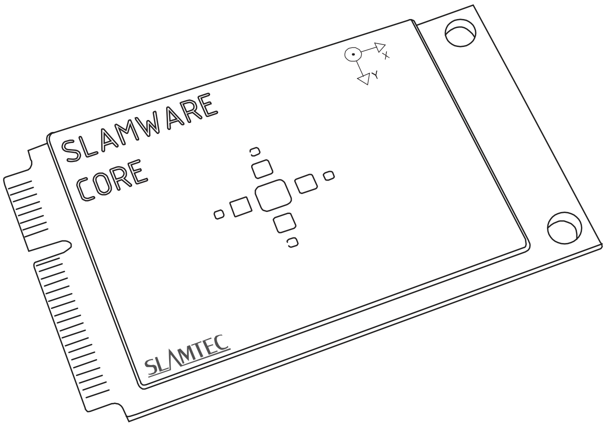
SLAMTEC

2016-11-23 rev.1.8

SLAMWARE

Modular Autonomous Robot Localization and Navigation Solution

API Reference Manual



www.slamtec.com

Shanghai Slamtec.Co.,Ltd

Directory structure

SLAMWARE SDK contains a lot of resources, source code and project files which you may use during the development. The directory structure is shown as below:

List	Description
Bin	Pre-compiled tool
Include	SDK Header file
Lib	Pre-compiled library file
Samples	Sample program
workspaces	Project file

Figure 1-1 SLAMWARE SDK Directory Structure

Header file structure

The directory of include contains the SLAMWARE SDK and the header files of all the libraries it depends on:

List	Description
boost	Boost 1.53.0
Eigen	Eigen Matrix library
rpos	SLAMWARE SDK related Header files

Figure 1-2 SLAMWARE SDK Header Files Structure

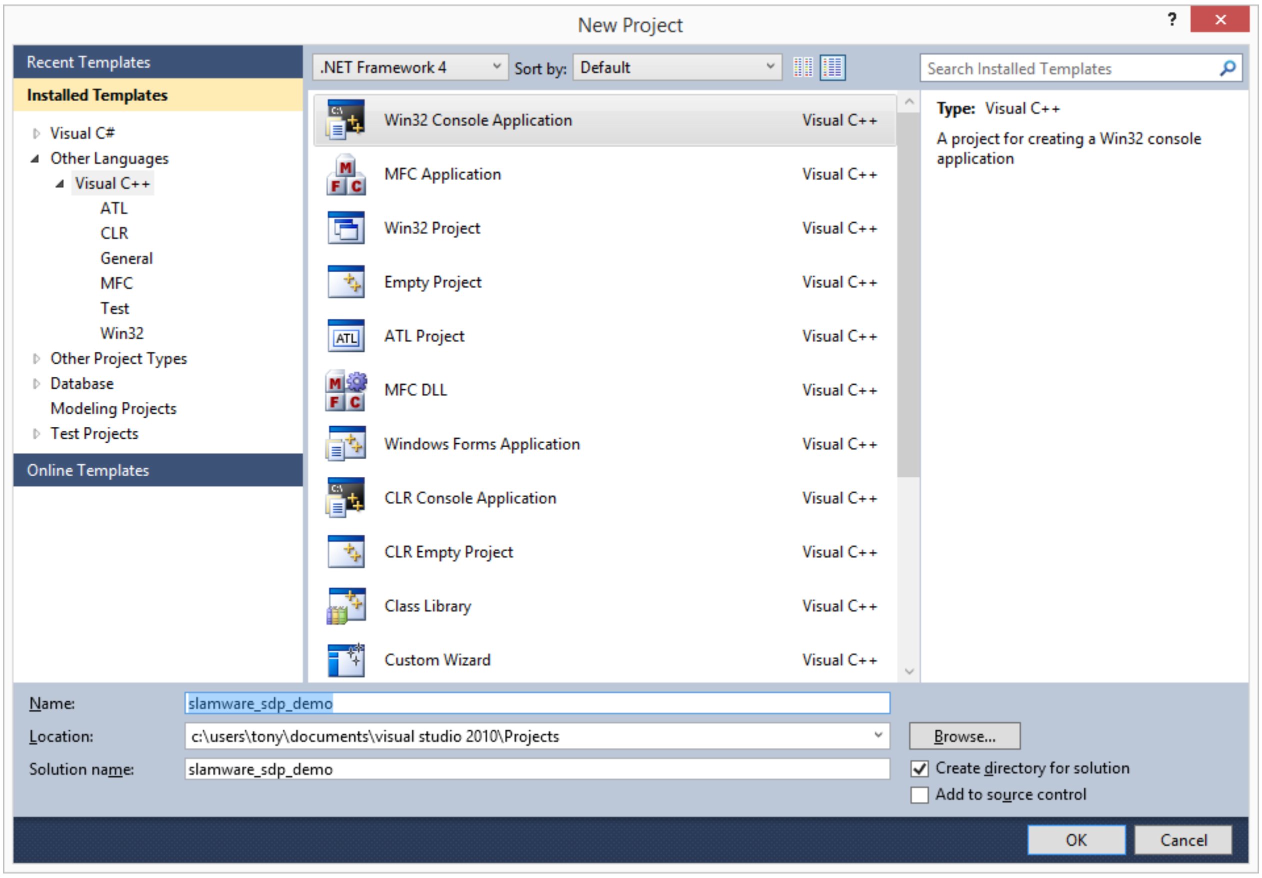
When you want to develop applications based on the Slamware SDK, the development environment needs to meet the following requirements:

- You need to install Visual Studio 2010 SP1 on your PC.

Hint: We uses Visual Studio 2010 SP1 to compile the pre-compiled library, so we cannot use other versions such as Visual Studio 2012 or 2013 to develop applications.

New Project

Step1 Open the Visual Studio 2010 and create project.



Type: Visual C++

A project for creating a Win32 console application

Name: slamware_sdp_demo

Location: c:\users\tony\documents\visual studio 2010\Projects

Solution name: slamware_sdp_demo

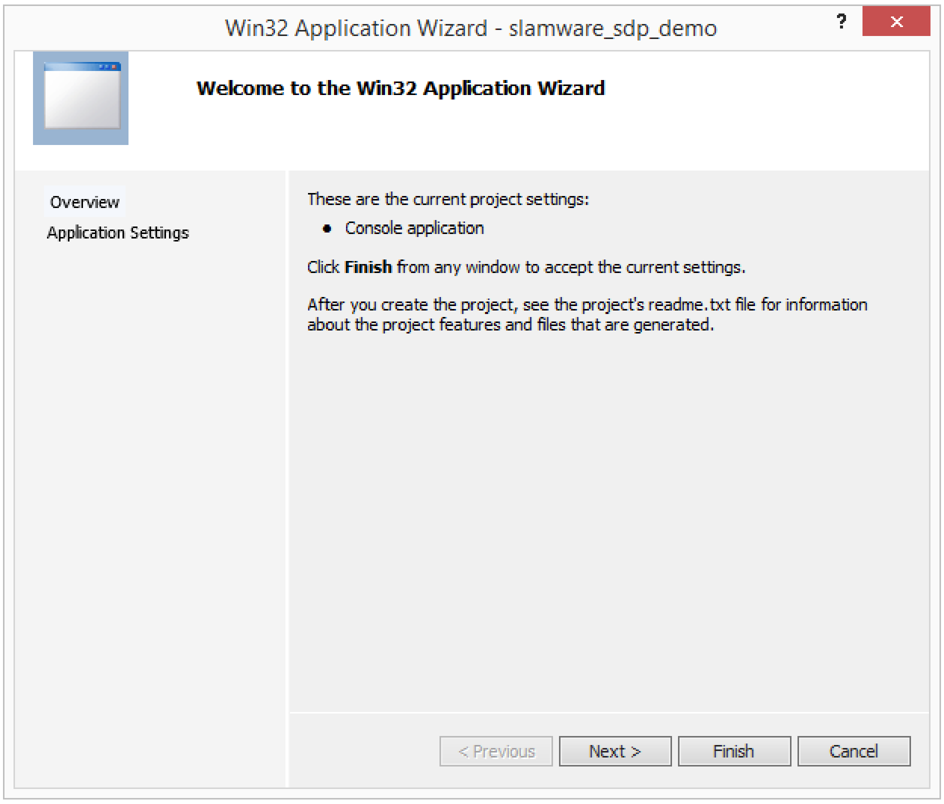
☑ Create directory for solution

☐ Add to source control

Figure 3-1 Create New Project

1. choose Visual C++ templates, and choose Win32 Console Application project.
2. enter the project Name such as slamware_sdp_demo.
3. click OK button.

Step2 Application settings



Win32 Application Wizard - slamware_sdp_demo

Welcome to the Win32 Application Wizard

These are the current project settings:

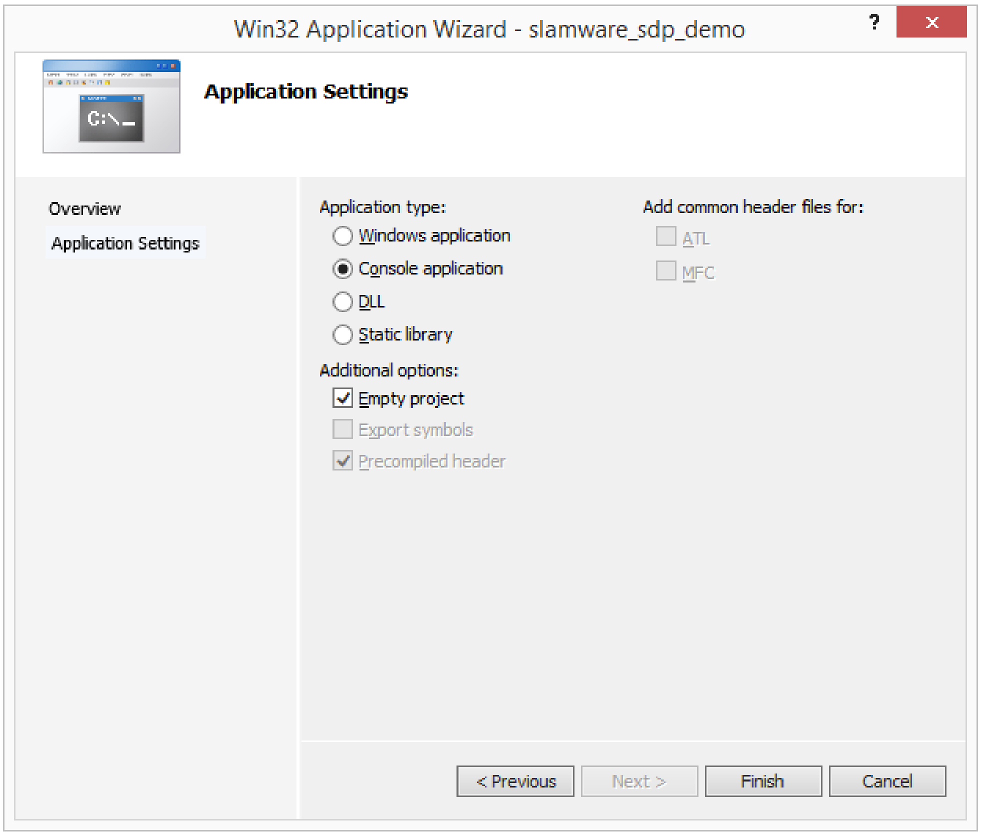
- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

Figure 3-2 Application Guide

Click Next button.



