

The interface of the Extension Provider contains two public methods:

- `getInfo(): struct`  
Returns information about the provider
- `listApplications():query`  
Lists all the applications provided by the provider

#### getInfo

This method returns information about the extension provider itself. It returns a struct that contains the following Data (still under construction):

Key	Description
title	Title of the extension provider
description	Description of the extension provider
image	Link to an image
url	URL for more information
mode	Defines how the Information of the ExtensionProvider is cached in the client (Railo Administrator). Valid values are: <ul style="list-style-type: none"> <li>• <code>develop</code> does not cache the result of the extension provider</li> </ul>

	<ul style="list-style-type: none"> <li>production (or no value) caches the result in the session scope of the consumer</li> </ul>
--	---------------------------------------------------------------------------------------------------------------------------------------

### listApplications

This method returns a query with all available Extensions. One row represents one extension. Below the contents of the columns of the query are described.

Key	Description
id	Identifier of the extension
name	Name of the extension, must be a valid CFML variable name
label	Display name of the extension
description	Extension description
author	Author of the extension
codename	Codename of the extension
video	Video for the extension (HTTP Link)
image	Image for the extension (HTTP Link)
support	Support Link of the extension
documentation	Documentation link of the extension
forum	Link to a forum
mailinglist	Link to a mailing list of the extension
network	Link to networking
created	When has the extension been created
version	Version of the extension
category	Category of the extension
download	Link to the extension itself (more on this below)
type	The type of the extension. Is it usable in the Server or Web Administrator. Valid values are: server,web,all – web is default

### Sample code as CFC

```

<cfcomponent>
    <cffunction name="getInfo" access="remote" returnType="struct">
        <cfset var info=struct{}>
        <cfset info.title="Railo">
        <cfset info.description="This is the Railo Extension Provider">
        <cfset info.image="http://www.railo.ch/railo.jpg">
        <cfset info.url="http://www.railo.ch">
        <cfreturn info>
    </cffunction>

    <cffunction name="listApplications" access="remote" returnType="query">
        <cfset var apps=queryNew(
            'id,name,label,description,version,category,image,download,paypal,type,author,codename,video,support,documentation,forum,mailinglist,network,created'>

            <cfset QueryAddRow(apps)>
            <cfset QuerySetCell(apps,'id','100')>
            <cfset QuerySetCell(apps,'name','galleon2')>
            <cfset QuerySetCell(apps,'label','Galleon ColdFusion Forums')>
            <cfset QuerySetCell(apps,'description','')>
            <cfset QuerySetCell(apps,'author','Raymond Camden')>
            <cfset QuerySetCell(apps,'support','http://galleon.riaforge.org/')>
            <cfset QuerySetCell(apps,'documentation','http://galleon.riaforge.org/')>
            <cfset QuerySetCell(apps,'created',CreateDateTime(2008,8,6,41,0))>
            <cfset QuerySetCell(apps,'version','2.2.3')>
            <cfset QuerySetCell(apps,'category','Forum')>
            <cfset QuerySetCell(apps,'download','http://dev.railo.ch/ext/galleon2/galleon.rep')>
            <cfset QuerySetCell(apps,'image','http://dev.railo.ch/glogo.png')>

            <cfreturn apps>
        </cffunction>
</cfcomponent>
    
```

The most important column here is *download*, since it contains the link to the extension itself.



### 3. Extension consumer

The extension consumer is one of the two Railo administrators. Extensions are available either for the Web Administrator, for local applications and extensions or for the Server Administrator, for global ones (like CFVIDEO). How the call of the extension provider inside the administrators is built internally is not relevant. In this section only it's functionality is described. The Railo Administrator contains in the left menu a section called "Extensions" which contains two links:

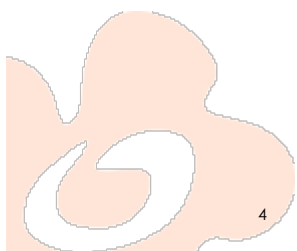
- Applications
- Providers

Under Applications you can install, uninstall or (under construction) buy several functionalities, applications or services. The Railo Administrator queries all extension providers for all their extensions and lists them according to the filter criteria you enter.

By clicking on the detail link you can receive additional information about a certain extension that you find interesting.

Click on the install (remove or update) button in order to install a specific extension. After the click just follow the information displayed in the Railo Administrator.

"Providers" allows you to manually add the URL of other Extension Providers (next to Railo Technologies) that provide extensions for applications. Just enter the URL provided by your chosen extension provider and update the list.



