RTMaps

RTMaps v3.4 (13/10/2010)

Generated by Doxygen 1.7.2

Wed Oct 13 2010 18:59:21

Contents

Chapter 1

Main Page

1.1 How to use this documentation

This documentation is the reference of the RTMaps SDK. Various entries are available, depending on what your are looking for, and what you know.

- You know the name of the class: the menus "Class Hierarchy", "Alphabetical List" and "Class List" give a direct access to the documentation of a class.
- You know the name of the function, define, typedef...: under "Class Members" you can find the documentation of a member of a class, while "File Members" offers a list of all the other documented items, for example C-style functions, defines or typedefs.
- Under "Related Pages" is the "Deprecated List". It lists all deprecated features that will be removed in the next version.
- An important section, especially for beginners, is called "Modules".

This is only a reference documentation. Don't forget to read the Developer's manual!

Chapter 2

Deprecated List

Member MAPS::AbsoluteTime2Integer(MAPSAbsoluteTime &time) Transforms an absolute time to an integer, for absolute time comparison

Member MAPS::Integer2AbsoluteTime(MAPSInt64 integer, MAPSAbsoluteTime &time) Transforms an integer to an absolute time

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 5

Class Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 8

Module Documentation

8.1 Basic data structures

Classes

- class MAPSCouple< T >

  Template class for couple data.

- class MAPSBasicListItem

  The basic list item class.

- class MAPSListIterator

  The list iterator class.

- class MAPSListItem< T >

  The list item template class.

- class MAPSBasicList

  The base class for the MAPSList template class.

- class MAPSList< T >

  The RTMaps double linked list template class.

- class MAPSList< T >::MAPSIterator

  Iterator on MAPSList objects.

- class MAPSStackedString< BUFFER_SIZE >

  The RTMaps flexible and fast string template class.

- class MAPSStreamedString

  The RTMaps streamed string class.

- class MAPSArray< T >

  The RTMaps array class.

- class MAPSPair< TKey, TContent >

  The RTMaps (key,content) pair template class.

- class MAPSHashTableIterator

  The RTMaps hash table iterator.

- class MAPSHashTable< TKey, TContent, H >
- class MAPSStringHashTable< TContent >

  The RTMaps string hash table class.

- class MAPSIntegerHashTable< TContent >

  The RTMaps integer hash table template class.

- class MAPSPtrHashTable< TContent >

  The RTMaps pointer hash table template class.

Defines

- #define MAPSForallItems(it, L)
- #define MAPSForall(x, L)
- #define MAPSForallPtr(x, L)

Typedefs

- typedef MAPSStackedString< 40 > MAPSString

  The RTMaps flexible and fast string class.

Functions

- template<typename T >
  MAPSMatrix2< T > MAPSMatrix2::operator∗ (const MAPSMatrix2< T > &A, const MAPSMatrix2< T > &B)

  MAPSMatrix2 multiplication operator.

- template<typename T >
  MAPSMatrix2< T > MAPSMatrix2::operator+ (const MAPSMatrix2< T > &A, const MAPSMatrix2< T > &B)

  MAPSMatrix2 addition operator.

- template<typename T >
  MAPSMatrix2< T > MAPSMatrix2::operator- (const MAPSMatrix2< T > &A, const MAPSMatrix2< T > &B)

  MAPSMatrix2 substraction operator.

- template<typename T >
  MAPSArray< T > MAPSArray::operator∗ (const MAPSMatrix2< T > &A, const MAPSArray< T > &b)

  Multiplication of a MAPSMatrix2 with a MAPSArray.

- template<typename T >
  MAPSMatrix2< T > MAPSMatrix2::operator∗ (const MAPSMatrix2< T > &A, const T &b)

  Multiplication of a MAPSMatrix2 with a scalar.

- template<typename T >
  MAPSMatrix2< T > MAPSMatrix2::operator/ (const MAPSMatrix2< T > &A, const T &b)

  Division of a MAPSMatrix2 by a scalar.

- template<typename T >
  MAPSArray< T > MAPSArray::operator+ (const MAPSArray< T > &a, const MAPSArray< T > &b)

  MAPSArrays addition.

- template<typename T >
  MAPSArray< T > MAPSArray::operator- (const MAPSArray< T > &a, const MAPSArray< T > &b)

  MAPSArrays substraction.

- template<typename T >
  MAPSArray< T > MAPSArray::operator+ (const MAPSMatrix2< T > &a, const MAPSArray< T > &b)

  Addition of a MAPSMatrix2 with a MAPSArray.

8.1.1 Detailed Description

The following classes are the basis for all the data structures used by the RTMaps Engine. They are generally simpler but similar to their STL counterparts. These classes are available to developers using the RTMaps SDK to ensure portable code as they have been especially designed to be fully cross-platform.

8.1.2 Define Documentation

8.1.2.1 #define MAPSForall(*x*, *L*)

Helper macro to walk through a MAPSList with an access to the list elements.

```
MAPSList<MAPSString> li;
// ... populate li ...
// Displays li elements:
MAPSString x;
MAPSForall(x,li) {
    std::cout << x << std::endl;
}
```

Parameters

x	a *T* object
L	a MAPSList<T>

8.1.2.2 #define MAPSForallItems(*it*, *L*)

Helper macro to walk through a MAPSBasicList. Typical use is like:

```
MAPSList<MAPSString> li;
// ... populate the list ...
MAPSListIterator iter;
// Walk through the list:
MAPSForallItems(iter,li) {
    std::cout << li[iter] << std::endl;
}
```

Parameters

it	a MAPSListIterator
L	a MAPSBasicList

8.1.2.3 #define MAPSForallPtr(*x*, *L*)

Helper macro to walk through a MAPSList with an access to a pointer to the list elements.

```
MAPSList<MAPSString> li;
// ... populate li ...
// Displays li elements:
MAPSString* x;
MAPSForallPtr(x,li) {
    std::cout << (*x) << std::endl;
}
```

Parameters

x	a $T*$ object
L	a MAPSList<T>

8.1.3 Typedef Documentation

8.1.3.1 MAPSStackedString< 40 > MAPSString

The RTMaps flexible and fast string class.

See also

MAPSStreamedString

8.1.4 Function Documentation

8.1.4.1 template<typename T > MAPSArray< T > operator+ (const MAPSMatrix2< T > & *a*, const MAPSArray< T > & *b*) `[related, inherited]`

Addition of a MAPSMatrix2 with a MAPSArray.

The result is a MAPSArray. Only the first column of the matrix *a* is added to the array *b*.

8.2 Input definition macros

Defines

- #define MAPS_BEGIN_INPUTS_DEFINITION(className)

  Starts the definition of the inputs of a module.

- #define MAPS_INPUT(namex, filter, typex)

  Basic definition of an input.

- #define MAPS_END_INPUTS_DEFINITION

  Ends the definition of the inputs of a module.

8.2.1 Define Documentation

8.2.1.1 #define MAPS_BEGIN_INPUTS_DEFINITION(*className*)

Starts the definition of the inputs of a module.

Parameters

className	The module class name

8.2.1.2 #define MAPS_INPUT(*namex, filter, typex*)

Basic definition of an input.

Parameters

namex	a simple string used to further identify the input (e.g. "input")
filter	the kind of expected data, a MAPSTypeFilterBase (e.g. MAPS::FilterInteger)
typex	the reader type of the input (e.g. MAPS::FifoReader, MAPS::SamplingReader)

8.3 Output definition macros

Defines

- #define MAPS_BEGIN_OUTPUTS_DEFINITION(className)

  Starts the definition of the outputs of a module.

- #define MAPS_OUTPUT(name, value, namex, unit, size)

  Basic definition of an output.

- #define MAPS_OUTPUT_FIFOSIZE(name, value, namex, unit, size, fifosize)

  Definition of an output with control of the FIFO size.

- #define MAPS_OUTPUT_USER_STRUCTURE(name, structureName)

  Definition of an output using a user structure.

- #define MAPS_END_OUTPUTS_DEFINITION

  Ends the definition of the outputs of a module.

8.3.1 Define Documentation

8.3.1.1 #define MAPS_BEGIN_OUTPUTS_DEFINITION(*className*)

Starts the definition of the outputs of a module.

Parameters

className	The module class name

8.3.1.2 #define MAPS_OUTPUT(*name, value, namex, unit, size*)

Basic definition of an output.

Parameters

name	a simple string used to further identify the output (e.g. "output")
value	the type of the output data, a MAPSTypeInfoValue (e.g. MAPS::Integer)
namex	An optional name associated to the output data type. Can be `NULL`.
unit	A string describing the unit of the output data (e.g. "m/s"). Can be `NULL`.
size	The size of the output. Use 1 for a single piece of data, 0 if you do not know the data size yet, but do not forget to allocate the output later. MAPSOutput::AllocOutputBuffer documentation gives more details about output buffer allocation.

8.3.1.3 #define MAPS_OUTPUT_FIFOSIZE(*name, value, namex, unit, size, fifosize*)

Definition of an output with control of the FIFO size.

Basic definition of an output.

Parameters

name	a simple string used to further identify the output (e.g. "output")
value	the type of the output data, a MAPSTypeInfoValue (e.g. MAPS::Integer)
namex	An optional name associated to the output data type. Can be `NULL`.
unit	A string describing the unit of the output data (e.g. "m/s"). Can be `NULL`.
size	The size of the output. Use 1 for a single piece of data, 0 if you do not know the data size yet, but do not forget to allocate the output later. MAPSOutput::AllocOutputBuffer documentation gives more details about output buffer allocation.