Win32 Application Wizard - slamware_sdp_demo

Application Settings

Application type:

○ Windows application
● Console application
○ DLL
○ Static library

Additional options:

☑ Empty project
☐ Export symbols
☑ Precompiled header

Add common header files for:

☐ ATL
☐ MFC

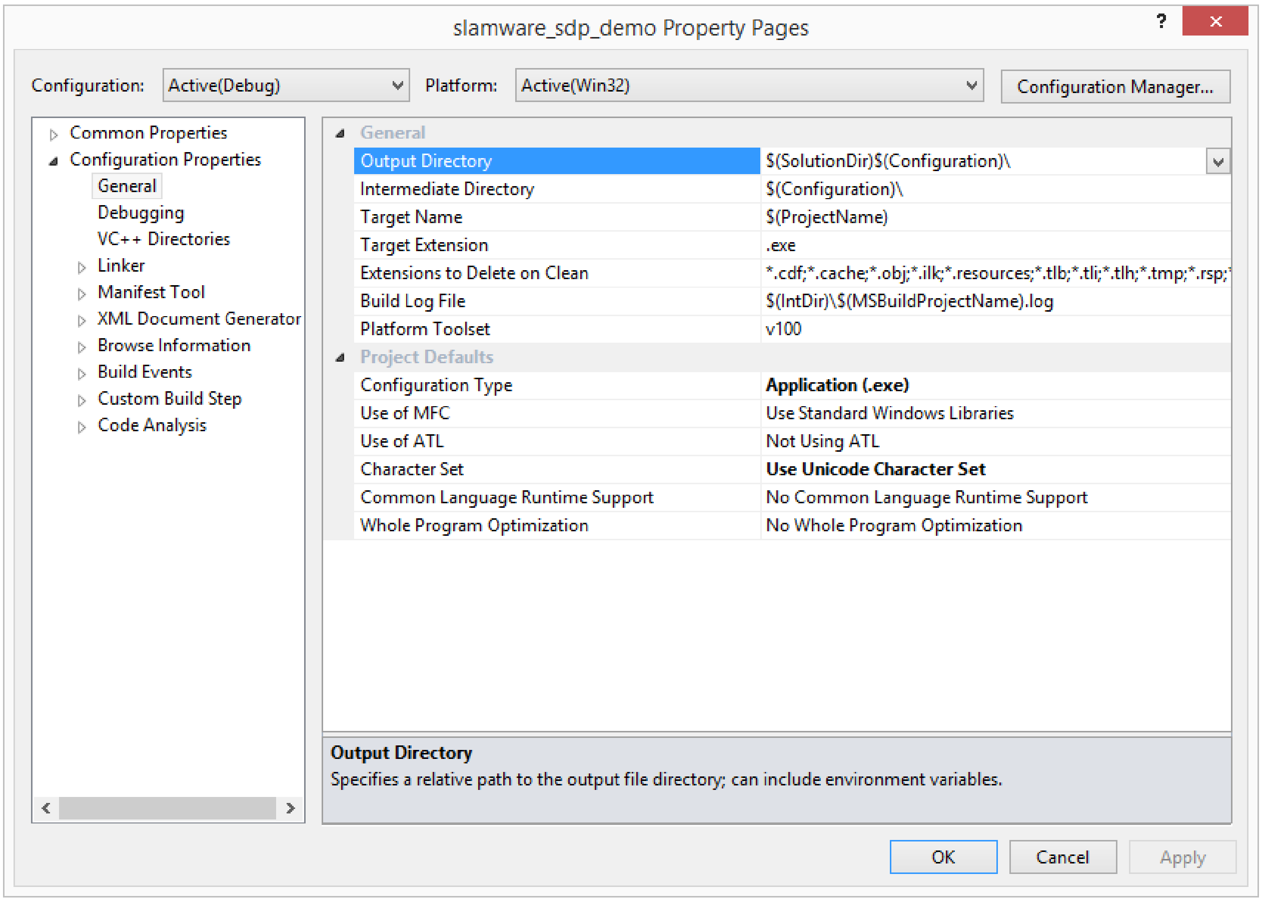
Figure 3-3 Application Settings

1. choose Console application in Application type.
2. check Empty project in Additional options.
3. click Finish button to finish creating project.

Configuration options

Step1 Open the Project Property Pages

Right click the new project in Solution Explorer, and choose Properties to open the property pages as below:



slamware_sdp_demo Property Pages

Configuration: Active(Debug)

Platform: Active(Win32)

General	
Output Directory	$(SolutionDir)$(Configuration)\
Intermediate Directory	$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.ilk;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;
Build Log File	$(IntDir)\$(MSBuildProjectName).log
Platform Toolset	v100
Project Defaults	
Configuration Type	**Application (.exe)**
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Character Set	**Use Unicode Character Set**
Common Language Runtime Support	No Common Language Runtime Support
Whole Program Optimization	No Whole Program Optimization

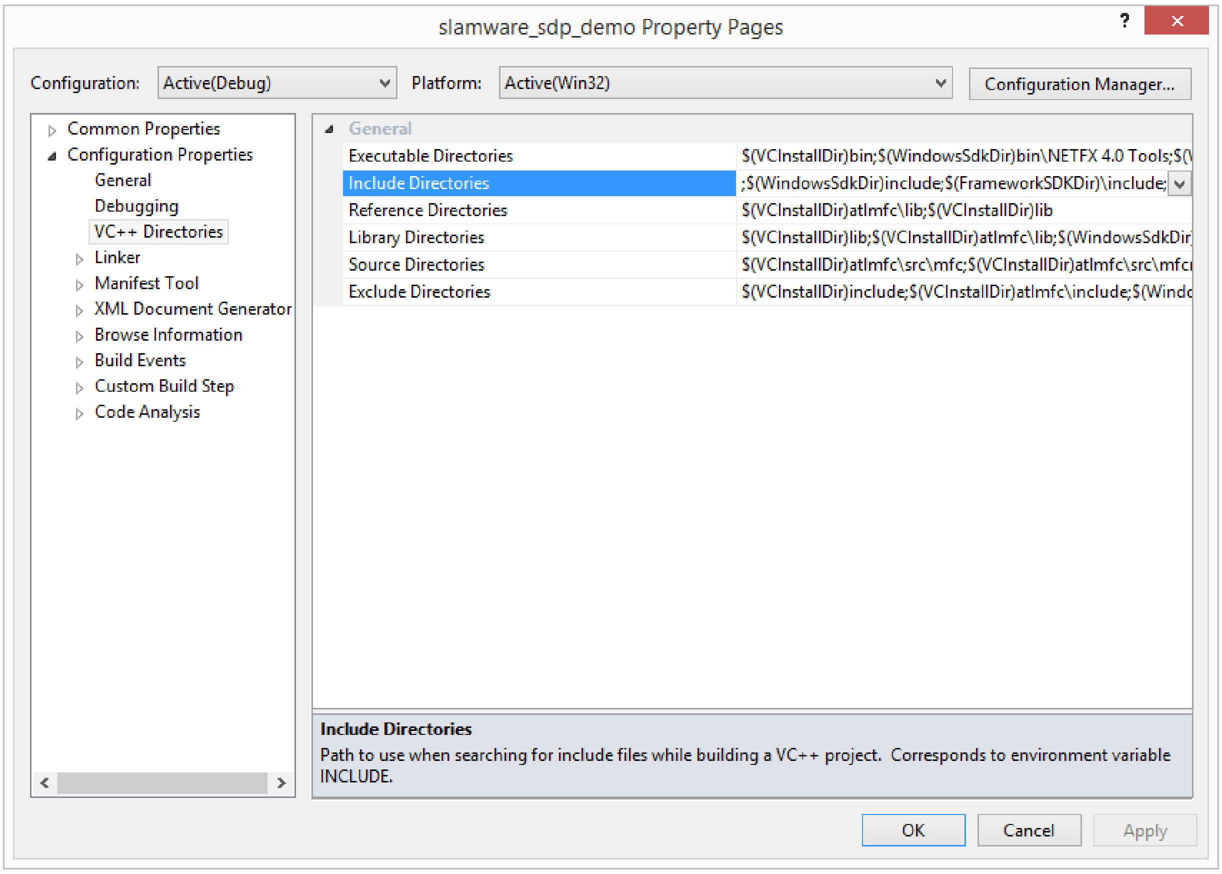
Output Directory

Specifies a relative path to the output file directory; can include environment variables.

Figure 3-4 Project Properties Page

Step2 Configuration VC++ directories

Choose VC++ Directories in the Configuration Properties.



Configuration: Active(Debug)

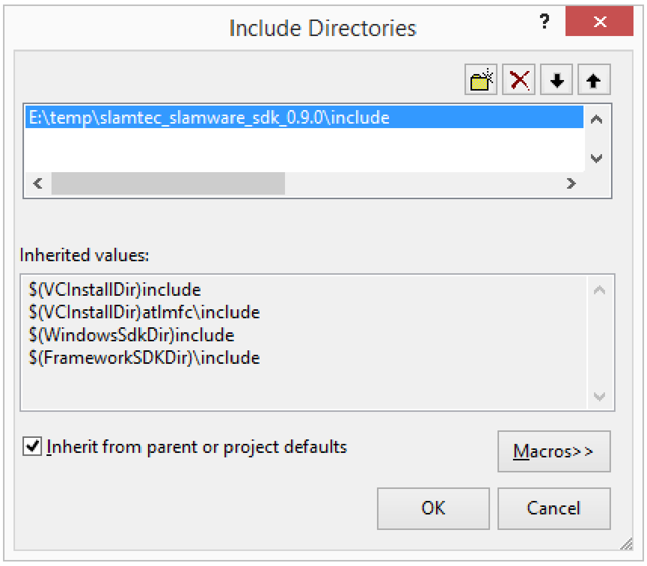
Platform: Active(Win32)

Executable Directories	$(VCInstallDir)bin;$(WindowsSdkDir)bin\NETFX 4.0 Tools;$(
Include Directories	;$(WindowsSdkDir)include;$(FrameworkSDKDir)\include;
Reference Directories	$(VCInstallDir)atlmfc\lib;$(VCInstallDir)lib
Library Directories	$(VCInstallDir)lib;$(VCInstallDir)atlmfc\lib;$(WindowsSdkDir
Source Directories	$(VCInstallDir)atlmfc\src\mfc;$(VCInstallDir)atlmfc\src\mfc
Exclude Directories	$(VCInstallDir)include;$(VCInstallDir)atlmfc\include;$(Windo

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

Figure 3-5 VC++ Directories

1. choose Include Directories and click the drop button.
2. choose <Edit···>.
3. enter the Include directory we have introduced in Figure 1-1 SDK Directory Structure into the Include Directories List.