## 4. Extension

Let's have a closer look at a Railo extension. An extension is delivered as a zip file, which next to the application itself, must contain the two following files:

- `install.cfc`  
The `install.cfc` contains the installation code and methods that can be used in order to create dynamic content of the installation form.
- `config.xml`  
The `config.xml` contains a definition of form entries of the installation form.

### 4.1. `install.cfc`

The `install.cfc` is the main file for the installation. Next to the code that is used for the installation process itself it contains methods that can be used together with the `config.xml` in order to dynamically populate the form items of the install form or to dynamically generate form items as well.

The main task of the component `install.cfc` is to install, uninstall or update an extension. The installation of an extension is „application based“, which means that the installation is not done by Railo but by the extension itself. Railo just tells the extension to install itself and passes the configuration data to it. Therefore the `install.cfc` is the communication interface between Railo and the extension. This interface looks like follows:

Public methods of `install.cfc`

Method	Description
<code>Validate(struct: error, string: path, struct: config, numeric: step): void</code>	The method <code>validate</code> is called after a user has submitted the installation form. If the installation form consists out of several steps (wizard), the <code>validate</code> method is called after each step. <code>Validate</code> will check the data. In case an entry is not valid or perhaps missing, a corresponding error can be thrown. By passing the argument <code>error</code> the extension notifies Railo about the errors of the validation. The other arguments help the method to validate the data. More about these arguments further below.
<code>install(struct: error, string: path, struct:config): string:message</code>	The method <code>install</code> is called by Railo after the last step of the form has been submitted and the <code>validate</code> method has been successfully executed. The method is responsible for the installation of the extension by using the data

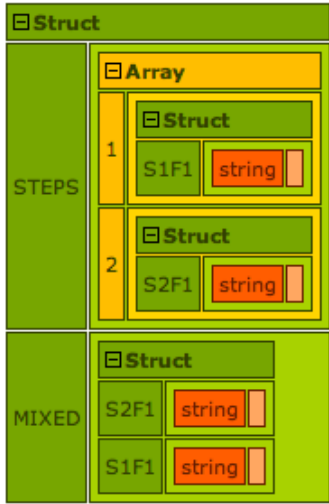
<pre>update(struct: error,        string: path,        struct: config,        struct: previousConfig): string: message</pre>	<p>of the passed arguments (more on these arguments further below). How this method executes its task is up to the method itself. The method <i>update</i> is called by Railo after the last step of the form has been submitted and the validate method has been successfully executed. The method is responsible for updating an extension by using the data of the passed arguments (more on these arguments further below). How this method executes its task is up to the method itself.</p>
<pre>uninstall(string: path,           struct:formerConfig): string:message</pre>	<p>The method <i>uninstall</i> is called by Railo when a user has clicked on the <i>uninstall</i> button. The method is responsible for uninstalling an extension by using the data of the passed arguments (more on these arguments further below).</p>

## Arguments

Argument	Description
struct: error	<p>The argument <i>error</i> contains a struct that gathers all occurred errors. In addition to this struct a regular exception can be thrown as well. The structure contains the following keys:</p> <ul style="list-style-type: none"> <li>• <i>fields</i> a struct that contains all erroneous fieldnames as keys. The value is the error message itself.</li> <li>• <i>common</i> common error message, that is not related to a field</li> </ul>
string: path	<p>The argument <i>path</i> contains the path to the root directory of the extension and therefore allows you to access all the files of the extension zip.</p>
struct: config	<p>The <i>config</i> struct contains all the data of the form, that has been entered by the user. It is structured as follows:</p> <ul style="list-style-type: none"> <li>• <i>mixed</i> contains all values of the input fields, from all steps merged into a struct. Caution: Data is overwritten, in case two steps contain a field with the same name</li> <li>• <i>step</i> contains an array of all the steps. An array element contains all input field values of the corresponding step</li> </ul> <p>Example: Let's assume we have a form that consists out of two steps. The first step defines the field <i>s1f1</i> and the second the field <i>s2f1</i>. If you dump the config argument, you receive the following output:</p>

struct:  
previousConfig

numeric: step



The argument previousConfig is analog to config. It just contains the data of the previous installation. In case of an update this might be important in order to have information about already entered Data.

This argument contains the current step of the form. This helps defining the data that needs to be validated.

Return Value

Argument	Description
message	Message is a text that is displayed after the successful installation/update.