Parameters

fifosize	size of the FIFO (default size of the output FIFO is 16)

8.3.1.4 #define MAPS_OUTPUT_USER_STRUCTURE(*name, structureName*)

Definition of an output using a user structure.

If you need to transmit your own data structure between RTMaps components, you can define your own structure. It is the goal of this macro.

Parameters

name	a simple string used to further identify the output (e.g. "output")
structure-Name	the class name of your own structure

8.4 Property definition macros

Defines

- #define MAPS_BEGIN_PROPERTIES_DEFINITION(className)

  Starts the definition of the properties of a module.

- #define MAPS_PROPERTY(name, value, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_UNIT(name, value, unit, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_ENUM(name, enumstr, selected, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_ENUM_UNIT(name, enumstr, selected, unit, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_READ_ONLY(name, value)

  Definition of a read-only property.

- #define MAPS_PROPERTY_READ_ONLY_UNIT(name, value, unit)

  Definition of a read-only property with a specified unit.

- #define MAPS_END_PROPERTIES_DEFINITION

  Ends the definition of the properties of a module.

8.4.1 Define Documentation

8.4.1.1 #define MAPS_BEGIN_PROPERTIES_DEFINITION(*className*)

Starts the definition of the properties of a module.

Parameters

className	The module class name

8.4.1.2 #define MAPS_PROPERTY(*name, value, needs2BeInitialized, canBeChangedAfterInstantiation*)

Basic definition of a property.

Parameters

name	a simple string used to further identify the output (e.g. "property")
value	the initial value of the property. Its type is automatically deduced from this value. Use `(char*)NULL` to initialize an empty string.
needs2BeIniti	when set to *true*, the property must be initialized before run
can-BeChangedA terInstantia-tion	when set to *false*, the property cannot be changed while RTMaps is running

8.4.1.3 #define MAPS_PROPERTY_ENUM(*name, enumstr, selected, needs2BeInitialized, canBeChangedAfterInstantiation*)

Basic definition of a property.

Parameters

name	a simple string used to further identify the output (e.g. "property")
enumstr	a string of enumeration values, e.g. "value0|value1|value2"
selected	the initial selected index (starting from 0). -1 means not yet selected.
needs2BeIniti	when set to *true*, the property must be initialized before run
can-BeChangedA terInstantia-tion	when set to *false*, the property cannot be changed while RTMaps is running

8.4.1.4 #define MAPS_PROPERTY_ENUM_UNIT(*name, enumstr, selected, unit, needs2BeInitialized, canBeChangedAfterInstantiation*)

Basic definition of a property.

Basic definition of a property.

Parameters

name	a simple string used to further identify the output (e.g. "property")
enumstr	a string of enumeration values, e.g. "value0|value1|value2"
selected	the initial selected index (starting from 0). -1 means not yet selected.
needs2BeIniti	when set to *true*, the property must be initialized before run

can-BeChangedAfterInstantia-tion	when set to *false*, the property cannot be changed while RTMaps is running

Parameters

unit	a simple string used for display only

8.4.1.5 #define MAPS_PROPERTY_READ_ONLY(*name, value*)

Definition of a read-only property.

Parameters

name	a simple string used to further identify the output (e.g. "property")
value	the initial value of the property. Its type is automatically deduced from this value. Use `(char*)NULL` to initialize an empty string.

8.4.1.6 #define MAPS_PROPERTY_READ_ONLY_UNIT(*name, value, unit*)

Definition of a read-only property with a specified unit.

Definition of a read-only property.

Parameters

name	a simple string used to further identify the output (e.g. "property")
value	the initial value of the property. Its type is automatically deduced from this value. Use `(char*)NULL` to initialize an empty string.

Parameters

unit	a simple string used for display only

8.4.1.7 #define MAPS_PROPERTY_UNIT(*name, value, unit, needs2BeInitialized, canBeChangedAfterInstantiation*)

Basic definition of a property.

Basic definition of a property.

Parameters

name	a simple string used to further identify the output (e.g. "property")
value	the initial value of the property. Its type is automatically deduced from this value. Use `(char*)NULL` to initialize an empty string.

needs2BeIniti	when set to *true*, the property must be initialized before run
can-BeChangedA terInstantia-tion	when set to *false*, the property cannot be changed while RTMaps is running

Parameters

unit	a simple string used for display only

8.5 Action definition macros

Defines

- #define MAPS_BEGIN_ACTIONS_DEFINITION(className)

  Starts the definition of the actions of a module.

- #define MAPS_ACTION(name, proc)

  Basic definition of an action.

- #define MAPS_ACTION2(name, proc, allowedWhenDead)

  Second definition of an action.

- #define MAPS_END_ACTIONS_DEFINITION

  Ends the definition of the actions of a module.

8.5.1 Define Documentation

8.5.1.1 #define MAPS_ACTION(*name,* *proc*)

Basic definition of an action.

This action definition is for use only while RTMaps is running.

Parameters

name	a simple string used to further identify the output (e.g. "action")
proc	the callback function to launch when the action is called (e.g. `MyModule::anAction`)

8.5.1.2 #define MAPS_BEGIN_ACTIONS_DEFINITION(*className*)

Starts the definition of the actions of a module.

Parameters

className	The module class name

8.6 Component design

Classes

- class MAPSOutput

  The RTMaps Module Output class.

- class MAPSInput

  The RTMaps Module Input class.

- class MAPSProperty

  The RTMaps Module Property class.

- class MAPSAction

  The RTMaps Module Action class.

- class MAPSRunShutdownListener

  The MAPSRunShutdownListener interface.

- class MAPSTimeStateListener

  The MAPSTimeStateListener interface.

- class MAPSRecordingStateListener

  The MAPSRecordingStateListener interface.

- class MAPSSynchronizer

  The RTMaps Synchroniser tool.

- class MAPSModule

  The base class for all RTMaps modules.

- class MAPSComponent

  The base class for all RTMaps components.

Modules

- Input definition macros
- Output definition macros
- Property definition macros
- Action definition macros

Component definition macros

- #define MAPS_COMPONENT_DEFINITION(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions)

  One of the standard macros required for the definition of an RTMaps component.

- #define MAPS_COMPONENT_DEFINITION_DOC(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions, doc)

  One of the standard macros required for the definition of an RTMaps component.

- #define MAPS_COMPONENT_DEFINITION_UNIQUE(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions)

  One of the standard macros required for the definition of an RTMaps component.

- #define MAPS_COMPONENT_DEFINITION_REGISTRATION(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions)

  One of the standard macros required for the definition of an RTMaps component.

Component class declaration macros

- #define MAPS_COMPONENT_STANDARD_HEADER_CODE(component)
- #define MAPS_CLOCK_COMPONENT_HEADER_CODE(component)
- #define MAPS_COMPONENT_DYNAMIC_HEADER_CODE(component)
- #define MAPS_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(component)
- #define MAPS_COMPONENT_REGISTERING_HEADER_CODE(component)
- #define MAPS_CHILD_COMPONENT_HEADER_CODE(component, parent)

  Include this macro inside your component class definition if its parent is a descendant of MAPSComponent.

- #define MAPS_CHILD_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(component, parent)

  Include this macro inside your component class definition if its parent is a descendant of MAPSComponent and you need to implement your own constructor.

- #define MAPS_PARENT_COMPONENT_HEADER_CODE(component, parent)

  Include this macro inside a parent component class definition.

- #define MAPS_PARENT_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(component, parent)

  Include this macro inside a parent component class definition when you need to implement your own constructor.

8.6.1 Define Documentation

8.6.1.1 #define MAPS_CHILD_COMPONENT_HEADER_CODE(*component, parent*)

Include this macro inside your component class definition if its parent is a descendant of MAPSComponent.

This macro replaces MAPS_COMPONENT_STANDARD_HEADER_CODE.

Parameters

component	The class name of the component
parent	The class name of the parent (e.g. MAPSStream8IOComponent)

8.6.1.2 #define MAPS_CHILD_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(*component, parent*)

Include this macro inside your component class definition if its parent is a descendant of MAPSComponent and you need to implement your own constructor.

This macro replaces MAPS_COMPONENT_STANDARD_HEADER_CODE.

Parameters

component	The class name of the component
parent	The class name of the parent (e.g. MAPSStream8IOComponent)

8.6.1.3 #define MAPS_CLOCK_COMPONENT_HEADER_CODE(*component*)

One of the standard macros required for the definition of an RTMaps component class. This macro has to be placed in the class definition (usually in the header file).

Parameters

component	The class name of the component

Include this macro inside your component class definition if your component implements an RTMaps clock. Your component must derive from MAPSBaseClock in this case.

8.6.1.4 #define MAPS_COMPONENT_DEFINITION(*component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions*)

One of the standard macros required for the definition of an RTMaps component.

When building an RTMaps component, one of these macros must be used in the implementation file.

Parameters

component	the class component name (e.g. MAPSMyComponent)

name	a string used to identify the component (e.g. "myComponent")
version	the version of the component; can be any string, but we recommand the use of a float number when possible (e.g. "2.3")
priority	the default component priority; must be an integer between 0 (lowest) and 255
kind	MAPS::Sequential for a sequential only component, MAPS::Threaded for a threaded only component, or a combination with a '|' if both behaviours are allowed
defaultBehaviour	MAPS::Sequential or MAPS::Threaded; must be coherent with the *kind* parameter
nbOfInputs	an integer indicating the (minimum) number of inputs
nbOfOutputs	an integer indicating the (minimum) number of outputs
nbOfProperties	an integer indicating the (minimum) number of properties
nbOfActions	an integer indicating the (minimum) number of actions

8.6.1.5 #define MAPS_COMPONENT_DEFINITION_DOC(*component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions, doc*)

One of the standard macros required for the definition of an RTMaps component.