E:\temp\slamtec_slamware_sdk_0.9.0\include

Inherited values:

$(VCInstallDir)include
$(VCInstallDir)atlmfc\include
$(WindowsSdkDir)include
$(FrameworkSDKDir)\include

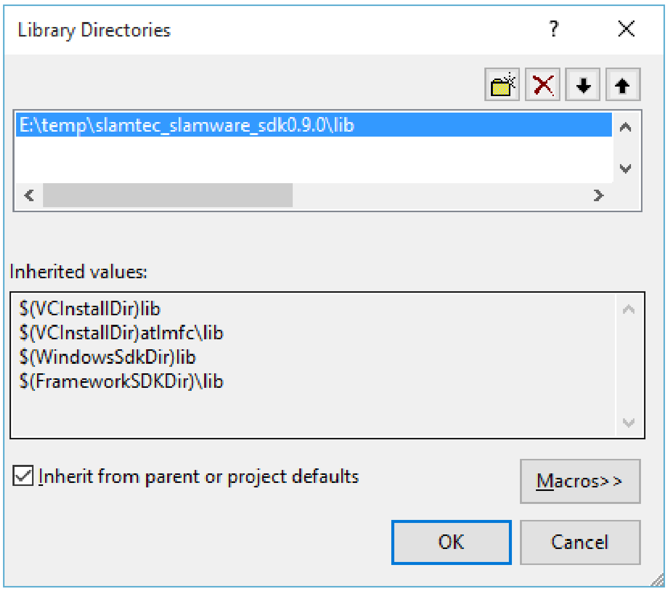
☑ Inherit from parent or project defaults

Figure 3-6 include Directories

1. Choose Library Directories and click the drop button.

2. Choose <Edit···>

3. Enter the Lib directory we have introduced in Figure 1-1 SDK Directory Structure into the Library Directories List.



Library Directories

`E:\temp\slamtec_slamware_sdk0.9.0\lib`

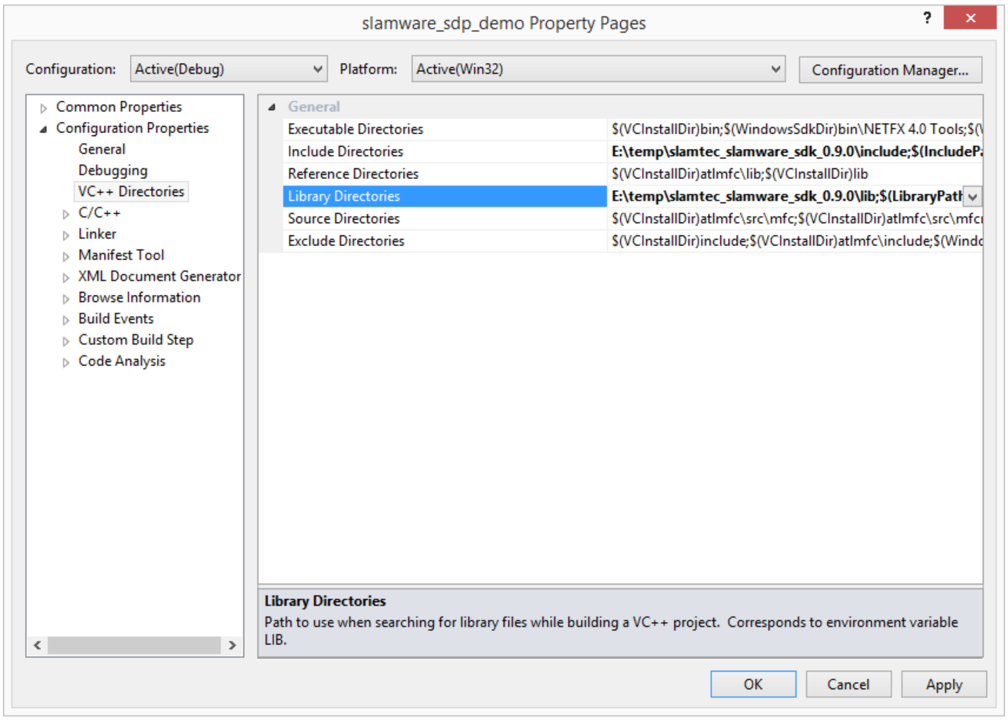
Inherited values:

```
$(VCInstallDir)lib
$(VCInstallDir)atlmfc\lib
$(WindowsSdkDir)lib
$(FrameworkSDKDir)\lib
```

☑ Inherit from parent or project defaults

Figure 3-7 Library Directories

After completion, the project Property Page should like page as follow:



slamware_sdp_demo Property Pages

Configuration: Active(Debug)   Platform: Active(Win32)

General	
Executable Directories	`$(VCInstallDir)bin;$(WindowsSdkDir)bin\NETFX 4.0 Tools;$(`
Include Directories	**`E:\temp\slamtec_slamware_sdk_0.9.0\include;$(IncludeP`**
Reference Directories	`$(VCInstallDir)atlmfc\lib;$(VCInstallDir)lib`
Library Directories	**`E:\temp\slamtec_slamware_sdk_0.9.0\lib;$(LibraryPath`**
Source Directories	`$(VCInstallDir)atlmfc\src\mfc;$(VCInstallDir)atlmfc\src\mfc`
Exclude Directories	`$(VCInstallDir)include;$(VCInstallDir)atlmfc\include;$(Windo`

Library Directories
Path to use when searching for library files while building a VC++ project. Corresponds to environment variable LIB.

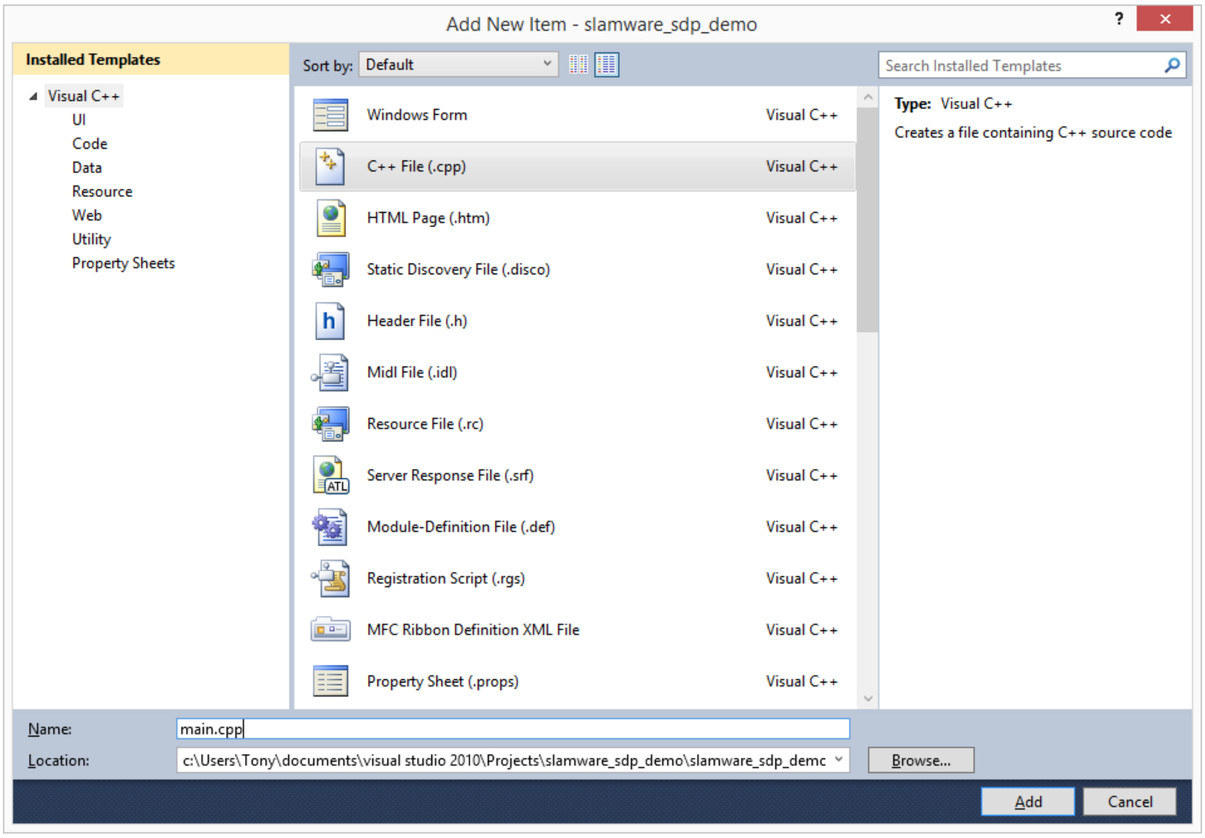
Figure 3-8 Completed Property Pages

Click OK button to finish the configuration.

Hello World

Step1 Add Source Files

Right click Source Files in the Solution Explorer of your project, Choose Add->New Item.


Add New Item - slamware_sdp_demo
?
✕
Installed Templates
Visual C++
UI
Code
Data
Resource
Web
Utility
Property Sheets
Sort by: Default
Search Installed Templates
Windows Form Visual C++
C++ File (.cpp) Visual C++
HTML Page (.htm) Visual C++
Static Discovery File (.disco) Visual C++
Header File (.h) Visual C++
Midl File (.idl) Visual C++
Resource File (.rc) Visual C++
Server Response File (.srf) Visual C++
Module-Definition File (.def) Visual C++
Registration Script (.rgs) Visual C++
MFC Ribbon Definition XML File Visual C++
Property Sheet (.props) Visual C++
Browse...
Add
Cancel

Type: Visual C++

Creates a file containing C++ source code

Name: main.cpp

Location: c:\Users\Tony\documents\visual studio 2010\Projects\slamware_sdp_demo\slamware_sdp_demc

Figure 3-9 Add New Item

Choose C++ File (.cpp), and enter the Name such as main.cpp.

Step2 Enter Code

Enter the following code in the pop editor:

```
#include <rpos/robot_platforms/slamware_core_platform.h>

#include <iostream>

using namespace std;

using namespace rpos::robot_platforms;

int main(int argc, char* argv[])

{

	SlamwareCorePlatform platform = SlamwareCorePlatform::connect("192.168.111.1", 1445);

	cout << "Base version: " << platform.getSDPVersion() << endl;
```

```
        return 0;

}
```

Step3 Build and run

Click Debug->Start Debugging in the menu bar of Visual Studio to build and run your application.