## 4.2. config.xml

The config.xml describes the data entry forms for the installation process. It is build as a XML structure which can contain the following data tags:

```
<config>
  L <step>
    L <group>
      L <item>
        L <option>
```

Above you can see the hierarchy structure of the possible tags.

The root tag of the XML is named <config>. More on the <config> tag further below.

### 4.2.1. step

Hierarchy



```
<config>
  L <step>
```

A form can consist out of several steps, like a wizard. This is why the `<config>` tag contains any number of `<step>` tags.

**config.xml:**

```
<config>
  <step label="First Step" description="This is the first step"/>
  <step label="Second Step" description="This is the second step"/>
</config>
```

Each step describes a page of the installation form. Above the `config.xml` defines two pages that provide a label and description of the page with the corresponding attributes *label* and *description*.

#### 4.2.2. group

Next to pages, the `<group>` tag can group several form elements on a page.

**config.xml:**

```
<config>
  <step label="First Step" description="This is the first step">
    <group label="First Group" description="This is the first group"/>
    <group label="Second Group" description="This is the second group"/>
  </step>
</config>
```

The group item itself allows the attributes *label* and *description* for detailed information.

#### 4.2.3. item

The single fields are described with the tag `<item>`.

**config.xml:**

```
<config>
  <step label="First Step" description="This is the first step">
    <group label="First Group" description="This is the first group">
      <item label="First Item" description="This is the first item"
        name="firstItem" type="text">
        First Item Value
      </item>
    </group>
  </step>
</config>
```

Next to the usual attributes *label* and *description*, the tag `<item>` allows the attribute *name* for the name of the form field and *type* for the type of the item. A *value* as the default value can be used as well.

#### 4.2.4. option

An item might contain a sub element named option which contains the different options for a certain item type (like select).

**config.xml:**

```
<config>
  <step label="First Step" description="This is the first step">
    <group label="First Group" description="This is the first group">
      <item label="First Item" description="This is the first item"
        name="firstItem" type="text">
        First Item Value
      </item>
      <item label="Second Item" description="This is the second item"
        name="secondItem" type="radio">
        <option selected="true" label="First Option"
          description="This is the first option">
          first
        </option>
        <option label="Second Option"
          description="This is the second option">
          second
        </option>
      </item>
    </group>
  </step>
</config>
```

Inside the second item tag in the above config.xml the <option> has been used. It can be used when item is of type *radio*, *checkbox* or *select*. Of course this tag allows the attributes *label* and *description* as well. In addition next to the value the attribute *selected* can be used in order to mark the selected option.

The above config.xml would be generating the form below:

#### **First Step**

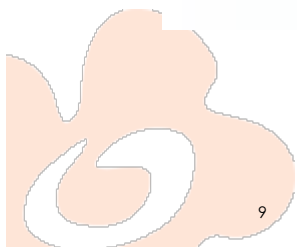
This is the first step

#### **First Group**

This is the first group

First Item	<input type="text" value="First ItemValue"/> This is the first item
Second Item	<input type="radio"/> First Option This is the first option <input checked="" type="radio"/> Second Option This is the second option This is the second item

**install**



## 4.3. dynamic content

The above examples show how to build a static form for the extension. Now how do we fill the form with dynamic content or even dynamic form fields?

### 4.3.1. evaluate

The easiest way to use dynamic content is to use evaluate functionality:

```
...
<item label="Root" name="root" type="text">
    evaluate:expandPath('{web-root-directory}')
</item>
...
```

Everything you can do in Railo with the Build In Function `evaluate()` can be used in the `config.xml`. So the above code would return a string which could be displayed in Railo like this:

```
<cfdump var="#evaluate("expandPath('{web-root-directory}')" )#">
```

### 4.3.2. dynamic

When you use the attribute *dynamic*, you can define a method that is called in order to fill the value of this element. The value must be a valid method name of the component `install.cfc`.

```
...
<item dynamic="fillRoot" label="Root" name="root" type="text"/>
...
```

The corresponding method in the `install.cfc` might look like this:

```
install.cfc:
<cfcomponent>
    <cffunction name="fillRoot" returntype="void" output="no">
        <cfargument name="item" required="yes" >
        <cfset item.setValue(now())>
    </cffunction>
</cfcomponent>
```

For all the tags in the XML there is an equivalent CFC component that can be filled with data. The CFC hierarchy is described further below.