Allows to add a short documentation to the component definition. One of the standard macros required for the definition of an RTMaps component.

When building an RTMaps component, one of these macros must be used in the implementation file.

Parameters

component	the class component name (e.g. MAPSMyComponent)
name	a string used to identify the component (e.g. "myComponent")
version	the version of the component; can be any string, but we recommand the use of a float number when possible (e.g. "2.3")
priority	the default component priority; must be an integer between 0 (lowest) and 255
kind	MAPS::Sequential for a sequential only component, MAPS::Threaded for a threaded only component, or a combination with a '|' if both behaviours are allowed
defaultBehaviour	MAPS::Sequential or MAPS::Threaded; must be coherent with the *kind* parameter
nbOfInputs	an integer indicating the (minimum) number of inputs
nbOfOutputs	an integer indicating the (minimum) number of outputs
nbOfProperties	an integer indicating the (minimum) number of properties
nbOfActions	an integer indicating the (minimum) number of actions

Parameters

doc	a string describing the component

8.6.1.6 #define MAPS_COMPONENT_DEFINITION_REGISTRATION(*component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions*)

One of the standard macros required for the definition of an RTMaps component.

Use this macro when you need to perform special operations on component (un)registration. One of the standard macros required for the definition of an RTMaps component.

When building an RTMaps component, one of these macros must be used in the implementation file.

Parameters

component	the class component name (e.g. MAPSMyComponent)
name	a string used to identify the component (e.g. "myComponent")
version	the version of the component; can be any string, but we recommand the use of a float number when possible (e.g. "2.3")
priority	the default component priority; must be an integer between 0 (lowest) and 255
kind	MAPS::Sequential for a sequential only component, MAPS::Threaded for a threaded only component, or a combination with a '|' if both behaviours are allowed
defaultBehaviour	MAPS::Sequential or MAPS::Threaded; must be coherent with the *kind* parameter
nbOfInputs	an integer indicating the (minimum) number of inputs
nbOfOutputs	an integer indicating the (minimum) number of outputs
nbOfProperties	an integer indicating the (minimum) number of properties
nbOfActions	an integer indicating the (minimum) number of actions

Warning

When using this macro, you must use MAPS_COMPONENT_REGISTERING_-HEADER_CODE too.

8.6.1.7 #define MAPS_COMPONENT_DEFINITION_UNIQUE(*component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions*)

One of the standard macros required for the definition of an RTMaps component.

Use this macro when the component can be instantiated only once. One of the standard macros required for the definition of an RTMaps component.

When building an RTMaps component, one of these macros must be used in the implementation file.

Parameters

component	the class component name (e.g. MAPSMyComponent)
name	a string used to identify the component (e.g. "myComponent")
version	the version of the component; can be any string, but we recommand the use of a float number when possible (e.g. "2.3")
priority	the default component priority; must be an integer between 0 (lowest) and 255
kind	MAPS::Sequential for a sequential only component, MAPS::Threaded for a threaded only component, or a combination with a '|' if both behaviours are allowed
defaultBehaviour	MAPS::Sequential or MAPS::Threaded; must be coherent with the *kind* parameter
nbOfInputs	an integer indicating the (minimum) number of inputs
nbOfOutputs	an integer indicating the (minimum) number of outputs
nbOfProperties	an integer indicating the (minimum) number of properties
nbOfActions	an integer indicating the (minimum) number of actions

Warning

When using this macro, you must use MAPS_COMPONENT_REGISTERING_HEADER_CODE too.

8.6.1.8 #define MAPS_COMPONENT_DYNAMIC_HEADER_CODE(*component*)

Include this macro inside your component class definition if you need to create/suppress inputs, outputs, properties or actions dynamically. Then you need to implement MAPSComponent::Dynamic.

Parameters

component	The class name of the component

8.6.1.9 #define MAPS_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(*component*)

Include this macro inside your component class definition if you need to implement your own constructor. This macro replaces MAPS_COMPONENT_STANDARD_HEADER_CODE. You need then to implement `MAPSMyComponent::MAPSMyComponent(const char* componentName, MAPSComponentDefinition& md):MAPSComponent(componentName,md)`

Parameters

component	The class name of the component

8.6.1.10 #define MAPS_COMPONENT_REGISTERING_HEADER_CODE(*component*)

Include this macro inside your component class definition if you need to perform special operations whe the component is (un)registered. You need then to implement `void MAPSMyComponent::Register(void)` and `void MAPSMyComponent::Unregister(void)`, which are static functions.

Parameters

component	The class name of the component

8.6.1.11 #define MAPS_COMPONENT_STANDARD_HEADER_CODE(*component*)

One of the standard macros required for the definition of an RTMaps component class. This macro has to be placed in the class definition (usually in the header file).

Parameters

component	The class name of the component

8.6.1.12 #define MAPS_PARENT_COMPONENT_HEADER_CODE(*component, parent*)

Include this macro inside a parent component class definition.

Parameters

component	The class name of the component
parent	The class name of the parent (usually MAPSComponent).

8.6.1.13 #define MAPS_PARENT_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(*component, parent*)

Include this macro inside a parent component class definition when you need to implement your own constructor.

Parameters

component	The class name of the component
parent	The class name of the parent (usually MAPSComponent).

8.7 Record/replay method design

Classes

- class MAPSOutput

The RTMaps Module Output class.

- class MAPSInput

  The RTMaps Module Input class.

- class MAPSProperty

  The RTMaps Module Property class.

- class MAPSModule

  The base class for all RTMaps modules.

- class MAPSRecordReplayMethod

  The base class for all record/replay methods.

Modules

- Property definition macros
- Action definition macros

Defines

- #define MAPS_RECORD_REPLAY_METHOD_DEFINITION(rrm, namex, version, filter, recordThreaded, replayThreaded, nbPropertiesRecord, nbPropertiesReplay)

  Required for the implementation of an RTMaps record/replay method.

- #define MAPS_RECORD_REPLAY_METHOD_STANDARD_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition.

- #define MAPS_RECORD_REPLAY_METHOD_BLACKBOX_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you intend to implement blackbox features.

- #define MAPS_RECORD_REPLAY_METHOD_PROCESS_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you need preprocessing before recording.

- #define MAPS_RECORD_REPLAY_METHOD_COPY_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you intend to implement special copy features.

- #define MAPS_RECORD_REPLAY_METHOD_HEADING_HINT_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you intend to implement hint header.

8.7.1 Define Documentation

8.7.1.1 #define MAPS_RECORD_REPLAY_METHOD_BLACKBOX_HEADER_CODE(*rrm*)

Include this macro inside your RRM class definition if you intend to implement blackbox features.

You need then to implement MAPSRecordReplayMethod::BlackboxSwith and MAPSRecordReplayMethod::BlackboxDump.

Parameters

rrm	The class name of the RRM

8.7.1.2 #define MAPS_RECORD_REPLAY_METHOD_COPY_HEADER_CODE(*rrm*)

Include this macro inside your RRM class definition if you intend to implement special copy features.

You need then to implement MAPSRecordReplayMethod::Copy.

Parameters

rrm	The class name of the RRM

8.7.1.3 #define MAPS_RECORD_REPLAY_METHOD_DEFINITION(*rrm, namex, version, filter, recordThreaded, replayThreaded, nbPropertiesRecord, nbPropertiesReplay*)

Required for the implementation of an RTMaps record/replay method.

When building a MAPSRecordReplayMethod, this macro must be used in the cpp file.

Parameters

rrm	the class RRM name (e.g. MAPSMyRRM)
namex	a string used to identify the RRM (e.g. "myRRM")
version	the version of the RRM; use a float number if possible (e.g. "2.3")
filter	a MAPSTypeFilterBase value
recordThread	a boolean to indicate whether the record method is threaded
re-playThreaded	a boolean to indicate whether the replay method is threaded
nbProper-tiesRecord	the number of propreties for the record method

nbPropertiesReplay	the number of propreties for the replay method

8.7.1.4 #define MAPS_RECORD_REPLAY_METHOD_HEADING_HINT_HEADER_CODE(*rrm*)

Include this macro inside your RRM class definition if you intend to implement hint header.

You need then to implement MAPSRecordReplayMethod::HeadingHint(MAPSIOElt&).

Parameters

rrm	The class name of the RRM

8.7.1.5 #define MAPS_RECORD_REPLAY_METHOD_PROCESS_HEADER_CODE(*rrm*)

Include this macro inside your RRM class definition if you need preprocessing before recording.

You need then to implement MAPSRecordReplayMethod::DoProcessBufferItem and MAPSRecordReplayMethod::ProcessBufferItem.

Parameters

rrm	The class name of the RRM

8.7.1.6 #define MAPS_RECORD_REPLAY_METHOD_STANDARD_HEADER_CODE(*rrm*)

Include this macro inside your RRM class definition.

You need then to implement MAPSRecordReplayMethod::Store, MAPSRecordReplayMethod::Hint, MAPSRecordReplayMethod::Replay, MAPSRecordReplayMethod::BirthRecord, MAPSRecordReplayMethod::DeathRecord, MAPSRecordReplayMethod::BirthReplay and MAPSRecordReplayMethod::DeathReplay.

Parameters

rrm	The class name of the RRM

Chapter 9

Namespace Documentation

9.1 MAPS Namespace Reference

The main RTMaps namespace.

RTMaps Typing related constants

- const MAPSTypeInfoValue NoMask

  No mask.