Overview

Class	Description
rpos::core::Location Class	Location
rpos::core::Rotation Class	Rotation Attitude
rpos::core::Pose Class	Attitude
rpos::core::Action Class	Action
rpos::core::ActionStatus Enum	Status of Action
rpos::core::Feature Class	Feature Class
rpos::core::RectangleF Class	Rectangle using float
rpos::core::Vector2f Class	Bivector using float
rpos::core::Vector2i Class	Bivector using int
rpos::core::LaserPoint Class	LiDAR scanning spot
rpos::core::RobotPlatform Class	Robot Platform base class
rpos::robot::heading::HeadingMode Enum	The heading mode when the robot is moveing.
rpos::robot::heading::RobotHeading Class	Setting the heading of the robot.
rpos::robot::option::MoveOption Structure	Store the settings about the robot moving.
rpos::actions::MoveAction Class	Move Action
rpos::features::ArtifactProvider Class	Feature of Artifact
rpos::features::LocationProvider Class	Feature of Location
rpos::features::MotionPlanner Class	Feature of Motion Planning
rpos::features::SweepMotionPlanner Class	Feature of Sweep Planning

rpos::features::SystemResource Class	Feature of System Resource
rpos::features::location_provider::Map Class	Map
rpos::features::location_provider::MapType Enum	Map Type
rpos::features::location_provider::BitmapMap Class	Bitmap Map
rpos::features::location_provider::BitmapMapPixelFormat Enum	Pixel Format
rpos::features::motion_planner::Path Class	Path
rpos::features::system_resource::LaserScan Class	LiDAR Scanning Result
rpos::robot_platforms::SlamwareCorePlatform Class	Slamware CORE Robot Platform

Figure 4-1 API Overview

rpos::core::Location Class

Overview

Location Class is used to represent coordinates of 3D space. It consists of three members, x, y and z, according to the right-handed coordinate system.

Header

rpos/core/pose.h

Constructor

> `Location()`
> `Location(double x, double y, double z)`
> `Location(const Location&)`

Operator

> `Location& operator=(const Location&)`

Function

> `double x() const`
> `double& x()`
> `double y() const`
> `double& y()`
> `double z() const`

```
double& z()
```

`Location()` Constructor

Create a Location object, and set property, x, y and z, to zero.

Location(double x, double y, double z) Constructor

Create a Location object with given property, x, y and z.

Location(const Location&) Constructor

Copy constructor.

Location& operator=(const Location&) Operator

Assignment operator.

double x() const、double& x() Property

Property: x.

Example

```
Location location;
std::cout<<location.x()<<std::endl; // output 0
location.x() = 10;
std::cout<<location.x()<<std::endl; // output 10
```

double y() const、double& y() Property

Property: y.

See also Property: x.

double z() const、double& z() Property

Property: z.

See also Property: x.

rpos::core::Rotation Class

Overview

Rotation Class is used to represent rotation attitude. Its angular unit is radian.

Header

rpos/core/pose.h

Constructor

> `Rotation()`
> `Rotation(double yaw, double pitch, double roll)`
> `Rotation(const Rotation&)`

Operator

> `Rotation& operator=(const Rotation&)`

Function

> `double yaw() const`
> `double& yaw()`
> `double pitch() const`
> `double& pitch()`
> `double roll() const`
> `double& roll()`

Rotation() Constructor

Create a Rotation object, and set property, yaw, pitch and roll to zero.

Rotation(double yaw, double pitch, double roll) Constructor

Create a Rotation object with given property, yaw，pitch，roll.

Rotation(const Rotation&) Constructor

Copy constructor.

Rotation& operator=(const Rotation&) Operator

Assignment operator.

double yaw() const、double& yaw() Property

Property: yaw, according to Tait-Bryan angles, See Resources:

http://en.wikipedia.org/wiki/Euler_angles#Tait.E2.80.93Bryan_angles

See also Example: Location::x().

double pitch() const、double& pitch() Property

Property: pitch.

double roll() const, double& roll() Property

Property: roll.

rpos::core::Pose Class

Overview

Pose contains data of Location and Rotation to describe a complete attitude in 3D space.

Header

rpos/core/pose.h

Constructor

`Pose()`
`Pose(const Location&, const Rotation&)`
`Pose(const Location&)`
`Pose(const Rotation&)`
`Pose(const Pose&)`

Operator

`Pose& operator=(const Pose&)`

Function

`const Location& location() const`
`Location& location()`
`const Rotation& rotation() const`
`Rotation& rotation()`
`double x() const`
`double& x()`
`double y() const`
`double& y()`
`double z() const`
`double& z()`
`double yaw() const`
`double& yaw()`

```
double pitch() const
double& pitch()
double roll() const
double& roll()
```

Pose() Constructor

Create a Pose object, and set property, x, y, z, yaw, pitch and roll, to zero.

Pose(const Location&, const Rotation&) Constructor

Create a Pose object with given Location and Rotation object.

Pose(const Location&) Constructor

Create Pose object with given Location object, and set other property to zero.

Pose(const Rotation&) Constructor

Create a Pose object with given Rotation object, and set other property to zero.

Pose(const Pose&) Constructor

Copy constructor.

Pose& operator=(const Pose&) Operator

Assignment operator.

const Location& location() const、Location& location() Property

Location, see also Location Class.

const Rotation& rotation() const、Rotation& rotation() Property

Rotation, see also Rotation Class.

double x() const、double& x() Property

Property: x.

Example

```
Location location;
std::cout<<location.x()<<std::endl; // output 0
location.x() = 10;
std::cout<<location.x()<<std::endl; // output 10
```

double y() const、double& y() Property

y Property

See also Property x.

double z() const、double& z() Property

z Property

See also Property x.

double yaw() const、double& yaw() Property

Property: yaw, according to Tait-Bryan angles, See Resources:

http://en.wikipedia.org/wiki/Euler_angles#Tait.E2.80.93Bryan_angles

See also Example: Location::x().

double pitch() const、double& pitch() Property

Property: pitch.

double roll() const, double& roll() Property

Property: roll.

rpos::core::Action Class

Overview

Action Class describe an action.

Header

rpos/core/action.h

Constructor

`Action(const Action&)`

Operator

`Action& operator=(const Action&)`

Function

`ActionStatus getStatus()`

`void cancel()`

`ActionStatus waitUntilDone()`

```
template<class ActionT> ActionT cast()
```

Action(const Action&) Constructor

Copy constructor.

Action& operator=(const Action&) Operator

Assignment operator.

ActionStatus getStatus()

Get current status of action. Return code refers to: ActionStatus Enum

void cancel()

Cancel the current action.

ActionStatus waitUntilDone()

Wait until action is done and return the last action or error. See also ActionStatus Enum

template<class ActionT> ActionT cast()

Up casting.

Example

```
rpos::core::Action someAction = robotPlatform.startSomeAction();
rpos::actions::MoveAction                    moveAction                =
someAction.cast<rpos::actions::MoveAction>;
```

rpos::core::ActionStatus Enum

Overview

ActionStatus Enum describe the action status.

Header

rpos/core/action.h

Enumeration

```
ActionStatusWaitingForStart
ActionStatusRunning
ActionStatusFinished
```

> `ActionStatusPaused`
> `ActionStatusStopped`
> `ActionStatusError`

ActionStatusWaitingForStart

Action is waiting for start.

ActionStatusRunning

Action is running.

ActionStatusFinished

Action has Finished action.

ActionStatusPaused

Action has been paused.

ActionStatusStopped

Action has been canceled.

ActionStatusError

Error occurred.

rpos::core::Feature Class

Overview

Feature Class contains some specific functions.

Header

rpos/core/feature.h

Constructor

> `Feature(const Feature&)`

Operator

> `Feature& operator=(const Feature&)`

Feature(const Feature&) Constructor

Copy constructor.

Feature& operator=(const Feature&) Operator

Assignment operator.

rpos::core::RectangleF Class

Overview

RectangleF Class describe a rectangle; its type of coordinate parameter is float.

Header

rpos/core/geometory.h

Constructor

> `RectangleF()`
> `RectangleF(Vector2f position, Vector2f size)`
> `RectangleF(float x, float y, float width, float height)`
> `RectangleF(const RectangleF&)`

Operator

> `RectangleF& operator=(const RectangleF&)`

Function

> `const Vector2f& position()`
> `Vector2f& position()`
> `const Vector2f& size()`
> `Vector2f& size()`
> `float x() const`
> `float& x()`
> `float y() const`
> `float& y()`
> `float width() const`
> `float& width()`
> `float height() const`
> `float& height()`
> `float left() const`
> `float right() const`
> `float top() const`
> `float bottom() const`
> `bool contains(const Vector2i& point)`

> bool empty()
> bool contains(const RectangleF& dest)
> void unionOf(const RectangleF& dest)

RectangleF() Constructor

Create a RectangleF object, and set property to zero.

RectangleF(Vector2f position, Vector2f size) Constructor

Create a RectangleF object with given position and size.

RectangleF(float x, float y, float width, float height) Constructor

Create a RectangleF object with given position and size.

RectangleF(const RectangleF&) Constructor

Copy constructor.

RectangleF& operator=(const RectangleF&) Operator

Assignment operator.

const Vector2f& position()、Vector2f& position() Property

Property: Position. The left top position of rectangle.

const Vector2f& size()、Vector2f& size() Property

Property: Size.

float x() const、float& x() Property

Property: x. The x-coordinate of position.

float y() const、float& y() Property

Property: y. The y-coordinate of position.

float height() const、float& height() Property

Height.

float width() const、float& width() Property

Width.

float left() const Property

The x-coordinate of rectangle left side.

float right() const Property

The x-coordinate of rectangle right side.

float top() const Property

The y-coordinate of rectangle top side.

float bottom() const Property

The y-coordinate of rectangle bottom side.

bool contains(const Vector2i& point)

To determine whether the point within the scope of the rectangle.

bool empty()

To determine whether the rectangle is empty, which means width and height of rectangle is small enough to ignore.

bool contains(const RectangleF& dest)

To determine whether the part of the source and destination rectangles overlap exists.

void unionOf(const RectangleF& dest)

Calculate the rectangular part of the source and destination rectangles overlap, and reset the source rectangle to it.

rpos::core::Vector2f Class

Overview

Bivector, data type is float.

Header

rpos/core/geometry.h

Constructor

`Vector2f()`

> `Vector2f(float x, float y)`
> `Vector2f(const Vector2f&)`

Operator

> `Vector2f& operator=(const Vector2f&)`

Function

> `float x() const`
> `float& x()`
> `float y() const`
> `float& y()`

Vector2f() Constructor

Create a bivector with indeterminate property.

Vector2f(float x, float y) Constructor

Create a bivector with given property.

Vector2f(const Vector2f&) Constructor

Copy constructor.

