**User Manual**

Version 0.0.01

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 08-06-2017 | 0.0.01 | First use case document | Yunpeng Xu |

**Formal Approval**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Issues** | **Approved by** |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_Toc487995851)

[1.1 Purpose 4](#_Toc487995852)

[1.2 Definitions, Acronyms, Abbreviations 4](#_Toc487995853)

[2. User Guide 5](#_Toc487995854)

[3. Application Programing Interface (API) offered by SCN Library 6](#_Toc487995855)

[4. SCN Services 16](#_Toc487995856)

# **Introduction**

The Turtlebot project aims at building a framework to support configuration/reconfiguration for mobile robots at run time.

## Purpose

The purpose of this document is to provide an understanding about the usage of the framework to the developers who will be using this framework for their respective nodes.

## Definitions, Acronyms, Abbreviations

|  |  |  |
| --- | --- | --- |
| **DAA** | **Term** | **Definition** |
| ROS | Robot Operating System |  |
| Bot | Robot |  |
| TBD | To Be Discussed |  |
| SCN | System Control Node |  |
| API | Application Programing Interface |  |

# **Code Guide**

1. **Precondition:**

The framework requires and installation of ROS Indigo over Ubuntu 14.04. The framework was developed and tested for this combination.

1. **Usage of our framework**

A node that uses the framework shall incorporate the bellow mentioned code snippets in order to correctly employee the framework

* 1. **Main function**

The main function of a node shall call the “scnInit” in place of the normal “ros::init”.

A sample node -

This node provides a service named “demoNode1TestService”. Also, the main function overrides two default mechanism in ROS. One is signal handler for SIGINT, and the other one is XMLRPC for shutdown. The reason of doing this is that the framework needs to update the dependency tree after one node is killed either by using ctrl-C or “rosnode kill” commands. There’s a global parameter called “g\_request\_shutdown” that will track whether the user triggers the node kill via the above two methods.

int main(int argc, char \*\* argv) {

gNodeName = "demoNode1";

// Override SIGINT handler

ros::scnInit(argc, argv, gNodeName, ros::init\_options::NoSigintHandler, saveStateCb, reconModeCb, loadStateCb);

signal(SIGINT, demoNodeSigIntHandler);

// Override XMLRPC shutdown

ros::XMLRPCManager::instance()->unbind("shutdown");

ros::XMLRPCManager::instance()->bind("shutdown", shutdownCallback);

ros::SCNNodeHandle n;

// service provided for demo node 1

std::string testService1 = "demoNode1TestService";

ros::SCNServiceServer testService = n.advertiseService(gNodeName, testService1, demoNode1CallBack);

//ros::spin();

while (!g\_request\_shutdown) {

// Do non-callback stuff

ros::spinOnce();

usleep(100000);

}

// Do pre-shutdown tasks

ros::shutdown();

return 0;

}

* 1. **Override sigint and shutdown XMLRPC**

Below is the code for overriding the default SIGINT handler and XMLRPC shutdown callback.

// Replacement SIGINT handler

void demoNodeSigIntHandler(int sig) {

unregisterDependencyToSCN();

g\_request\_shutdown = 1;

}

// Replacement "shutdown" XMLRPC callback

void shutdownCallback(XmlRpc::XmlRpcValue& params, XmlRpc::XmlRpcValue& result) {

int num\_params = 0;

if (params.getType() == XmlRpc::XmlRpcValue::TypeArray)

num\_params = params.size();

if (num\_params > 1)

{

std::string reason = params[1];

ROS\_WARN("Shutdown request received. Reason: [%s]", reason.c\_str());

unregisterDependencyToSCN();

g\_request\_shutdown = 1; // Set flag

}

result = ros::xmlrpc::responseInt(1, "", 0);

}

* 1. **It is required to implement three callback functions –** 
     1. **Save State Callback**

Allows the node to save the state of the node when the node receives an “Enter reconfiguration mode” message. The node shall take a snapshot of the current state and save it to disk.

void saveStateCb(uint8\_t reconType) {

}

* + 1. **Load State Callback**

Allows the node to load state from previously saved state at launch time. This function is called if the SCN starts a node and wants it to load its state from a previously stored state snapshot.

void loadStateCb() {

ROS\_INFO("saveStateCb %s", \_\_FILE\_\_);

}

* + 1. **Load State Callback**

This is called once the node has saved its state. This function should only set flags and perform important operations that are required for a node to enter reconfiguration mode. The function should not implement any heavy operations or busy loops.

STATUS\_T reconModeCb(uint8\_t reconType, uint8\_t command) {

ROS\_INFO("Enter recon mode callback!\n");

ROS\_INFO("Leave recon mode callback!\n");

return SCN\_ST\_OK;

}

* 1. **Header files**

The framework provides the following header files that node developer can use:

#include <reconfigure/demoNodeService.h>

#include <scn\_library/systemControlRegisterService.h>

#include <scn\_library/scn\_utils.h>

#include <scn\_library/scn\_core.h>

#include <scn\_library/scn\_node\_handle.h>

#include <scn\_library/scn\_service\_client.h>

#include <scn\_library/scn\_service\_server.h>

#include <scn\_library/scn\_publisher.h>

#include <scn\_library/scn\_subscriber.h>

# **Test Guide**

This part will introduce how to run the newly developed use case and test it using the turtlebot.

1. **Compile the use case**
2. Update the CMakeList.txt



For more details about CMakeList.txt, please check the patch files in Github Repo for AMCL and Move\_Base.

1. Compile the code using catkin\_make

After updating source code of one node, and the corresponding CMakeList.txt, use “catkin\_make” in the package folder. Or, if you only want to compile two specific packages “package1”, “package2”, you can use the following command:

$ catkin\_make -DCATKIN\_WHITELIST\_PACKAGES="package1; package2"

If you want to revert back to building all packages, do the following:

$ catkin\_make -DCATKIN\_WHITELIST\_PACKAGES=""

1. **Test the code**
2. Start the roscore in one terminal

$> roscore

1. Start the reconfigure node in a new terminal

$> rosrun reconfigure reconfigure\_node

1. Start the demoNode in a new terminal

$> rosrun reconfigure demoNode1

1. Start the demoNode3 in a new terminal

$> rosrun reconfigure demoNode3

1. Trigger reconfigure

$> rosservice call /userInterfaceService '{reconType: 2, oldNodePackage: reconfigure, oldNode: demoNode1, newNode: demoNode2, newNodePackage: reconfigure}'