- const MAPSTypeInfoValue NoType

  No type.

- const MAPSTypeInfoValue AnyType

  Any type.

- const MAPSTypeInfoValue TypeMask

  Type mask.

- const MAPSTypeInfoValue Structure

  User defined structure.

- const MAPSTypeInfoValue Integer

  Integer (32 bits, signed)

- const MAPSTypeInfoValue Float

  MAPSFloat (double : 64 bits, double precision)

- const MAPSTypeInfoValue TextAscii

  Ascii characters (1 character = 8 bits).

- const MAPSTypeInfoValue TextUnicode

  Unicode characters (1 character = 16 bits)

- const MAPSTypeInfoValue IplImage

  IplImage (Image Processing Library image description structure)

- const MAPSTypeInfoValue MAPSImage

  MAPSImage (RTMaps specific image structure)

- const MAPSTypeInfoValue CANFrame

  CAN frame.

- const MAPSTypeInfoValue Matrix

  MATLAB-Like matrix.

- const MAPSTypeInfoValue RealObject

  Real object (car, tree, etc.)

- const MAPSTypeInfoValue DrawingObject

  Drawing object (line, circle, etc.)

- const MAPSTypeInfoValue Triangles3D

  3D triangles for 3D scene rendering

- const MAPSTypeInfoValue Stream8

  8-bit stream (sound, numerized data)

- const MAPSTypeInfoValue Stream16

  16-bit stream (sound, numerized data)

- const MAPSTypeInfoValue Stream32

  32-bit stream (numerized data)

- const MAPSTypeInfoValue Integer64

  Integer (64 bits, signed)

- const MAPSTypeInfoValue AnyText

  Some textual information, ascii or unicode.

- const MAPSTypeInfoValue AnyInteger

  32 or 64 bits integers.

- const MAPSTypeInfoValue VectorFlag

  Indicates that the piece of data is a vector (an array) of a basic type.

- const MAPSTypeInfoValue FrequencyFlag

Indicates that a frequency is provided with the data.

- const MAPSTypeInfoValue QualityFlag

  Indicates that a quality is transmitted along with the data (for instance a compression or a noise ratio).

- const MAPSTypeInfoValue MiscFlag

  Indicates that 3 miscellaneous integers are transmitted along with the data. See MAPSIOElt::Misc1(), Misc2() and Misc3()

RTMaps Inputs behaviours

- const int FifoReader

  FIFO Reader behaviour.

- const int NeverSkippingReader

  Never Skipping Reader behaviour.

- const int LastOrNextReader

  Last or Next Reader behaviour.

- const int Wait4NextReader

  Wait For Next Reader behaviour.

- const int SamplingReader

  Sampling Reader behaviour.

RTMaps Properties types

- const int BoolProperty

  Boolean property (`false` or `true`)

- const int IntegerProperty

  Integer property (64-bit signed integer)

- const int FloatProperty

  Floating point property (`double`: 64-bit floating point number)

- const int StringProperty

  String property (ASCII string) Enum property (ASCII string)

- const int EnumProperty

RTMaps Type Filters

- const MAPSTypeFilterBase FilterStructure

  Filters any user-defined structure type.

- const MAPSTypeFilterBase FilterInteger

  Filters integer type (same as FilterIntegers)

- const MAPSTypeFilterBase FilterInteger64

  Filters 64 bits integer type (same as FilterIntegers64)

- const MAPSTypeFilterBase FilterFloat

  Filters MAPSFloat type (same as FilterFloats)

- const MAPSTypeFilterBase FilterNumber

  Filters integer or MAPSFloat type (same as FilterNumbers)

- const MAPSTypeFilterBase FilterIntegers

  Filters integer scalars or vectors.

- const MAPSTypeFilterBase FilterIntegers64

  Filters 64 bits integer scalars or vectors.

- const MAPSTypeFilterBase FilterFloats

  Filters MAPSFloat scalars or vectors.

- const MAPSTypeFilterBase FilterNumbers

  Filters integer or MAPSFloat scalars or vectors.

- const MAPSTypeFilterBase FilterOneInteger

  Filters integer type (excludes vectors of integers)

- const MAPSTypeFilterBase FilterOneInteger64

  Filters 64 bits integer type (excludes vectors of integers64)

- const MAPSTypeFilterBase FilterOneFloat

  Filters MAPSFloat type (excludes vectors of MAPSFloat)

- const MAPSTypeFilterBase FilterOneNumber

  Filters integer or MAPSFloat type (excludes vectors)

- const MAPSTypeFilterBase FilterTextAscii

  Filters ASCII text string.

- const MAPSTypeFilterBase FilterTextUnicode

  Filters Unicode (16 bits) text string.

- const MAPSTypeFilterBase FilterImage

  Filters IplImages or MAPSImages.

- const MAPSTypeFilterBase FilterIplImage

  Filters IplImages.

- const MAPSTypeFilterBase FilterMAPSImage

  Filters MAPSImages.

- const MAPSTypeFilterBase FilterCANFrame

  Filters CANFrames.

- const MAPSTypeFilterBase FilterMatrix

  Filters MATLAB-Like matrices.

- const MAPSTypeFilterBase FilterRealObjects

  Filters RTMaps Real Objects.

- const MAPSTypeFilterBase FilterDrawingObjects

  Filters RTMaps Drawing Objects.

- const MAPSTypeFilterBase FilterTriangles3D

  Filters 3D triangles.

- const MAPSTypeFilterBase FilterStream8

  Filters 8-bit data streams.

- const MAPSTypeFilterBase FilterStream16

  Filters 16-bit data streams.

- const MAPSTypeFilterBase FilterStream32

  Filters 32-bit data streams.

- const MAPSTypeFilterBase FilterAudioSignal

  Filters audio signals (either MAPSFloat or Stream8)

- const MAPSTypeFilterBase FilterAny

  Filters any kind of data.

RTMaps kinds of information outputs

- const int Info

  Simple information.

- const int Warning

  Warning.

- const int Error

  Error.

- const int ParserEcho

  Echo from the parser.

RTMaps replay modes

- const int ReplayModeNormal

  Normal replay.

- const int ReplayModeImmediate

  Immediate replay mode (replay ahead of real time)

- const int ReplayModeTimestamp

  Replay using timestamp instead of time of issue.

RTMaps VCR Keys codes

- const int **VCRKeyStop**
- const int **VCRKeyPlay**
- const int **VCRKeyRecord**
- const int **VCRKeyPause**
- const int **VCRKeyRewind**
- const int **VCRKeyForward**
- const int **VCRKeyNext**
- const int **VCRKeyOrganize**
- const int **VCRSlider**
- const int **VCRAllKeys**

RTMaps Engine Keys codes

- const int **KernelKeyRun**
- const int **KernelKeyShutdown**
- const int **DefaultKeysState**

Others RTMaps Constants

- const MAPSString OperatingSystem

  Contains the operating system string information ("Win32","QNX" or "Linux" for instance)

- const int OSBuild

  Contains the OS support build number information.

- const MAPSString Distribution

  Contains the distribution information string ("3.0" for instance)

- const MAPSString KernelVersion

  Contains the engine version string information ("1.0" for instance)

- const MAPSString RTMapsMinorVersion

  Contains the RTMaps minor version (changes in the minor versions preserve backward compatibility with packages). "5" for example -> complete version string will look like "3.0.5".

- const int KernelBuild

  Contains the engine build number information.

- const MAPSString Copyright

  Contains the RTMaps copyright string information.

- const MAPSString License

  Contains the RTMaps license grant string information (depends on the customer)

- const MAPSString ProductName

  Contains the operating system string information ("Win32","QNX" or "Linux" for instance)

- const bool BigEndian

  Tells if we are running on a big-endian platform (`true`) or a little-endian platform (`false`)

- const int DefaultTextSize

  Default allocation size (in characters) for ascii text type outputs.

- const int Infinite

  Infinite number.

- const int ModuleDied

  State indicating a dead RTMaps module.

- const int GotAMessage

State indicating that RTMaps got a Windows message.

- const int TimeOut

  State indicating a time out in RTMaps.

- const int FatalKernelError

  State indicating a fatal engine error.

- const int ErrorException

  State indicating an error in an RTMaps module.

- const int Running

  State telling that RTMaps is running (that time is flowing)

- const int Paused

  State telling that RTMaps is in pause mode.

- const int ShuttingDown

  RTMaps is currently shutting down.

- const int Resetting

  RTMaps is currently resetting.

- const int WaitingForSynchBeforeRun

  RTMaps is currently waiting to be synchronized before running.

- const int DeadState

  State indicating a dead thread or module.

- const int DyingState

  State indicating a dying thread or module.

- const int LivingState

  State indicating a living thread or module.

- const int Threaded

  The component is threaded.

- const int Sequential

  The component is sequential.

C standard library wrapper

These functions should be called instead of their C counterparts to ensures easier cross-platform design of components.

- int Atoi (const char ∗a)

  Ascii to integer conversion.

- MAPSInt64 Atoi64 (const char ∗a)

  Ascii to integer (64 bits) conversion.

- MAPSInt32 Strtol (const char ∗s, char ∗∗endptr, int base)

  Converts the initial part of the string in s to a MAPSInt32 according to the given base.

- MAPSUInt32 Strtoul (const char ∗s, char ∗∗endptr, int base)

  Converts the initial part of the string in s to a MAPSUInt32 according to the given base.

- MAPSInt64 Strtoi64 (const char ∗s, char ∗∗endptr, int base)

  Converts the initial part of the string in s to a MAPSInt64 according to the given base.