Vector2f& operator=(const Vector2f&) Operator

Assignment operator.

float x() const、float& x() Property

X-component of bivector.

float y() const、float& y() Property

Y-component of bivector.

rpos::core::Vector2i Class

Overview

Bivector, data type is int.

Header

rpos/core/geometry.h

Constructor

Vector2i()

Vector2i(float x, float y)

Vector2i(const Vector2i&)

Operator

Vector2i& operator=(const Vector2i&)

Function

int x() const

int& x()

int y() const

int& y()

See also rpos::core::Vector2f Class

rpos::core::LaserPoint Class

Overview

Single-point data of LiDAR distance measure, contains distance, angle and validity information.

Header

rpos/core/laser_point.h

Constructor

`LaserPoint()`
`LaserPoint(float distance, float angle, bool valid)`
`LaserPoint(const LaserPoint&)`

Operator

`LaserPoint& operator=(const LaserPoint&)`

Function

`float distance() const`
`float& distance()`
`float angle() const`

`float& angle()`
`bool valid() const`
`bool& valid()`

LaserPoint() Constructor

Create a LaserPoint object.

LaserPoint(float distance, float angle, bool valid) Constructor

Create a LaserPoint object with given distance, angle and validity information.

LaserPoint(const LaserPoint&) Constructor

Copy constructor.

LaserPoint& operator=(const LaserPoint&) Operator

Assignment operator.

float distance() const、float& distance() Property

Property: distance in metres.

float angle() const、float& angle() Property

Property: angle in radian.

bool valid() const、bool& valid() Property

Property: validity.

rpos::core::RobotPlatform Class

Overview

Robot platform is a combination of a series of equipment, provides a series of features to support difference function. RobotPlatform Class is the base class of all the robot platform class.

Header

rpos/core/robot_platform.h

Constructor

`RobotPlatform(const RobotPlatform&)`

Operator

> `RobotPlatform& operator=(const RobotPlatform&)`

Function

> `std::vector<Feature> getFeatures()`
> `template<class RobotPlatformT> RobotPlatformT cast()`

RobotPlatform(const RobotPlatform&) Constructor

Copy constructor.

RobotPlatform& operator=(const RobotPlatform&) Operator

Assignment operator.

std::vector<Feature> getFeatures()

Get all the features the robot platform provides.

template<class RobotPlatformT> RobotPlatformT cast()

Up casting, see also rpos::core::Action::cast<>

rpos::robot::heading::HeadingMode Enum

Overview

HeadingMode Enum represent the heading mode when the robot is moving.

Header

`rpos/features/motion_planner/move_heading.h`

Enumeration

> `HeadingModeAuto,`
> `HeadingModeFixAngle,`
> `HeadingModeCircleMotion,`
> `HeadingModeDirection`

HeadingModeAuto

Make the robot move at random.

HeadingModeFixAngle

Make the robot move with a fixed angle between the heading direction and the moving direction.

HeadingModeCircleMotion

Make the robot move with a fixed heading direction towards an object or a point.

HeadingModeDirection

Make the robot move without heading change.

rpos::robot::heading::RobotHeading Class

Overview

RobotHeading class represents the setting about the heading direction towards an object or a fixed direction when the robot is moving.

Header

`rpos/features/motion_planner/move_heading.h`

Structure

> `RobotHeading()`
> `RobotHeading(HeadingMode headingMode, rpos::core::Pose pose)`

Operator

> `RobotHeading& operator=(const RobotHeading&)`

Function

> `const rpos::core::Pose& pose() const`
> `const HeadingMode& headingMode() const`

RobotHeading()Construct

Copy constructor.

RobotHeading(HeadingMode headingMode, rpos::core::Pose pose) Construct

Constructor.

RobotHeading& operator=(const RobotHeading&) Operator

Assignment operator.

const rpos::core::Pose& RobotHeading::pose() const

Get the heading angle of the moving robot or its location.

The relationship between the heading direction and the angle.

Heading Direction	Value
HeadingModeAuto	Pose value is unavailable.
HeadingModeFixAngle or HeadingModeDirection	Pose's Rotation parameter is available.
HeadingModeCircleMotion	Pose's Location parameter is available.

const HeadingMode& RobotHeading::headingMode() const

Get the heading settings of moving robot.

Please refer to `rpos::robot::heading::HeadingMode.`

rpos::robot::option::MoveOption Structure

Overview

MoveOption Structure stores the setting of the robot moving.

Header

`rpos/features/motion_planner/move_option.h`

Structure Description

- `appending`
- `isMilestone`
- `robotHeading`

appending

It indicates if Slamware should clear current tasks or append these point to the visit list when the robot is executing other operations.

isMilestone

It indicates if Slamware should plan a route to the points or go directly to the point.

robotHeading

Setting the heading direction for moving robot.

Please refer to `rpos::robot::heading::RobotHeading`.

rpos::actions::MoveAction Class

Overview

MoveAction Class describe a move action, contains the planned path, check point list and moving process.

Header

rpos/features/motion_planner/move_action.h

Parent Class

`rpos::core::Action` Class

Constructor

> `MoveAction(boost::shared_ptr<rpos::actions::detail::MoveActionImpl>)`
> `MoveAction(const MoveAction&)`

Operator

> `MoveAction& operator=(const MoveAction&)`

Function

> `rpos::features::motion_planner::Path getRemainingPath()`
> `rpos::features::motion_planner::Path getRemainingMilestones()`

Function inherits from rpos::core::Action Class

> `ActionStatus getStatus()`
> `void cancel()`
> `ActionStatus waitUntilDone()`
> `template<class ActionT> ActionT cast()`

MoveAction(boost::shared_ptr<rpos::actions::detail::MoveActionImpl>) Constructor

SDK internal use only.

MoveAction(const MoveAction&) Constructor

Copy constructor.

MoveAction& operator=(const MoveAction&) Operator

Assignment operator.

rpos::features::motion_planner::Path getRemainingPath()

Get remaining planned path.

rpos::features::motion_planner::Path getRemainingMilestones()

Get remaining milestones.

rpos::features::ArtifactProvider Class

Overview

Feature of artifact, contains virtual wall function.

Header

rpos/features/artifact_provider.h

Parent Class

`rpos::core::Feature Class`

Constructor

> `ArtifactProvider(boost::shared_ptr<detail::ArtifactProviderImpl>)`
> `ArtifactProvider(const ArtifactProvider&)`

Operator

> `ArtifactProvider& operator=(const ArtifactProvider&)`

Function

> `std::vector<rpos::core::Line> getWalls()`
> `bool addWall(const rpos::core::Line&)`
> `bool addWalls(const std::vector<rpos::core::Line>&)`
> `bool clearWallById(const rpos::core::SegmentID&)`
> `bool clearWalls()`

`ArtifactProvider(boost::shared_ptr<detail::ArtifactProviderImpl>)` Constructor

SDK internal use only.

`ArtifactProvider(const ArtifactProvider&)` Constructor

Copy constructor.

`ArtifactProvider& operator=(const ArtifactProvider&)` Operator

Assignment operator.

`std::vector<rpos::core::Line> getWalls()`

Get all virtual walls information.

`bool addWall(const rpos::core::Line&)`

Add a virtual wall information.

`bool addWalls(const std::vector<rpos::core::Line>&)`

Add several virtual walls information.

`bool clearWallById(const rpos::core::SegmentID&)`

Remove the specified virtual wall information by ID.

`bool clearWalls()`

Remove all virtual walls information.

rpos::features::LocationProvider Class

Overview

localization features, contains auto mapping and localization, i.e., SLAM function.

Header

rpos/features/location_provider.h

Parent Class

`rpos::core::Feature Class`

Constructor

> `LocationProvider(boost::shared_ptr<detail::LocationProviderImpl>)`
> `LocationProvider(const LocationProvider&)`

Operator

```
LocationProvider& operator=(const LocationProvider&)
```

Function

```
std::vector<rpos::features::location_provider::MapType>
getAvailableMaps()
rpos::features::location_provider::Map getMap(
    rpos::features::location_provider::MapType,
    rpos::core::RectangleF,
    rpos::features::location_provider::MapKind)
bool setMap(
    const rpos::features::location_provider::Map&,
    rpos::features::location_provider::MapType,
    rpos::features::location_provider::MapKind)
rpos::core::RectangleF getKnownArea(
    rpos::features::location_provider::MapType,
    rpos::features::location_provider::MapKind)
bool clearMap()
rpos::core::Location getLocation()
rpos::core::Pose getPose()
bool setPose(const rpos::core::Pose&)
bool getMapLocalization()
bool setMapLocalization(bool)
bool getMapUpdate()
bool setMapUpdate(bool)
```

`LocationProvider(boost::shared_ptr<detail::LocationProviderImpl>)` Constructor

SDK internal use only.

`LocationProvider(const LocationProvider&)` Constructor

Copy constructor.

`LocationProvider& operator=(const LocationProvider&)` Operator

Assignment operator.

```
std::vector< rpos::features::location_provider::MapType>
getAvailableMaps()
```

Get all the map types the localization features provide.

```
rpos::features::location_provider::Map getMap(
rpos::features::location_provider::MapType,
rpos::core::RectangleF,
rpos::features::location_provider::MapKind)
```

Get the map information by the given map type and the scope of rectangle of the localization features.

```
bool setMap(
const rpos::features::location_provider::Map&,
rpos::features::location_provider::MapType,
rpos::features::location_provider::MapKind)
```

Upload the map information with the given map type to the localization features, and return whether or not the function succeeded.

```
rpos::core::RectangleF getKnownArea(
rpos::features::location_provider::MapType,
rpos::features::location_provider::MapKind)
```

Get the scope of map which is mapped by the given map type.

```
bool clearMap()
```

Remove data of map.

```
rpos::core::Location getLocation()
```

Get the location of robot in the coordinate system.

```
rpos::core::Pose getPose()
```

Get the pose of robot in the coordinate system.

```
bool setPose(const rpos::core::Pose&)
```

Upload the pose of robot in the coordinate system, and return whether or not the function succeeded.

`bool getMapLocalization()`

Get whether or not the robot opens the localization function.

`bool setMapLocalization(bool)`

Set the robot to open the localization function or not.

`bool getMapUpdate()`

Get whether or not the robot opens the mapping function.

`bool setMapUpdate(bool)`

Set the robot to open the mapping function or not.

rpos::features::MotionPlanner Class

Overview

Features of route planning, contains dynamic path planning and automatic obstacle avoidance function.

Header

rpos/features/motion_planner.h

Parent Class

> `rpos::core::Feature Class`

Constructor

> `MotionPlanner(boost::shared_ptr<detail::MotionPlannerImpl>)`
> `MotionPlanner(const MotionPlanner&)`

Operator

> `MotionPlanner& operator=(const MotionPlanner&)`

Function

```
rpos::actions::MoveAction moveTo(
     const std::vector<rpos::core::Location>&,
     bool, bool)
rpos::actions::MoveAction moveTo(const rpos::core::Location&,
bool,bool)
rpos::actions::MoveAction getCurrentAction()
rpos::features::motion_planner::Path searchPath(const
```

rpos::core::Location&)

MotionPlanner(boost::shared_ptr<detail::MotionPlannerImpl>) Constructor

SDK internal use only.