Inside the method you can create item options as well:

```
install.cfc:
<cfcomponent>
    <cffunction name="fillOther" returntype="void" output="no">
        <cfargument name="item" required="yes" >
        <cfset item.createOption(value="first",label:"First",selected:true)>
        <cfset item.createOption(value="second",label:"Second")>
    </cffunction>
</cfcomponent>
```

The attribute dynamic can be used in all tags defined in the xml, including the root tag *config* (except of course the *option* tag).

```

...
<group dynamic="fillGroup" label="dynamic Group">
...

install.cfc:
<cfcomponent>
  <cffunction name="fillGroup" returntype="void" output="no">
    <cfargument name="group" required="yes" >
    <cfset item=group.createItem(type:'radio',name:'happy',label:'Happy?')>
    <cfset item.createOption(value:true,label:'Yes',selected:true)>
    <cfset item.createOption(value:false,label:'No')>
    <cfset item=group.createItem(type:'text',name:'fullname',label:'Fullname',
      description:'What is your Fullname?')>
    </cffunction>
  </cfcomponent>

```

## 4.4. config.xml matrix

### 4.4.1. Tag config

Attribute	Description
dynamic	This attribute contains a method name of the install.cfc. It can completely fill the form dynamically. The method receives the loaded ExtensionConfig CFC as an argument.

### 4.4.2. Tag step

Attribute	Description
dynamic	This attribute contains a method name of the install.cfc. It can completely fill the step of the installation form dynamically. The method receives the loaded ExtensionStep CFC as an argument.
label	Label of the step
description	Description of the step

### 4.4.3. Tag group

Attribute	Description
dynamic	This attribute contains a method name of the install.cfc. It can completely fill the group on a form step dynamically. The method receives the loaded ExtensionGroup CFC as an argument.
label	Label of the group

description	Description of the group
-------------	--------------------------

#### 4.4.4. Tag item

Attribute	Description
full-dynamic	This attribute contains a method name of the <code>install.cfc</code> . In contrary to the attribute <i>dynamic</i> , the method does not receive the current item as a parameter, but the parent group element (ExtensionGroup CFC). This allows inserting items in an existing group dynamically.
dynamic	This attribute contains a method name of the <code>install.cfc</code> . It can completely fill the item of the group on a form step dynamically. The method receives the loaded ExtensionItem CFC as an argument.
label	Label of the item
description	Description of the item
name	Name of the item
type	Type of the item. Valid values are: <ul style="list-style-type: none"> <li>• text</li> <li>• radio</li> <li>• checkbox</li> <li>• select</li> <li>• hidden</li> <li>• password</li> </ul>
value	Default value of the item. The default value can be defined in the body of the tag as well.

#### 4.4.5. Tag option

Attribute	Description
label	Label of the option
description	Description of the option
selected	Sets whether the option is selected or not. Valid values are <i>true</i> and <i>false</i>
value	Default value of the item. The default value can be defined in the body of the tag as well.

## 4.5. Extension CFCs Matrix

The methods marked red are system methods and should therefore not be called in the installation application. The hierarchy looks like follows:

```

ExtensionConfig.cfc
├─ ExtensionStep.cfc
│   └─ ExtensionGroup.cfc
│       └─ ExtensionItem.cfc
│           └─ ExtensionOption.cfc
  
```

### 4.5.1. ExtensionConfig

```
ExtensionConfig.cfc
```

Method	Description
createStep(string: label, string: decription): ExtensionStep	Creates and links a new step.
getSteps(): ExtensionStep[]	Returns all ExtensionSteps of the ExtensionConfig.

### 4.5.2. ExtensionStep

```

ExtensionConfig.cfc
├─ ExtensionStep.cfc
  
```

Method	Description
<b>init(string: label, string: description): ExtensionStep</b>	Initializes the ExtensionStep CFC. Is called on creation.
createGroup(string: label, string: decription): ExtensionStep	Creates and links a new group.
getGroups(): ExtensionGroup[]	Returns all ExtensionGroups of the ExtesnionStep.
getLabel(): string	Returns the label of the step .
getDescription(): string	Returns the description of the step.
setLabel(string: label): void	Sets the label
setDescription(string: label): void	Sets the description

### 4.5.3. ExtensionGroup

```

ExtensionConfig.cfc
├─ ExtensionStep.cfc
│   └─ ExtensionGroup.cfc
  
```

Method	Description
init(string: label, string: description): ExtensionGroup	Initializes the ExtensionGroup CFC. Is called on creation.
createItem(string: type, string: name, string: value, boolean: selected, string: label, string: decription): ExtensionItem	Creates and links a new item.
getItems(): ExtensionItem[]	Returns all ExtensionItems of the ExtesnionGroup.