- MAPSUInt64 Strtoui64 (const char ∗s, char ∗∗endptr, int base)

  Converts the initial part of the string in s to a MAPSUInt64 according to the given base.

- int Strlen (const char ∗s)

  Calculates the length of the string s.

- const char ∗ Strchr (const char ∗s, char ch)

  Finds character ch in s.

- char ∗ Strchr (char ∗s, char ch)

  Finds character ch in s.

- int Strcmp (const char ∗s1, const char ∗s2)

  String comparison.

- int Stricmp (const char ∗s1, const char ∗s2)

  Lowercase comparison of strings.

- int Strncmp (const char ∗s1, const char ∗s2, MAPSInt64 size)

  Compares the first size characters of two strings.

- char ∗ Strstr (const char ∗s, const char ∗strSearch)

  Returns a pointer to the first occurrence of a search string strSearch in the string s.

- char ∗ Strcpy (char ∗strDest, const char ∗strSrc)

  Copies strSrc into strDest.

- char ∗ Strdup (const char ∗strSrc)

    Duplicate strings.

- char ∗ Itoa (int val, char ∗buf, int radix=10)

    Integer to Ascii conversion.

- char ∗ Itoa (unsigned long val, char ∗buf, int radix=10)

    Integer to Ascii conversion.

- char ∗ Itoa (MAPSInt64 val, char ∗buf, int radix=10)

    Integer to Ascii conversion.

- bool IsSpace (char c)

    Is the character c a space, a tab or a carriage return?

- bool IsDigit (char c)

    Is the character c a digit?

- void ∗ Memcpy (void ∗dest, const void ∗source, MAPSInt64 size, MAPSEvent ∗event=NULL)

    Memory copy.

- void ∗ Memset (void ∗dest, int c, MAPSInt64 size)

    Sets buffers to a specified character c, repeated size times.

- void ∗ Memmove (void ∗dest, const void ∗source, MAPSInt64 size)

    Moves one buffer to another.

- double Modf (double x, double ∗intptr)

    Splits a floating-point value into fractional and integer parts.

- double Fabs (double x)

    Calculates the absolute value of the floating-point argument.

RTMaps Main functions

- void Exit ()

    Required after any use of RTMaps...

- bool Run ()

    Starts the execution of the current session.

- bool Shutdown ()

    Shutdowns the RTMaps session currently running.

- bool Reset ()

  Resets the RTMaps system.

- bool Parse (const char ∗s)

  Parses a string containing MAPSScript instructions.

- bool ParseFile (const char ∗s)

  Parses a file containing MAPSScript instructions.

- void LoadCoreFunction (const char ∗cf)

  Loads a core function and activates it.

- MAPSCoreFunctionInterface ∗ CoreFunction (const char ∗cf)

  Returns a pointer to the core function named `cf` if it was previously loaded (returns NULL otherwise)

- void RegisterPackage (const char ∗fileName)

  Loads a package (set of components compiled as a shared object).

- void UnregisterPackage (const char ∗fileName)

  Unloads a package (set of components compiled as a shared object).

- bool IsRunning ()

  Is there a RTMaps system currently running?

- bool IsPaused ()

  Is the RTMaps clock in paused state ?

- bool IsReplaying ()

  Returns true if a file is open for replay.

- int CheckLevel ()

  Returns the level of structure control in the current system (0=no control, fastest, 2=max control, slowest)

RTMaps functions for distribution and synchronization

- bool IsDistributedAsMaster ()

  Is RTMaps acting as a Master on a network of distributed RTMaps.

- bool IsDistributedAsSlave ()

  Is RTMaps acting as a Slave on a network of distributed RTMaps.

- bool GetSynchronizer (void ∗owner, MAPSSynchronizer ∗∗ppSynchronizer)

Requests a pointer on the synchronizer object.

- bool ReleaseSynchronizer (void ∗module, MAPSSynchronizer ∗∗ppSynchronizer)

  Releases a pointer on the synchronizer object.

RTMaps Virtual Time management functions

- MAPSAbsoluteTime & TimeReference ()

  Gets the time reference (the absolute time the timestamps always refer to)
- MAPSTimestamp CurrentTime (bool lock=true, bool release=true)

  Gets the current time in the RTMaps system (virtual time)
- void SetCurrentTime (MAPSTimestamp t)

  Sets the current time (jumps in time!)
- void SetTimeSpeed (int speed)

  Sets the current time speed (1000 = real time)
- int TimeSpeed ()

  Gets the current time speed (1000 = real time)
- int TimeState ()

  Gets the current time state (MAPS::Running or MAPS::Paused)

RTMaps Index management

- void SetIndex ()

  Adds an index now (during recording)
- void Go2Index (int num)

  Goes to index num during replay.
- void Go2PreviousIndex ()

  Goes to previous index (during replay)
- void Go2NextIndex ()

  Goes to next index (during replay)

RTMaps Recording management

- void StartRecording (void)

  Starts recording information (starts all)

- void StopRecording (void)

  Stops recording information (stops all)

RTMaps Type Filtering management

- bool TypeFilter (const MAPSTypeInfo &outputType, MAPSTypeFilterBase &filter)

RTMaps General purpose functions

- ::IplImage IplImageModel (int width, int height, unsigned int channelSeq=MAPS_-CHANNELSEQ_BGR, unsigned int dataOrder=IPL_DATA_ORDER_PIXEL, unsigned int depth=IPL_DEPTH_8U)

  Creates a model of an operational IplImage structure with the provided parameters.

- ::IplImage IplImageModel (int width, int height, const char ∗channelSeq, unsigned int dataOrder=IPL_DATA_ORDER_PIXEL, unsigned int depth=IPL_DEPTH_8U)

  Creates a model of an operational IplImage structure with the provided parameters.

- bool IplImageCheck (::IplImage &image,::IplImage &model)

  Checks that an image is the same type and size as the model given in second parameter.

RTMaps Maintenance functions

- MAPSString About (void)

RTMaps Reporting functions

For all these functions, the importance must be set between 0 (not important) to 2 (of the utmost importance)

- void ReportInfo (const char ∗text, int importance=0)

  Reports a piece of information.

- void ReportWarning (const char ∗text, int importance=0)

  Reports a warning.

- void ReportError (const char ∗text, int importance=0)

  Reports an error.

- void Report (const char ∗text, int type, int importance=0)

  Reports something (with user feedback + logging)

- void MessageBox (const char ∗message, int type, int importance=0)

  Displays a modal MessageBox. To be used for debugging only since this function blocks until the user.

Time conversion and handling functions

- MAPSTimestamp TimestampFromString (const char ∗s, char ∗∗endptr=NULL)

  Transforms a string of form hh:mm:ss.mmmuuu into a timestamp.

- MAPSString TimeString (MAPSTimestamp t)
- MAPSString Timestamp2String (MAPSTimestamp t)

  Transforms a timestamp into a human readable string of form hh:mm:ss.mmmmmm.

- void Timestamp2AbsoluteTimeUTC (MAPSTimestamp t, MAPSAbsoluteTime ∗utctime)

  Transforms a timestamp in microseconds into an absolute time.

- MAPSTimestamp AbsoluteTimeUTC2Timestamp (const MAPSAbsoluteTime ∗utctime)

  Transforms an absolute time into a timestamp.

- MAPSInt64 AbsoluteTime2Integer (MAPSAbsoluteTime &time)
- void Integer2AbsoluteTime (MAPSInt64 integer, MAPSAbsoluteTime &time)
- MAPSString AbsoluteTime2String (MAPSAbsoluteTime absTime)

  Transforms an absolute time into a string of form yyyy/mm/dd hh:mm:ss.mmmuuu.

- void GetAbsoluteTime (MAPSAbsoluteTime ∗localtime)

  Gets the absolute current time (local time). Deprecated. Prefer using MAPS::GetAbsoluteTimeLocal.

- void GetAbsoluteTimeLocal (MAPSAbsoluteTime ∗localtime)

  Gets the absolute current local time.

- void GetAbsoluteTimeUTC (MAPSAbsoluteTime ∗utctime)

  Gets the absolute current time (UTC)

- bool AbsoluteTimeUTCToAbsoluteTimeLocal (const MAPSAbsoluteTime ∗utctime, MAPSAbsoluteTime ∗localtime)

  Converts a UTC time to a local time. Returns true in case of success, false otherwise.

- bool AbsoluteTimeLocalToAbsoluteTimeUTC (const MAPSAbsoluteTime ∗localtime, MAPSAbsoluteTime ∗utctime)

  Converts a local time to a UTC time. Returns true in case of success, false otherwise.

- bool SetSystemTime (const MAPSAbsoluteTime ∗utctime)

  Sets the computer system clock to the provided absolute time.

RTMaps Flow monitoring functions

- MAPSString GetWriteStatistics ()

  Gets the detailed statistics about the data flow to the hard disks (write file operations).

- MAPSString GetReadStatistics ()

  Gets the detailed statistics about the read file operations.

- int GetRemainingTime ()

  Gets the overall remaining recording time, if the flows stays constant.

- MAPSInt64 GetWriteFlow ()

  Gets the total write flow (file write operations)

- MAPSInt64 GetReadFlow ()

  Gets the total read flow (file read operations)

- void RecordMemoryWriteFlow (MAPSInt64 size)

  Notifies RTMaps that such a memory write flow has occured. Do not use in conjonction with MAPS::Memcpy(...)

- void RecordMemoryReadFlow (MAPSInt64 size)

  Notifies RTMaps that such a memory read flow has occured. Do not use in conjonction with MAPS::Memcpy(...)

- MAPSInt64 GetMemoryReadFlow ()

  Gets the total memory read flow.