MotionPlanner(const MotionPlanner&) Constructor

Copy constructor.

MotionPlanner& operator=(const MotionPlanner&) Operator

Assignment operator.

rpos::actions::MoveAction moveTo(const std::vector<rpos::core::Location>&, bool, bool)

Make robot move along the specified path. The robot will try to move to the point in route one by one harmoniously and automatically avoid obstacles.

Parameter

Name	Type	Description
locations	const std::vector <rpos::core::Location>&	The points we hope robot passes through.
appending	bool	If robot is doing other move action, this parameter determines whether the new points will be added to or replace existing location list.
isMilestone	bool	If the value of parameter is true, the points will be seen as milestones, and robot will open the route planning function to move to it. If the parameter is false, the points will be seen as normal points and the route planning function will not be opened.

rpos::actions::MoveAction moveTo(const rpos::core::Location&, bool,bool)

Make robot move to the target location.

Parameter

Name	Type	Description
location	const rpos::core::Location&	The location we hope robot move to.
appending	bool	If robot is doing other move action, this parameter determines whether the new points will be added to or replace existing location list.
isMilestone	bool	If the value of parameter is true, the location will be seen as a milestone, and robot will open the route planning function to move to it. If the parameter is false, the location will be seen as a normal point, and robot will not open the route planning function.

`rpos::actions::MoveAction getCurrentAction()`

Get the current action which robot is doing.

You can use Action::isEmpty() function to judge if the action exist. If robot is doing nothing, the Action::isEmpty() function will return true.

`rpos::features::motion_planner::Path searchPath(const rpos::core::Location&)`

Find the path to the specified location with the built-in algorithm.

rpos::features::SweepMotionPlanner Class

Overview

Features of sweeping route planning. This class is specific to the sweeping and refilling function which sweeping robot specified version, Slamware Core, provides.

Header

rpos/features/sweep_motion_planner.h

Parent Class

`rpos::core::Feature Class`

Constructor

> SweepMotionPlanner(boost::shared_ptr<detail::SweepMotionPlannerImpl>)
> SweepMotionPlanner(const SweepMotionPlanner&)

Operator

> SweepMotionPlanner& operator=(const SweepMotionPlanner&)

Function

> rpos::actions::SweepMoveAction startSweep()
> rpos::actions::SweepMoveAction sweepSpot(const rpos::core::Location& location)
> rpos::actions::MoveAction goHome()

`SweepMotionPlanner(boost::shared_ptr<detail::SweepMotionPlannerImpl>)` Constructor

SDK internal use only.

`SweepMotionPlanner(const SweepMotionPlanner&)` Constructor

Copy constructor.

`SweepMotionPlanner& operator=(const SweepMotionPlanner&)` Operator

Assignment operator.

`rpos::actions::SweepMoveAction startSweep()`

Let robot sweep. (applied to sweep robot version only)

`rpos::actions::SweepMoveAction sweepSpot(const rpos::core::Location& location)`

Let robot do spot cleaning. (applied to sweep robot version only)

`rpos::actions::MoveAction goHome()`

Let robot return to charge.

rpos::features::system_resource::DeviceInfo Class

Overview

Get the device information including device ID, manufacturer, model, hardware version, software version.

Header

`rpos/features/device_info.h`

Constructor

> `DeviceInfo()`
> `DeviceInfo(const DeviceInfo&)`
> `Operator`
> `DeviceInfo& operator=(const DeviceInfo&)`

Method

> `std::string deviceID() const、std::string& deviceID();`
> `int manufacturerID() const、int& manufacturerID();`
> `std::string manufacturerName() const 、 std::string& manufacturerName();`
> `int modelID() const、int& modelID();`
> `std::string modelName() const、std::string& modelName();`
> `std::string hardwareVersion() const 、 std::string& hardwareVersion();`
> `std::string softwareVersion() const 、 std::string& softwareVersion();`

`DeviceInfo()`

Constructor.

`DeviceInfo(const DeviceInfo&)`

Create a function with specific device information as its parameter.

`DeviceInfo& operator=(const DeviceInfo&)`

Assignment operator.

`std::string deviceID() const、std::string& deviceID();`

deviceID property.

`int manufacturerID() const、int& manufacturerID();`

manufacturerID property.

`std::string manufacturerName() const、std::string& manufacturerName();`

manufacturerName property.

`int modelID() const、int& modelID();`

modelID property.

`std::string modelName() const、std::string& modelName();`

modelName property.

`std::string hardwareVersion() const、std::string& hardwareVersion();`

Hardware version property.

`std::string softwareVersion() const、std::string& softwareVersion();`

Software version property.

rpos::features::SystemResource Class

Overview

Features of system resource. The class provides the API to get the original data of laser scanning and access to resource of power management system.

Header

rpos/features/system_resource.h

Parent Class

`rpos::core::Feature Class`

Constructor

`SystemResource(boost::shared_ptr<detail::SystemResourceImpl>)`
`SystemResource(const SystemResource&)`

Operator

> [`SystemResource& operator=(const SystemResource&)`]

Function

> [`int getBatteryPercentage()`]
> [`bool getBatteryIsCharging()`]
> [`bool getDCIsConnected()`]
> [`int getBoardTemperature()`]
> [`std::string getSDPVersion()`]
> [`rpos::features::system_resource::LaserScan getLaserScan()`]

`SystemResource(boost::shared_ptr<detail::SystemResourceImpl>)` Constructor

SDK internal use only.

`SystemResource(const SystemResource&)` Constructor

Copy constructor.

`SystemResource& operator=(const SystemResource&)` Operator

Assignment operator.

`int getBatteryPercentage()`

Get remaining capacity (Unit: percent). e.g., If remaining capacity is 56%, then 56 will be returned.

`bool getBatteryIsCharging()`

Check if robot is under the status of battery charge.

`bool getDCIsConnected()`

Check if external power is connected.

`int getBoardTemperature()`

Get the body temperature.

`std::string getSDPVersion()`

Get version number of board.

`rpos::features::system_resource::LaserScan getLaserScan()`

Get original data of laser scanning.

rpos::features::location_provider::Map Class

Overview

Base class of map, means in general the map localization function gets.

Header

rpos/features/location_provider.h

Constructor

`Map(boost::shared_ptr<detail::MapImpl>)`
`Map(const Map&)`

Operator

`Map& operator=(const Map&)`

Function

`rpos::core::RectangleF getMapArea()`
`rpos::core::Vector2f getMapPosition()`
`rpos::core::Vector2i getMapDimension()`
`rpos::core::Vector2f getMapResolution()`
`rpos::system::types::timestamp_t getMapTimestamp()`
`void setMapData(float, float, int, int, float, const std::vector< u8>&, rpos::system::types:: u64)`
`std::vector< u8>& getMapData()`
`template<class MapT> MapT cast()`

`Map(boost::shared_ptr<detail::MapImpl>)` Constructor

SDK internal use only.

`Map(const Map&)` Constructor

Copy constructor.

`Map& operator=(const Map&)` Operator

Assignment operator.

`rpos::core::RectangleF getMapArea()`

Get the scope of map.

`rpos::core::Vector2f getMapPosition()`

Get the x-coordinate of the scope of rectangle of map.

`rpos::core::Vector2i getMapDimension()`

Get the size of map (Unit: pixel).

`rpos::core::Vector2f getMapResolution()`

Get the map resolution (one metre per pixel).

`rpos::system::types::timestamp_t getMapTimestamp()`

Get the timestamp when the map is created.

`void setMapData(float, float, int, int, float, const std::vector<_u8>&, rpos::system::types::_u64)`

Set the Map data.

`std::vector<_u8>& getMapData()`

Get data of map.

`template<class MapT> MapT cast()`

Up casting.

rpos::features::location_provider::MapType Enum

Overview

MapType Enum means map type.

Header

rpos/features/location_provider.h

Enumeration

`MapTypeBitmap8Bit`

MapTypeBitmap8Bit

8 bit per pixel.

rpos::features::location_provider::BitmapMap Class

Overview

Bitmap map.

Header

rpos/features/location_provider.h

Parent Class

`rpos::features::location_provider::Map Class`

Constructor

> `BitmapMap(boost::shared_ptr<detail::BitmapMapImpl>)`
> `BitmapMap(const BitmapMap&)`

Operator

> `BitmapMap& operator=(const BitmapMap&)`

Function

> `BitmapMapPixelFormat getMapFormat()`

Function inherits from rpos::features::location_provider::Map Class

> `rpos::core::RectangleF getMapArea()`
> `rpos::core::Vector2f getMapPosition()`
> `rpos::core::Vector2i getMapDimension()`
> `rpos::core::Vector2f getMapResolution()`
> `rpos::system::types::timestamp_t getMapTimestamp()`
> `void setMapData(float, float, int, int, float, const std::vector<_u8>&, rpos::system::types::_u64)`
> `std::vector<_u8>& getMapData()`
> `template<class MapT> MapT cast()`

BitmapMap(boost::shared_ptr<detail::BitmapMapImpl>) Constructor

SDK internal use only.

`BitmapMap(const BitmapMap&)` Constructor

Copy constructor.

`BitmapMap& operator=(const BitmapMap&)` Operator

Assignment operator.

`BitmapMapPixelFormat getMapFormat()`

Get the pixel format of bitmap map.

rpos::features::location_provider::BitmapMapPixelFormat Enum

Overview

BitmapMapPixelFormat Enum means the pixel format of Bitmap map.

Header

rpos/features/location_provider.h

Enumeration

> `BitmapMapPixelFormat8Bit`

`BitmapMapPixelFormat8Bit`

One byte per pixel.

rpos::features::motion_planner::Path Class

Overview

Path object consists f several Location objects, which means a route.

Header

rpos/features/motion_planner.h

Constructor

> `Path(const std::vector<rpos::core::Location>&)`
> `Path(const Path&)`

Operator

> `Path& operator=(const Path&)`

Function

> `std::vector<rpos::core::Location>& getPoints()`

`Path(const std::vector<rpos::core::Location>&)` Constructor

Create a route that consists of several locations.

`Path(const Path&)` Constructor

Copy constructor.

`Path& operator=(const Path&)` Operator

Assignment operator.

`std::vector<rpos::core::Location>& getPoints()`

Get list of locations in the route.

rpos::features::system_resource::LaserScan Class

Overview

LaserScan object consists of several LaserPoint objects, which means data of once laser scanning.

