# Workspace Desktop Edition Developer's Guide

## Use Customizable Commands

12/13/2025

# Use Customizable Commands

| | |
|---|---|
|  | **Purpose:** To provide information about the customizable commands available in the Workspace Desktop Edition. |

## Contents

## Before You Start

- The command system is based on the chain of command (or chain of responsibility) design pattern.
- All the code snippets in this page are extracted from the Genesyslab.Desktop.Modules.ExtensionSample source files.
- In addition to this page, read:
  - Creating a New Module
  - Deploying Your Custom Module into the Genesys Out-Of-The-Box Application

## Inserting a Command in a Chain

Each element of command is unique across the given chain. You can use the `ICOmmandManager.InsertCommandToChainOfCommandAfter()` method to insert your command after a specific command by passing its name. The following code snippet shows how to insert the element of command "CloseSample" in the chain of command "BundleClose" after the element of command "IsPossibleToClose":
**[C#]**

```
commandManager.InsertCommandToChainOfCommandAfter("BundleClose", "IsPossibleToClose", new List<CommandActivator>()
  {
      new CommandActivator()
      {
        CommandType = typeof(CloseSampleCommand), Name = "CloseSample"
      }
  });
```

## Creating a Command

Creating a new command is considered an advanced topic. Genesys recommends that you do so with caution. Consider doing this in consultation

with the development community:

- Genesys Engage DevFoundry
- Genesys Community

The following example illustrates how to create your own commands by using Genesys best practices. For each new command, create a class which implements the `IElementOfCommand` interface. After creating the command, you must add it to a chain of command in your module (see Creating a New Module). The custom command created in the following step-by-step example displays a confirmation dialog before executing the `ReleaseCall` command.

1. Create the elementary command class: **[C#]**

```
// File: CustomCommand.cs
namespace Genesyslab.Desktop.Modules.ExtensionSample.CustomCommand
{
  // Custom command which prompts a confirmation dialog before executing the ReleaseCall command
  class BeforeReleaseCallCommand : IElementOfCommand
  {
    readonly IObjectContainer container;
    ILogger log;
    public BeforeReleaseCallCommand(IObjectContainer container)
    {
      this.container = container;
      // Initialize the trace system
      this.log = container.Resolve<ILogger>();
      // Create a child trace section
      this.log = log.CreateChildLogger("BeforeReleaseCallCommand");
    }
    public string Name { get; set; }
    public bool Execute(IDictionary<string, object> parameters, IProgressUpdater progress)
    {
      // To go to the main thread
      if (Application.Current.Dispatcher != null && !Application.Current.Dispatcher.CheckAccess())
      {
        object result = Application.Current.Dispatcher.Invoke(DispatcherPriority.Send,
                                    new ExecuteDelegate(Execute), parameters, progress);
        return (bool)result;
      }
      else
      {
```

```
        log.Info("Execute");
        // Get the parameter
        IInteractionVoice interactionVoice = parameters["CommandParameter"] as IInteractionVoice;
        // Prompt the alert dialog
        return MessageBox.Show("Do you really want to release this call?\r\nThe call",
          "Release the call?", MessageBoxButton.YesNo) == MessageBoxResult.No;
      }
    }
    delegate bool ExecuteDelegate(IDictionary<string, object> parameters, IProgressUpdater progressUpdater);
  }
}
```

2. Create the chain of command in the `Module` initialization by using the `CommandManager`:
**[C#]**

```
// File: ExtensionSampleModule.cs
ICommandManager commandManager = container.Resolve<ICommandManager>();
// Add a command before the release call
// Method 1:
commandManager.CommandsByName["InteractionVoiceReleaseCall"].Insert(0, new CommandActivator() {
    CommandType = typeof(BeforeReleaseCallCommand), Name = "BeforeReleaseCall" });
// Method 2 (recommended):
commandManager.InsertCommandToChainOfCommandBefore("InteractionVoiceReleaseCall", "ReleaseCall",
  new CommandActivator() { CommandType = typeof(BeforeReleaseCallCommand), Name = "BeforeReleaseCall" });
```

3. You can add several commands to a chain of command. The order of execution follows the order in which the commands are added. `BeforeReleaseCallCommand` is executed before `ReleaseCallCommand`, for example: **[C#]**

```
commandManager.AddCommandToChainOfCommand("InteractionVoiceReleaseCall",
     new List<CommandActivator>()
     {
       new CommandActivator() { CommandType = typeof(BeforeReleaseCallCommand), Name = "BeforeReleaseCall" },
       new CommandActivator() { CommandType = typeof(ReleaseCallCommand), Name = "ReleaseCall" }
     });
```

4. Finally, execute the chain of command by using parameters, as shown in the following example (defined here: Command list): **[C#]**

```
IDictionary<string, object> parameters = new Dictionary<string, object>();
parameters.Add("CommandParameter", interaction);
parameters.Add("Reasons", reasons);
parameters.Add("Extensions", extensions);
```

```
commandManager.GetChainOfCommandByName("InteractionVoiceReleaseCall").Execute(parameters);
```

## Multiple Commands and Overlapping

When you pass several commands to a given chain, they share the parameters which have identical names. This can lead to over-lapping issues when you execute the command. To by-pass this issue, make sure that your parameters are correct before your application executes the command. For instance, consider using the `Command1` and `Command2` of `MyChain`:

| Chain of Command | Default commands | Parameters |
|---|---|---|
| MyChain | Command1 | • Parameter1: IInteractionChat<br>• Parameter2: KeyValueCollection |
| | Command2 | • Parameter1: IInteractionChat<br>• Parameter3: KeyValueCollection |

- IInteractionChat: Genesyslab.Desktop.Modules.OpenMedia.Model.Interactions.Chat.IInteractionChat
- KeyValueCollection: Genesyslab.Enterprise.Commons.Collections.KeyValueCollection

When you execute `MyChain`, you must pass all the parameters of Command1 and Command2. Parameter1 is shared amongst Command1 and Command2.
**[C#]**

```
IDictionary<string, object> parameters = new Dictionary<string, object>();
parameters.Add("Parameter1", interaction);
parameters.Add("Parameter2", reasons);
parameters.Add("Parameter3", extensions);
commandManager.GetChainOfCommandByName("MyChain").Execute(parameters);
```

## Subscribing to Events

When you are creating custom commands there is no “command” that you can intercept; however, there is an “event” that you can subscribe to that will notify you when the interaction bar tab selection is changed. The following sample demonstrates how this is done: CustoInteractionBarSelectHandler.zip