- MAPSInt64 GetMemoryWriteFlow ()

  Gets the total memory write flow.

- MAPSInt64 GetMemoryFlow ()

  Gets the total memory flow (read+write)

- MAPSInt64 GetDiskFreeSpace (const char ∗path)

  Gets the free disk space on the disk containing `path`*.*

RTMaps OS wrapping functions

These functions give access to all the needed features of an OS except file I/O. These functions should be called instead of their OS-specific counterparts to ensure a cross-platform programming.

- MAPSTimestamp GetSystemAccurateTiming (void)

  Gets an immediate timestamp (in microseconds).

- bool CreateThread (void ∗(∗startAdress)(void ∗), void ∗argList)

  Starts a thread.

- void SetCurrentThreadPriority (int priority)

  Sets the priority of the current thread (between 0 and 255)

- MAPSThreadId GetCurrentThreadId ()

  Gets an id for the current thread.

- void ReleaseCurrentThread (void)

  Releases the current thread.

- void Sleep (MAPSDelay delay)

  Sleeps for a certain amount of time.

- void Wait4AWhile (void)

  Waits for a while. Useful little function.

- MAPSString GetCurrentDirectory ()

  Gets the current directory path.

- bool SetCurrentDirectory (const MAPSString path)

  Sets the current directory path.

- bool CreateDirectory (const char ∗dirName)

  Creates a directory.

- MAPSString GetTempDirectory ()

  Gets a path to the temporary directory.

- MAPSString GetInstallPath (const char ∗pathName)

  Gets a path refering to RTMaps installation.

- void ∗ AllocSharedMemory (int size)

  Memory allocation. Assumes it is allocated in a way that it can be accessed by a RTMaps system running in another process.

- void DeallocSharedMemory (void ∗ptr)

Memory deallocation. Assumes it was allocated in a way that it can be accessed by a RTMaps system running in another process.

RTMaps C++ interface functions

These functions are designed to control the RTMaps engine directly through C++ calls. These are rather low-level functions that are not of any interest for a component developer.

- bool GetBoolProperty (const char ∗s, bool ∗ok=NULL)

  Gets a boolean property.

- MAPSInt64 GetIntegerProperty (const char ∗s, bool ∗ok=NULL)

  Gets an integer property or enum property selected index.

- MAPSString GetStringProperty (const char ∗s, bool ∗ok=NULL)

  Gets a string property or enum property selected string.

- MAPSFloat GetFloatProperty (const char ∗s, bool ∗ok=NULL)

  Gets a float property.

- MAPSEnumStruct GetEnumsProperty (const char ∗s, bool ∗ok=NULL)

  Gets an enum property.

- MAPSComponent ∗ Component (const char ∗s)

  Returns a pointer to component s *(returns NULL if it does not exist)*

- MAPSProperty ∗ Property (const char ∗s)

  Returns a pointer to the property named s *(returns NULL if it does not exist)*

- MAPSComponent ∗ CreateComponent (const char ∗modelName, const char ∗componentName, bool start=true)

  Component instantiation.

- void StartComponent (const char ∗componentName)

  Starts the component if frozen.

- void KillComponent (const char ∗componentName)

  Dynamically destroys a component.

- void KillComponent (MAPSComponent &component)

  Dynamically destroys a component.

- void RenameComponent (const char ∗componentName, const char ∗newName)

Renames a component.

- void Attach (MAPSOutput &output, MAPSInput &input)

  Dynamically attaches an input to an output.

- void Attach (const char ∗outputName, const char ∗inputName)

  Dynamically attaches an input to an output.

- bool Attach2 (const char ∗name, const char ∗inputName)

  Dynamically attaches an input to an output. Extended version.

- void Detach (MAPSOutput &output, MAPSInput &input)

  Dynamically detaches an output from an input.

- void Record (const char ∗outputName, const char ∗recorder=NULL, const char ∗method=NULL, bool neverskipping=false, bool useTimestamp=false)

  Records an output.

- void Open (const char ∗pattern, const char ∗nspace=NULL, MAPSInt64 offset=0, MAPSTimestamp beginning=0, MAPSTimestamp end=0)

  Opens a database to replay records from.

- void Replay (const char ∗outputname)

  Replays some data.

- void Copy (const char ∗outputname, const char ∗recorderName)

  Copies some data.

- void StopCopy (const char ∗outputname)

  Aborts the copy of an output.

- void SetTimeAdapt (int ta)

  Sets the time automatic adaptation algorithms.

- MAPSTimestamp GetFirstTimestamp ()

  Returns the first timestamp of all the currently open databases.

- MAPSTimestamp GetLastTimestamp ()

  Returns the last timestamp of all the currently open databases.

- MAPSEvent ∗ GetShutdownEvent ()

  Gets a pointer to the shutdown event.

9.1.1 Detailed Description

The main RTMaps namespace. The MAPS namespace contains a lot of useful functions to deal with the engine features. All its members are declared as static.

9.1.2 Function Documentation

9.1.2.1 MAPSString MAPS::About (void)

The returned string contains information on the RTMaps version and the list of all the registered modules with their respective versions.

9.1.2.2 MAPSInt64 MAPS::AbsoluteTime2Integer (MAPSAbsoluteTime & *time*)

Deprecated

Transforms an absolute time to an integer, for absolute time comparison

Warning

Windows specific : The implementation of this function is not correct in RTMaps 3.x under Windows it will be corrected in RTMaps 4. In the meanwhile, prefer MAPS::AbsoluteTimeUTC2Timestamp.
Linux specific : The implementation of this function on Linux interprets the time as a local time and not UTC time. Prefer MAPS::AbsoluteTimeUTC2Timestamp.

9.1.2.3 MAPSTimestamp MAPS::AbsoluteTimeUTC2Timestamp (const MAPSAbsoluteTime ∗ *utctime*)

Transforms an absolute time into a timestamp.

Parameters

utctime	Absolute time structure interpreted as UTC time to convert to a timestamp.

Returns

Returns a timestamp in microseconds since the origin of UTC (1970, Jan 1rst, 0h)

9.1.2.4 bool MAPS::Attach2 (const char ∗ *name,* const char ∗ *inputName*)

Dynamically attaches an input to an output. Extended version.

This new version of attach allows the `name` parameter to be the name of an output OR the `name` parameter of the type of an output. This allows dynamic connection using a plain name, not a technical output name.

This function is designed to be used in components that automatically attach to some outputs.

9.1.2.5 MAPSString MAPS::GetInstallPath (const char ∗ *pathName*)

Gets a path refering to RTMaps installation.

Parameters

pathName	Specifies the path to retrieve. Can be "mapspath", "jrepath", "classpath" for java classes, "docpath", "xmlpath".

9.1.2.6 bool MAPS::GetSynchronizer (void ∗ *owner*, MAPSSynchronizer ∗∗ *ppSynchronizer*)

Requests a pointer on the synchronizer object.

When the module no more needs the synchronizer ability, it has to call MAPS::ReleaseSynchronizer

Parameters

owner	Pointer to the MAPSModule requesting the synchronizer ability
ppSynchronizer	Pointer to the place where the synchronizer object pointer will be stored if the function succeeds. If the function fails, it is set to `NULL`.

Returns

`true` if the function succeeds, `false` if the function fails (i.e. the synchronizer object is already held by someone else).

9.1.2.7 MAPSTimestamp MAPS::GetSystemAccurateTiming (void)

Gets an immediate timestamp (in microseconds).

Note that you must generally use MAPS::CurrentTime() in your own components, since the GetSystemAccurateTiming function returns a timestamp in real time, not in the RTMaps virtual time. Furthermore, the timestamp returned by this function has no reference. It is the RTMaps engine that uses this OS wrapping function to get some accurate timings and translates them to a referenced and scaled virtual time.

9.1.2.8 void MAPS::Integer2AbsoluteTime (MAPSInt64 *integer*, MAPSAbsoluteTime & *time*)

Deprecated

Transforms an integer to an absolute time

9.1.2.9 ::IplImage MAPS::IplImageModel (int *width*, int *height*, unsigned int *channelSeq* = `MAPS_CHANNELSEQ_BGR`, unsigned int *dataOrder* = `IPL_DATA_ORDER_PIXEL`, unsigned int *depth* = `IPL_DEPTH_8U`)

Creates a model of an operational IplImage structure with the provided parameters.

The generated model can be thus used with MAPSOutput::AllocOutputBufferIplImage

See also

MAPSOutput::AllocOutputBufferIplImage

9.1.2.10 ::IplImage MAPS::IplImageModel (int *width*, int *height*, const char * *channelSeq*, unsigned int *dataOrder* = `IPL_DATA_ORDER_PIXEL`, unsigned int *depth* = `IPL_DEPTH_8U`)

Creates a model of an operational IplImage structure with the provided parameters.

The generated model can be thus used with MAPSOutput::AllocOutputBufferIplImage

See also

MAPSOutput::AllocOutputBufferIplImage

9.1.2.11 void* MAPS::Memcpy (void * *dest*, const void * *source*, MAPSInt64 *size*, MAPSEvent * *event* = `NULL`)

Memory copy.

Should be ALWAYS called instead of memcpy

9.1.2.12 void MAPS::MessageBox (const char * *message*, int *type*, int *importance* = `0`)

Displays a modal MessageBox. To be used for debugging only since this function blocks until the user.

Parameters

message	The message to be displayed in the message box
type	The type can be MAPS::Info, MAPS::Warning, or MAPS::Error
importance	Unused

9.1.2.13 void MAPS::ReleaseCurrentThread (void)

Releases the current thread.