Header

rpos/features/system_resource.h

Constructor

> `LaserScan(const std::vector<rpos::core::LaserPoint>&)`
> `LaserScan(const LaserScan&)`

Operator

> `LaserScan& operator=(const LaserScan&)`

Function

> `std::vector<rpos::core::LaserPoint>& getLaserPoints()`

`LaserScan(const std::vector<rpos::core::LaserPoint>&)` Constructor

Create a data of laser scanning that consists of several laser points.

`LaserScan(const LaserScan&)` Constructor

Copy constructor.

`LaserScan& operator=(const LaserScan&)` Operator

Assignment operator.

`std::vector<rpos::core::LaserPoint>& getLaserPoints()`

Get data of all laser points.

rpos::robot_platforms::SlamwareCorePlatform Class

Overview

SlamwareCorePlatform object means a robot that bases on Slamware. You can get equipment status and control equipment with it.

Header

rpos/robot_platforms/slamware_core_platform.h

Parent Class

`rpos::core::RobotPlatform Class`

Constructor

> `SlamwareCorePlatform(boost::shared_ptr<detail::SlamwareCorePlatformImpl>)`
> `SlamwareCorePlatform(const SlamwareCorePlatform&)`

Operator

> `SlamwareCorePlatform& operator=(const SlamwareCorePlatform&)`

Static Function

> `SlamwareCorePlatform connect(const std::string&, int, int)`

Function

> `void disconnect()`
> `std::vector<rpos::core::Line> getWalls()`
> `bool addWall(const rpos::core::Line&)`
> `bool addWalls(const std::vector<rpos::core::Line>&)`
> `bool clearWallById(const rpos::core::SegmentID&)`
> `bool clearWalls()`

```
std::vector<rpos::features::location_provider::MapType>
getAvailableMaps()
rpos::features::location_provider::Map getMap(
     rpos::features::location_provider::MapType,
     rpos::core::RectangleF,
     rpos::features::location_provider::MapKind)
bool setMap(
     const rpos::features::location_provider::Map&,
     rpos::features::location_provider::MapType,
     rpos::features::location_provider::MapKind, bool partially
= false)
bool setMap( const core::Pose&,
     const rpos::features::location_provider::Map&,
     rpos::features::location_provider::MapType,
     rpos::features::location_provider::MapKind, bool partially
= false)
rpos::core::RectangleF getKnownArea(
     rpos::features::location_provider::MapType,
     rpos::features::location_provider::MapKind)
bool clearMap()
bool clearMap(rpos::features::location_provider::MapKind kind)
rpos::core::Location getLocation()
rpos::core::Pose getPose()
bool setPose(const rpos::core::Pose&)
bool getMapLocalization()
bool setMapLocalization(bool)
bool getMapUpdate()
bool setMapUpdate(bool)
int getLocalizationQuality()
rpos::actions::MoveAction                          moveTo(const
std::vector<rpos::core::Location>&, bool, bool)
rpos::actions::MoveAction  moveTo(const  rpos::core::Location&,
bool, bool)
rpos::actions::MoveAction     moveTo(     const     std::vector<
rpos::core::Location>&, rpos::robot::option::MoveOption&)
```

rpos::actions::MoveAction moveTo(const rpos::core::Location&, rpos::robot::option::MoveOption&)
rpos::actions::MoveAction moveBy(const rpos::core::Direction& direction)
rpos::actions::MoveAction rotateTo(const rpos::core::Rotation&)
rpos::actions::MoveAction rotate(const rpos::core::Rotation&)
rpos::actions::MoveAction getCurrentAction()
rpos::features::motion_planner::Path searchPath(const rpos::core::Location& location)
rpos::actions::MoveAction goHome()
rpos::actions::SweepMoveAction startSweep()
rpos::actions::SweepMoveAction sweepSpot(const rpos::core::Location& location)
int getBatteryPercentage()
bool getBatteryIsCharging()
bool getDCIsConnected()
int getBoardTemperature()
std::string getSDPVersion()
std::string getSDKVersion()
rpos::features::system_resource::LaserScan getLaserScan()
bool restartModule(rpos::features::system_resource::RestartMode mode = rpos::features::system_resource::RestartModeSoft)
bool setSystemParameter(const std::string& param, const std::string& value)
std::string getSystemParameter(const std::string& param)
rpos::features::system_resource::DeviceInfo getDeviceInfo()
void startCalibration(rpos::features::system_resource::CalibrationType type)
void stopCalibration()
rpos::features::system_resource::BaseHealthInfo getRobotHealth()
void clearRobotHealth(int errorCode)
bool configurateNetwork(rpos::features::system_resouce::NetworkMode mode, const std::map<std::string, std::string>& options)

> std::map<std::string, std::string> getNetworkStatus()
>
> bool getSensors(std::vector<features::impact_sensor::ImpactSensorInfo>& sensors)
>
> bool getSensorValues(std::map<features::impact_sensor::impact_sensor_id_t, features::impact_sensor::ImpactSensorValue>& values)
>
> bool getSensorValues(const std::vector<features::impact_sensor::impact_sensor_id_t>& sensorIds, std::vector<features::impact_sensor::ImpactSensorValue>& values)
>
> bool getSensorValue(features::impact_sensor::impact_sensor_id_t sensorId, features::impact_sensor::ImpactSensorValue& value)
>
> void setCompositeMap(const rpos::robot_platforms::objects::CompositeMap& , const core::Pose&)
>
> rpos::robot_platforms::objects::CompositeMap getCompositeMap()

Function inherits from rpos::core::RobotPlatform Class

> std::vector<Feature> getFeatures()
>
> template<class RobotPlatformT> RobotPlatformT cast()

`SlamwareCorePlatform(boost::shared_ptr<detail::SlamwareCorePlatformImpl>)` Constructor

SDK internal use only.

`SlamwareCorePlatform(const SlamwareCorePlatform&)` Constructor

Copy constructor.

`SlamwareCorePlatform& operator=(const SlamwareCorePlatform&)` Operator

Assignment operator.

`SlamwareCorePlatform connect(const std::string&, int, int)`

Connect to the specified Slamware equipment.

Parameter

Name	Type	Description
host	const std::string&	IP host of Slamware Core
port	int	Post of Slamware Core, 1445 usually.
timeout_in_ms	int	Timeout(unit: millisecond).

`void disconnect()`

Disconnect Slamware CORE equipment.

`std::vector<rpos::core::Line> getWalls()`

Get all virtual walls information.

`bool addWall(const rpos::core::Line&)`

Add a virtual wall information.

`bool addWalls(const std::vector<rpos::core::Line>&)`

Add several virtual walls information.

`bool clearWallById(const rpos::core::SegmentID&)`

Remove the specified virtual wall information by ID.

`bool clearWalls()`

Remove all virtual walls information.

`std::vector<rpos::features::location_provider::MapType> getAvailableMaps()`

Get all the map types the Slamware CORE provides.

`rpos::features::location_provider::Map getMap( rpos::features::location_provider::MapType, rpos::core::RectangleF, rpos::features::location_provider::MapKind)`

Get the map information by the given map type and the scope of rectangle of the Slamware CORE.

Parameter

Name	Type	Description
type	rpos::features::location_provider::MapType	Data type of the map
area	core::RectangleF	The area of the map
kind	rpos::features::location_provider::MapKind	Map type

Sample

```
rpos::feature::location_provider:MapType mapType =
rpos::feature::location_provider:MapType::MapTypeBitmap8Bit;
rpos::feature::location_provider:Mapkind mapKind =
rpos::feature::location_provider:MapKind::EXPLORERMAP;
rpos::core::Rectangle knownArea = robotPlatform.getKnownArea(mapType,
mapKind);
rpos::feature::location_provider:Map map =
robotPlatform.getMap(mapType, knownArea, mapKind);
```

Note: SWEEPERMAP is available in the mapkind of vacuum robot edition SLAMWARE

`bool setMap( const rpos::features::location_provider::Map&, rpos::features::location_provider::MapType, rpos::features::location_provider::MapKind, bool partially)`

Upload the map information with the given map type to the Slamware CORE.

Parameter

Name	Type	Description
map	rpos::features::location_provider::Map	Map
type	rpos::features::location_provider::MapType	Data type pf the map
kind	rpos::features::location_provider::MapKind	Map type
partially	bool	Whether update the map partially

Sample

```
rpos::feature::location_provider:MapType mapType =
rpos::feature::location_provider:MapType::MapTypeBitmap8Bit;
rpos::feature::location_provider:Mapkind mapKind =
rpos::feature::location_provider:MapKind::EXPLORERMAP;
```

```
rpos::core::Rectangle knownArea = robotPlatform.getKnownArea(mapType, mapKind);
rpos::feature::location_provider:Map map = robotPlatform.getMap(mapType, knownArea,
mapKind);
bool bRet = robotPlatform.setMap(map, mapType, mapKind);
```

bool setMap(const core::Pose& pose, const rpos::features::location_provider::Map&, rpos::features::location_provider::MapType, rpos::features::location_provider::MapKind, bool partially)

上载指定地图类型指定区域的地图数据到该 Slamware CORE。

Parameter

名称	类型	说明
pose	core::Pose	机器人的 pose 信息
map	rpos::features::location_provider::Map	地图
type	rpos::features::location_provider::MapType	地图数据类型
kind	rpos::features::location_provider::MapKind	地图类型
partially	bool	是否部分更新地图

Sample

```
rpos::core::Pose pose;
rpos::feature::location_provider:MapType mapType =
rpos::feature::location_provider:MapType::MapTypeBitmap8Bit;
rpos::feature::location_provider:Mapkind mapKind =
rpos::feature::location_provider:MapKind::EXPLORERMAP;
rpos::core::Rectangle knownArea = robotPlatform.getKnownArea(mapType, mapKind);
rpos::feature::location_provider:Map map = robotPlatform.getMap(mapType, knownArea,
mapKind);
bool bRet = robotPlatform.setMap(pose, map, mapType, mapKind);
```

rpos::core::RectangleF getKnownArea(rpos::features::location_provider::MapType, rpos::features::location_provider::MapKind)

Get the scope of map which is mapped by the given map type.

Sample

```
rpos::feature::location_provider:MapType mapType =
rpos::feature::location_provider:MapType::MapTypeBitmap8Bit;
rpos::feature::location_provider:Mapkind mapKind =
rpos::feature::location_provider:MapKind::EXPLORERMAP;
```

```
rpos::core::Rectangle knownArea = robotPlatform.getKnownArea(mapType, mapKind);
```

`bool clearMap()`

Clear map data.

`bool clearMap(rpos::features::location_provider::MapKind kind)`

Clear map data with specified map type.

`rpos::core::Location getLocation()`

Get the location of robot in the coordinate system.

`rpos::core::Pose getPose()`

Get the pose of robot in the coordinate system.

`bool setPose(const core::Pose&)`

Upload the pose of robot in the coordinate system, and return whether or not the function succeeded.

`bool getMapLocalization()`

Whether get map localization.

`bool setMapLocalization(bool)`

Set whether get map localization.