getLabel(): string	Returns the label of the group.
getDescription(): string	Returns the description of the group.
setLabel(string: label): void	Sets the label
setDescription(string: label): void	Sets the description

#### 4.5.4. ExtensionItem

```

ExtensionConfig.cfc
├─ ExtensionStep.cfc
│   └─ ExtensionGroup.cfc
│       └─ ExtensionItem.cfc
  
```

Method	Description
init(string: type, string: name, string: value, boolean: selected, string: label, string: decription): ExtensionItem	Initializes the ExtensionItem CFC. Is called on creation.
createOption(string: value, boolean: selected, string: label, string: decription): ExtensionOption	Creates and links a new option.
getOptions(): ExtensionOption[]	Returns all ExtensionOptions of the ExtesnionItem.
getType(): string	Returns the type of the Item. Valid values are: <ul style="list-style-type: none"> <li>• text</li> <li>• radio</li> <li>• checkbox</li> <li>• select</li> <li>• hidden</li> <li>• password</li> </ul>
getName(): string	Returns the name of the item
getValue(): string	Returns the value of the item
getSelected(): boolean	Returns whether the item is selected or not. Can be used when type is: select,radio,checkbox
getLabel(): string	Returns the label of the item
getDescription(): string	Returns the description of the item
setType(string: type): void	Sets the type of the item. Valid values are: <ul style="list-style-type: none"> <li>• text</li> <li>• radio</li> <li>• checkbox</li> <li>• select</li> <li>• hidden</li> <li>• password</li> </ul>
setName(string: name): void	Sets the name
setValue(string: value): void	Sets the value

setSelected(boolean: selected): void	Sets whether a certain option should be selected or not.
setLabel(string: label): void	Sets the label
setDescription(string: label): void	Sets the description

#### 4.5.5. ExtensionOption

```

ExtensionConfig.cfc
├─ ExtensionStep.cfc
│   └─ ExtensionGroup.cfc
│       └─ ExtensionItem.cfc
│           └─ ExtensionOption.cfc
  
```

Method	Description
<b>init(string: value, boolean: selected, string: label, string: decription): ExtensionOption</b>	Initializes the ExtensionOption CFC. Is called on creation.
getValue(): string	Returns the value of the item
getSelected(): boolean	Returns whether the item is selected or not.
getLabel(): string	Returns the label of the item
getDescription(): string	Returns the description of the item
setValue(string: value): void	Sets the value of the option
setSelected(boolean: selected): void	Sets whether the item is selected or not.
setLabel(string: label): void	Sets the label
setDescription(string: label): void	Sets the description

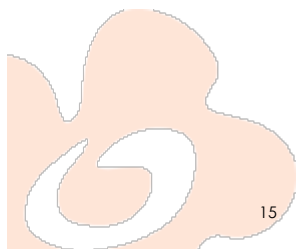
#### 4.6. Support components

Railo offers a couple of components that can be extended in the install.cfc. Diese helfen gebräuchliche Abläufe wie das erstellen eines Mapping oder entpacken eines zip zu vereinfachen

es gibt aber auch CFC die einem praktisch den kompletten Ablauf der Installation abnehmen, muss man also nur noch in der ein paar Konfigurationen vornehmen

mehr informationen und Downloads der CFC finden Sie hier ([Link folgt, under construction](#))

At the moment there exist only two of these support components. The number of these components will grow over time and therefore we will not describe these components here. Instead of this documentation please check this webpage for details: ().



## 5. Tips and tricks

### 5.1. Installed applications

The loaded extensions can be found in the directory `/yourwebroot/WEB-INF/railo/extension`. The directory structure under this folder reflects the extension provider, the application and the version of the extension you have installed. You can use any ZIP-program in order to inspect the installed extensions.

### 5.2. Application list, update and caching

The available applications listed in the extension manager are not downloaded every time. The list is cached by Railo depending on the mode of the extension provider. Two modes are possible: `develop` or `production`. In case the mode of the extension provider is set to `"develop"` the list of the application is downloaded every time the `"extensions"` menu in the Railo Administrator is clicked. The list is only cached in the request scope. If on the other hand the mode is set to `"production"` the list is only loaded once per session. So the list is cached in the session scope of the user.