Gives up its remaining time slice. This may optimize the behaviour of a component and it is used in the RTMaps engine to optimize threads switching.

9.1.2.14 void MAPS::Report (const char ∗ *text,* int *type,* int *importance* = `0`)

Reports something (with user feedback + logging)

Parameters

type	The report type. Can be MAPS::Info, MAPS::Warning or MAPS::Error.
text	The message to report to the user
importance	Unused

9.1.2.15 void MAPS::ReportInfo (const char ∗ *text,* int *importance* = `0`)

Reports a piece of information.

Only for use by programs out of the RTMaps system. For RTMaps components, please use the MAPSComponent::Report function.

See also

MAPSComponent::Report

9.1.2.16 bool MAPS::SetSystemTime (const MAPSAbsoluteTime ∗ *utctime*)

Sets the computer system clock to the provided absolute time.

Parameters

utctime	Absolute time in UTC to apply to the current system clock.

Returns

true in case of success, false otherwise.

Remarks

The function may fail if the caller does not have administrator or super-user rights.

9.1.2.17 void MAPS::Sleep (MAPSDelay *delay*)

Sleeps for a certain amount of time.

The current thread is released for a delay of `delay` microseconds. Note that it is not considered to be an accurate function. For instance, in the Windows implementation of RTMaps, the delay precision is about 40ms.

Note that you must generally use MAPSComponent::Rest() or MAPSComponent::Wait() in your own components, since the MAPS::Sleep function works in real time, not in the RTMaps virtual time. Furthermore, MAPSComponent::Rest() and MAPSComponent::Wait() are much more accurate.

9.1.2.18 void MAPS::Timestamp2AbsoluteTimeUTC (MAPSTimestamp *t*, MAPSAbsoluteTime ∗ *utctime*)

Transforms a timestamp in microseconds into an absolute time.

Parameters

t	Timestamp in microseconds, interpreted as the number of microseconds since the origin of UTC (1970, Jan 1rst, 0h).
utctime	Pointer to a structure of type MAPSAbsoluteTime to be filled in with UTC time (not local time).

9.1.2.19 MAPSString MAPS::Timestamp2String (MAPSTimestamp *t*)

Transforms a timestamp into a human readable string of form hh:mm:ss.mmmmmm.

Parameters

t	A timestamp in microseconds.

Warning

This method does not support absolute times so it will never generate strings of form yyyy/mm/dd hh:mm:ss.mmmuuu.

9.1.2.20 MAPSTimestamp MAPS::TimestampFromString (const char ∗ *s*, char ∗∗ *endptr* = `NULL`)

Transforms a string of form hh:mm:ss.mmmuuu into a timestamp.

Warning: it does not support absolute time strings of form yyyy/mm/dd hh:mm:ss.mmmmmm. Only relative time strings.

9.1.2.21 MAPSString MAPS::TimeString (MAPSTimestamp *t*)

See also

Equivalent to MAPS::Timestamp2String.

9.1.2.22 bool MAPS::TypeFilter (const MAPSTypeInfo & *outputType*, MAPSTypeFilterBase & *filter*)

Passes a type through a filter type

Parameters

outputType	type to pass through the filter, generally the type of an output
filter	filter to use

Returns

returns true when *filter* filters out *outputType* For RTMaps type filtering behaviour, see the documentation of MAPSTypeFilterBase.

See also

MAPSTypeInfo MAPSTypeFilterBase

9.1.3 Variable Documentation

9.1.3.1 const int MAPS::EnumProperty

See also

MAPSEnumStruct

9.1.3.2 const MAPSTypeInfoValue MAPS::TextAscii

Ascii characters (1 character = 8 bits).

Remark: When dealing with an output with type TextAscii, the arbitrary convention is: the text bytes are ended with the '\0' character in the MAPSIOElt buffer, but this '\0' character is not counted in the VectorSize() field of the MAPSIOElt. So, ioElt->VectorSize() should return the same result than std::strlen(ioElt->TextAscii()).

Chapter 10

Class Documentation

10.1 IplImage Struct Reference

The famous IplImage structure, today's standard structure for image processing.

Public Attributes

- MAPSInt32 nSize

  Size of iplImage struct.

- MAPSInt32 ID

  Image header version.

- MAPSInt32 nChannels

  Number of channels in the image (generally 1,3 or 4)

- MAPSInt32 alphaChannel

  Alpha channel number (0 if there is no alpha channel in the image)

- MAPSInt32 depth

  Bit depth of pixels.

- char colorModel [4]

  A four-character string describing the color model.

- char channelSeq [4]

  A four-character string describing the sequence of color channels.

- MAPSInt32 dataOrder

  Can be `IPL_DATA_ORDER_PIXEL` or `IPL_DATA_ORDER_PLANE`.

- MAPSInt32 origin

  The origin of the image: can be `IPL_ORIGIN_TL` (top left corner) or `IPL_ORIGIN_-BL` (bottom left corner)

- MAPSInt32 align

  Alignment of image data: can be `IPL_ALIGN_DWORD` (4-byte align) or `IPL_-ALIGN_QWORD` (8-byte align)

- MAPSInt32 width

  Width of the image in pixels.

- MAPSInt32 height

  Height of the image in pixels.

- IplROI ∗ roi

  Pointer to a ROI (Region of interest) structure.

- struct _IplImage ∗ maskROI

  Pointer to the header of another image that specifies the mask ROI.

- void ∗ imageId

  The image ID.

- struct _IplTileInfo ∗ tileInfo

  The pointer to the IplTileInfo structure containing information used for image tiling.

- MAPSInt32 imageSize

  Useful size in bytes.

- char ∗ imageData

  Pointer to aligned image.

- MAPSInt32 widthStep

  The size of aligned line in bytes.

- MAPSInt32 BorderMode [4]

  The top, bottom, left and right border mode.

- MAPSInt32 BorderConst [4]

  Constants for the top, bottom, left and right border.

- char ∗ imageDataOrigin

  Pointer to full, nonaligned image.

10.1.1 Detailed Description

The famous IplImage structure, today's standard structure for image processing. The IplImage structure is the preferred structure for image processing with RTMaps.

The IplImage structure is rather complex but very complete. Easy handling of this structure by RTMaps is provided by MAPS::IplImageModel and MAPSOutput::AllocOutputBufferIplImage.

10.1.2 Member Data Documentation

10.1.2.1 char IplImage::channelSeq[4]

A four-character string describing the sequence of color channels.

Use `MAPS_CHANNELSEQ_xxx` to fill this field.

10.1.2.2 char IplImage::colorModel[4]

A four-character string describing the color model.

Use `MAPS_COLORMODEL_xxx` to fill this field

10.1.2.3 MAPSInt32 IplImage::depth

Bit depth of pixels.

Can be one of:

- `IPL_DEPTH_1U`
- `IPL_DEPTH_8U`
- `IPL_DEPTH_8S`
- `IPL_DEPTH_16U`
- `IPL_DEPTH_16S`
- `IPL_DEPTH_32S`
- `IPL_DEPTH_32F`.

10.1.2.4 void∗ IplImage::imageId

The image ID.

Field reserved for the use of the application to identify the image.

10.1.2.5 struct _IplImage∗ IplImage::maskROI

Pointer to the header of another image that specifies the mask ROI.

This argument can be `NULL`, which indicates that no mask ROI is used. A pixel is processed if the corresponding mask pixel is 1, and is not processed if the mask pixel is 0. The *maskROI* field of the mask image's header is ignored.

10.1.2.6 IplROI∗ IplImage::roi

Pointer to a ROI (Region of interest) structure.

This argument can be `NULL`, which implies that a region of interest comprises all channels and the entire image area.

The documentation for this struct was generated from the following file:

- maps.hpp

10.2 IplROI Struct Reference

The IPL Region Of Interest structure.

Public Attributes

- MAPSInt32 coi

  The channel of interest number.

- MAPSInt32 xOffset

  The horizontal offset from the origin of the rectangular ROI.

- MAPSInt32 yOffset

  The vertical offset from the origin of the rectangular ROI.

- MAPSInt32 width

  The width of the rectangular ROI.

- MAPSInt32 height

  The height of the rectangular ROI.

10.2.1 Detailed Description

The IPL Region Of Interest structure.

10.2.2 Member Data Documentation

10.2.2.1 MAPSInt32 IplROI::coi

The channel of interest number.

This parameter indicates which channel in the original image will be affected by processing taking place in the region of interest; *coi* equal to 0 indicates that all channel will be affected

The documentation for this struct was generated from the following file:

- maps.hpp

10.3 MAPSAbsoluteTime Struct Reference

Absolute time structure.

Public Attributes

- MAPSUInt32 year

  19XX or 20XX

- MAPSUInt32 month

  Ranging from 1 to 12.

- MAPSUInt32 day

  Ranging from 1 to 31.

- MAPSUInt32 hour

  Ranging from 0 to 23.

- MAPSUInt32 minutes

  Ranging from 0 to 59.

- MAPSUInt32 seconds

  Ranging from 0 to 59.

- MAPSUInt32 milliseconds

  Ranging from 0 to 999.

- MAPSUInt32 microseconds

  Ranging from 0 to 999.

Related Functions

(Note that these are not member functions.)

- bool operator< (const MAPSAbsoluteTime &t0, const MAPSAbsoluteTime &t1)

  Standard comparison operator on absolute time.

- bool operator== (const MAPSAbsoluteTime &t0, const MAPSAbsoluteTime &t1)

  Standard comparison operator on absolute time.

- bool operator> (const MAPSAbsoluteTime &t0, const MAPSAbsoluteTime &t1)

  Standard comparison operator on absolute time.