`bool getMapUpdate()`

Whether get map update.

`bool setMapUpdate(bool)`

Set whether get map update.

`int getLocalizationQuality()`

Get the reliability of laser points (the return value is from 0~100. The higher the value is, the localization laser points are more reliable. Laser points with reliability value more than 50 are recommended.

`rpos::actions::MoveAction moveTo(`
`const std::vector< rpos::core::Location>&,`
`bool,bool)`

Make robot move along the specified path. The robot will try to move to the point in route one by one harmoniously and automatically avoid obstacles.

Please refere to `rpos::action::MoveAction MoveTo(const std::vector<rpos::core::Location>&, bool, bool)` for detailed parameters.

`rpos::actions::MoveAction moveTo(const rpos::core::Location&, bool, bool)`

Make robot move to the target location.

Please refere to `rpos::action::MoveAction MoveTo(const rpos::core::Location&, bool, bool)` for detailed parameters.

`rpos::actions::MoveAction moveTo(`
`const std::vector< rpos::core::Location>&,`
`rpos::robot::option::MoveOption &)`

Make the robot follow specified path (The robot will pass the points in the path one by one. The moving mode and heading direction during the moving process can be decided by the MoveAction parameters).

Parameter

Name	Type	Description
locations	const std::vector <rpos::core::Location>&	The points that the robot will pass by.
moveOption	rpos::robot::option::MoveOption	If the robot is executing other moving operations, this parameter decides whether to add as new points or replaced the existed ones.

Please refer to rpos::robot::option::MoveOption for details of moveOption.

Sample

```
std::vector<rpos::core::Location> locations;
rpos::core::Location location(1,1);
locations.push
```

```
rpos::robot::option::MoveAction moveOption;
moveOption.appending = false;
moveOption.isMilestone = true;
rpos::core::Pose pose(Rotation(2));
rpos::robot::heading::RobotHeading
robotHeading(rpos::robot::heading::HeadingMode::HeadingModeFixAngle, pose);
moveOption.robotHeading = robotHeading;
rpos::actions::MoveAction moveInfo = robotPlatform.moveTo(locations, moveOption);
```

`rpos::actions::MoveAction moveTo(const rpos::core::Location&, rpos::robot::option::MoveOption&)`

Make the robot follow specified path(The robot will pass the points in the path one by one. The moving mode and heading direction during the moving process

For details, please refer to: `rpos::actions::MoveAction moveTo(const std::vector<rpos::core::Location>&,rpos::robot::option::MoveAction&)`

`rpos::actions::MoveAction moveBy(const rpos::core::Direction&)`

Control the moving direction of the robot.

Parameter

Name	Type	Description
direction	const rpos::core::Direction&	The moving direction of the robot.

Sample

```
rpos::core::ACTION_DIRECTION actionDirection = rpos::core::ACTION_DIRECTION::FORWARD;
rpos::core::Direction direction(actionDirection);
rpos::actions::MoveAction moveBy = platform.moveBy(direction);
```

Note: the value definition of direction's parameter rpos::core::ACTION_DIRECTION is as below:

Value	Description
FORWARD	Move forward.
BACKWARD	Move backward.
TURNRIGHT	Move right.
TURNLEFT	Move left.

`rpos::actions::MoveAction rotateTo(const rpos::core::Rotation&)`

Make robot turn to a certain angle.

`rpos::actions::MoveAction rotate(const rpos::core::Rotation&)`

Make robot turn a certain angle.

`rpos::actions::MoveAction getCurrentAction()`

Get the current action which robot is doing.

You can use Action::isEmpty() function to judge if the action exist. If robot is doing nothing, the Action::isEmpty() function will return true.

`rpos::features::motion_planner::Path searchPath(const rpos::core::Location& location)`

Make the robot find the path to the specified location by using the embedded algorithm.

`rpos::actions::SweepMoveAction startSweep()`

Let the robot start sweep(applied to sweep robot version only)

`pos::actions::SweepMoveAction sweepSpot(const rpos::core::Location& location)`

Let the robot do spot cleaning. (applied to sweep robot version only)

`rpos::actions::MoveAction goHome()`

Go back to the charge station under the sweep mode.

`int getBatteryPercentage()`

Get remaining capacity (Unit: percent). 0 means the battery is completely drained, and 100 means the battery is full.

`bool getBatteryIsCharging()`

Check if robot is under the status of battery charge.

`bool getDCIsConnected()`

Check if external power is connected.

`int getBoardTemperature()`

Get the body temperature.

Unit is 0.1℃. E.g., if 452 is returned, it means the robot core temperature is 45.2℃.

`std::string getSDPVersion()`

Get version number of board.

`std::string getSDKVersion()`

Get version number of SDK.

`rpos::features::system_resource::LaserScan getLaserScan()`

Get original data of laser scanning.

bool restartModule(rpos::features::system_resource::RestartMode mode = rpos::features::system_resource::RestartModeSoft)

Restart operation with the specified restart mode(soft default).
RestartMode:

- RestartModeSoft, restarts the SDK quickly. Suggested use.
- RestartModeHard, restarts slowly and needs several minutes. Not recommended.

`bool setSystemParameter(const std::string& param, const std::string& value)`

Set system parameter.

Parameter

Name	Type	Description
param	const std::string&	The set parameter name
value	Const std::string &	The set parameter value

Note: Currently, it only supports the settings of the system speed.

The param could only be set as SYSPARAM_ROBOT_SPEED

Value can be set as below:

1.SYSVAL_ROBOT_SPEED_HIGH （High）

2.SYSVAL_ROBOT_SPEED_MEDIUM （Medium）

3.SYSVAL_ROBOT_SPEED_LOW （Low）

Sample

```
Bool bRet = platform.setSystemParameter(SYSPARAM_ROBOT_SPEED,
SYSVAL_ROBOT_SPEED_HIGH);
```

`std::string getSystemParameter(const std::string& param)`

Get system parameter.

Parameter

Name	Type	Description
param	const std::string&	The system parameter name that is expected to get.

Note: Currently, it only supports the settings of the system speed.

The param could only be set as SYSPARAM_ROBOT_SPEED

Sample

```
std::string robotSpeed = platform.getSystemParameter(SYSPARAM_ROBOT_SPEED);
```

`rpos::features::system_resource::DeviceInfo getDeviceInfo()`

Get device information including device ID, manufacturerID, manufacturerName, modelID, modelName, hardware version, software version. For the detail of the return value, please refer to rpos::features::system_resource::DeviceInfo Class

Void startCalibration(rpos::features::system_resource::CalibrationType type)

The robot starts compass calibration.

Void stoptCalibration()

The robot stops compass calibration.

rpos::features::system_resource::BaseHealthInfo getRobotHealth()

Get the current status information of the robot.

void clearRobotHealth(int errorCode)

Clear the error status information of the robot.

bool configureNetwork(rpos::features::system_resource::NetworkMode mode, const std::map<std::string, std::string>& options)

Configure the network information for the robot.

Parameter

Network Status	ssid	password	channel
NetworkModeAp	Optional field	Optional field	Optional field
NetworkModeStation	Required field	Optional field	--
NetworkModeWifiDisabled	--	--	--

Note: currently, only the above three modes are supported. The requirements for the ssid, password and channel are showed in the above table.

Sample

```
std::map<std::string, std::string> options;
options[“ssid”] = “Slamtec”;
options[“password”] = “slamtect”;
Bool bRet =
platform.configureNetwork(rpos::features::system_resource::NetworkMode::NetworkModeS
tation,options);
```

std::map<std::string, std::string> getNetworkStatus()

Get the current network status of the robot.

Note: currently the return results only contain the value of mode, ssid and ip.

`bool getSensors(std::vector<ImpactSensorInfo>& sensors);`

Used for getting the information of all the impact sensors on the robot, the return value is a ImpactSensorInfo list.

The dada structure of ImpactSensorInfo is as below:

```
struct ImpactSensorInfo {
        impact_sensor_id_t id;
        rpos::core::Pose pose;
        ImpactSensorType type;
        float refreshFreq;
};
```

Field description:

Field name	Unit	Description
Id		Id will be used in the next API.
Pose		Pose represents the installation location of the sensors in the robot.
Type		If the robot is performing other actions, the value represents the type of the sensors. It will be one of ImpactSensorTypeDigital or ImpactSensorTypeAnalog. The former represents normal impact sensor and only has two kinds of status: collided or not collided; the latter represents range sensors such as ultra sonic distance measurement sensor or infrared distancemeasuring sensor, etc.
refreshFreq	Hz	It represents the refresh rate of the sensors. The units come out as times per second.

`bool getSensorValues(std::map<impact_sensor_id_t, ImpactSensorValue>& values);`

Used for getting the current status of the impact sensors, the return value is map. Key field is the id obtained from the last API; value field is ImpactSensorValue type and the data structure is as below:

```
struct ImpactSensorValue {
        impact_sensor_timestamp_t time;
        float value;
};
```

Field description:

Field name	Type	Unit	Description
Time	Long long	Microsecond	It represents the time when getting the data.

Value	Float	Meter	It represents the detected distance between the impact sensor and the obstacle. If it is a digital sensor, 0~FLT_EPSILON represents collided while FLT_MAX for not collided (“<FLT_EPSILON” is recommended for judgement); if it is an analog sensor, the value stands for the distance between the impact sensor and the obstacle. FLT_MAX represents no obstacle discovered (“<1000” is recommended for judgement and 1000 is twice the length of the longest axis in the scenario）.

```
bool getSensorValues(const
std::vector<features::impact_sensor::impact_sensor_id_t>&
sensorIds,
std::vector<features::impact_sensor::ImpactSensorValue>&
values)
```

Get value of specified sensor. The return value is ImpactSensorValue array.

```
bool
getSensorValue(features::impact_sensor::impact_sensor_id_t
sensorId, features::impact_sensor::ImpactSensorValue&
value)
```

Get value of specified sensor. The return value is ImpactSensorValue.

```
void setCompositeMap(const
rpos::robot_platforms::objects::CompositeMap& , const
core::Pose& )
```

Set the map information.

```
rpos::robot_platforms::objects::CompositeMap
getCompositeMap()
```

Get the map information.

Date	Description
2014-9-2	Initial version.
2015-6-8	Rename to Slamware SDK API Reference, as well as add plenty of content.
2015-12-30	Add introduction for the two main APIs related to impact sensor in SDK. Add document for the API: rpos::features::location_provider::BitmapMap::setMapData Remove the logo of RoboPeak
2016-04-12	Add document for start sweep and spot cleaning APIs.
2016-04-19	Replaced the image in the cover
2016-05-26	Updated the layout
2016-06-14	Removed the duplicated feature introduction in SlamwareCorePlatform Class
2016-07-06	Added DeviceInfo Class description.
2016-11-23	Added setCompositeMap and getCompositeMap interfaces.

Image and Table Index