10.3.1 Detailed Description

Absolute time structure. This is the RTMaps standard structure for absolute time handling

The documentation for this struct was generated from the following file:

- maps.hpp

10.4 MAPSAction Class Reference

The RTMaps Module Action class.

Public Member Functions

- virtual void DoIt ()=0

  Executes the action.

10.4.1 Detailed Description

The RTMaps Module Action class.

The documentation for this class was generated from the following file:

- maps.hpp

10.5 MAPSArray< T > Class Template Reference

The RTMaps array class.

Public Member Functions

- T & operator[] (int i) const

  Retrieves the element with index i in the array.

- T & operator() (int i)

  Retrieves the element with index i in the array and resizes the array when necessary.

- MAPSArray< T > & operator= (const MAPSArray< T > &v)

  Copy operator.

- int Dim () const

  Gets the current size of the array.

- int Size () const

  Gets the current size of the array.

- T ∗ Vect () const

  Returns a pointer to the data.

- void Fill (const T &model)

  Fills the array with model model.

- T & Append ()

  Appends an element to the array.

- void Set (T ∗ptr, int d)

  Sets the content of the array. The data is not copied.

- void Realloc (int d)

  Reallocates the data.

- void Alloc (int d)

  Allocates some data.

- void SetSize (int d)

  Sets the size of the array. Moves the data (realloc) if needed.

- void Clear ()

  Clears the array (i.e. Resets size to zero)

- operator T ∗ ()

  Cast operator. Returns a C-like array.

- T & Shift ()

  Shifts an element from the array.

- void Unshift (const T &z)

  Unshifts an element to the array.

- T & Pop ()

  Pops an element from the end of the array.

- void Push (const T &z)

  Pushes an element to the end of the array.

- MAPSArray ()

  Basic constructor.

- MAPSArray (int d)

  Allocating constructor.

- MAPSArray (const MAPSArray< T > &v)

  Copy constructor.

- virtual ~MAPSArray ()

  Destructor.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  MAPSArray< T > operator∗ (const MAPSMatrix2< T > &A, const MAPSArray< T > &b)

  Multiplication of a MAPSMatrix2 with a MAPSArray.

- template<typename T >
  MAPSArray< T > operator+ (const MAPSArray< T > &a, const MAPSArray< T > &b)

  MAPSArrays addition.

- template<typename T >
  MAPSArray< T > operator- (const MAPSArray< T > &a, const MAPSArray< T > &b)

  MAPSArrays substraction.

- template<typename T >
  MAPSArray< T > operator+ (const MAPSMatrix2< T > &a, const MAPSArray< T > &b)

  Addition of a MAPSMatrix2 with a MAPSArray.

- template<typename T >
  MAPSStreamedString & operator<< (MAPSStreamedString &out, const MAPSArray< T > &v)

  Standard operator.

10.5.1 Detailed Description

template<typename T> class MAPSArray< T >

The RTMaps array class. This class implements a resizable one-dimensional array. It can also work as a very efficient ring buffer, a queue or even a stack. Indeed, it acts as a Perl-like array, that is with an offset management.

Using this class is generally faster than using MAPSList, since MAPSArray allocates objects by packets, and not one by one. MAPSArray is the preferable class to use when you need to use a queue or a stack.

Note

The constructor and the destructor of class `T` are never called.Use MAPSList if you need a systematic call to content constructors/destructors.
The size of the allocated array is always of power of 2, due to an optimization in the ring buffer behaviour management.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 template<typename T> MAPSArray< T >::MAPSArray ( )

Basic constructor.

Allocates nothing. The size of the array is set to 0.

10.5.2.2 template<typename T> MAPSArray< T >::MAPSArray (int *d*) `[explicit]`

Allocating constructor.

Allocates an array of size *d*. Indeed, the true allocated size is a power of 2.

10.5.3 Member Function Documentation

10.5.3.1 template<typename T> void MAPSArray< T >::Alloc (int *d*)

Allocates some data.

Destroys previously allocated data.

10.5.3.2 template<typename T> T& MAPSArray< T >::Append ( )

Appends an element to the array.

Resizes the array and returns a reference to the new appended element.

10.5.3.3 template<typename T> MAPSArray< T >::operator T $*$ ( )

Cast operator. Returns a C-like array.

Returns a pointer to the data.

Should be used with care... Especially after the use of Shift() function, it will not give you what you expect, since the array will thus be used as a ring buffer. Furthermore, the pointer might not be valid for long due to possible reallocations.

10.5.3.4 template<typename T> T& MAPSArray< T >::operator[] (int *i*) const

Retrieves the element with index *i* in the array.

No memory allocation if *i* is superior to the actual size of the array.

10.5.3.5 template<typename T> T& MAPSArray< T >::Pop ( )

Pops an element from the end of the array.

This function returns a reference, but this reference is only valid up to the next call to another array related function, since the element can and will certainly be crushed by a further call to Push() of Unshift().

10.5.3.6 template<typename T > void MAPSArray< T >::Realloc (int *d*)

Reallocates the data.

Moves previous data to a new location with room for *d* pieces of data.

10.5.3.7 template<typename T> void MAPSArray< T >::Set (T $*$ *ptr,* int *d*)

Sets the content of the array. The data is not copied.

This is not a copy operator. The array will point to the T objects given in arguments. This data will not be destroyed when the object is destroyed (i.e. the array does not own the data).

10.5.3.8 template<typename T> T& MAPSArray< T >::Shift ( )

Shifts an element from the array.

Perl-like operator. Removes an element from the beginning of the array (efficiently, i.e. without moving all the elements to the left, but through the management of an "offset" variable) Note that further pointers returned by a Vect() function are not valid any more after a call to Shift().

This function returns a reference, but this reference is only valid up to the next call to another array related function, since the element can and will certainly be crushed by a further call to Push() of Unshift().

10.5.3.9 template<typename T> void MAPSArray< T >::Unshift (const T & *z*)

Unshifts an element to the array.

Perl-like operator. It adds an element in front of the array. Inserts an element at the beginning of the array (efficiently, i.e. without moving all the elements to the right, but through the management of an "offset" variable) Note that further pointers returned by a Vect() function are not valid any more after a call to Shift().

10.5.3.10 template<typename T> T* MAPSArray< T >::Vect ( ) const

Returns a pointer to the data.

Should be used with care... Especially after the use of Shift() function, it will not give you what you expect, since the array will thus be used as a ring buffer. Furthermore, the pointer might not be valid for long due to possible reallocations.

10.5.4 Friends And Related Function Documentation

10.5.4.1 template<typename T > MAPSStreamedString & operator<< (MAPSStreamedString & *out*, const MAPSArray< T > & *v*) `[related]`

Standard operator.

The documentation for this class was generated from the following file:

- maps.hpp

10.6 MAPSBaseClock Class Reference

MAPSBaseClock class.

Public Member Functions

- unsigned char GetClockID ()

  For internal use.

- const char ∗ GetClockName ()

  For internal use.

- virtual bool IsSynchronizable ()

  For internal use. Do not overload.

- virtual bool IsReplayable ()

  For internal use. Do not overload.

- virtual void InitClock ()=0

  Overloading this function is necessary.

- virtual void RunClock ()=0

  Overloading this function is necessary.

- virtual void ShutdownClock ()=0

  Overloading this function is necessary.

- virtual MAPSTimestamp CurrentTime ()=0
- int GetAbsoluteTimeSpeed ()

  This function returns the maximum "real" timespeed currently declared by this clock.

- void SetAbsoluteTimeSpeed (int absoluteTimeSpeed)

  Use this function to declare the maximum "real" timespeed your custom clock can reach.

- MAPSBaseClock (const char ∗name)

  Constructor.

10.6.1 Detailed Description

MAPSBaseClock class. Every object inheriting from this class has to implement a basic clock and this clock will be available for the RTMaps engine.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 MAPSBaseClock::MAPSBaseClock (const char ∗ *name*)

Constructor.

Parameters

name	The name of your custom clock. To be simple, just pass the name of your RTMaps component.

10.6.3 Member Function Documentation

10.6.3.1 virtual MAPSTimestamp MAPSBaseClock::CurrentTime ( ) `[pure virtual]`

This function is called each time someone in RTMaps (the VCR, the Player, or any other component) queries the current (through the MAPS::CurrentTime() method).

Returns

The current time of your custom clock, expressed in microseconds.

Warning

This function is called very often. It has to be the fastest possible.
This function can be called concurrently from several threads. Make sure it is thread safe.

10.6.3.2 int MAPSBaseClock::GetAbsoluteTimeSpeed ( )

This function returns the maximum "real" timespeed currently declared by this clock.

Returns

The maximum timespeed your custom clock can reach on the absolute time base (i.e. your watch, not the RTMaps VCR).

See also

SetAbsoluteTimeSpeed(int absoluteTimeSpeed);

10.6.3.3 virtual void MAPSBaseClock::InitClock ( ) `[pure virtual]`

Overloading this function is necessary.

This function is called when the object is chosen as the current clock in the RTMaps diagram. You have to implement this function, but implemenation can often remain empty...

10.6.3.4 virtual bool MAPSBaseClock::IsReplayable ( ) `[virtual]`

For internal use. Do not overload.

Returns

Always return `false`.

10.6.3.5 virtual bool MAPSBaseClock::IsSynchronizable ( ) `[virtual]`

For internal use. Do not overload.

Returns

Always return `false`.

10.6.3.6 virtual void MAPSBaseClock::RunClock ( ) `[pure virtual]`

Overloading this function is necessary.

This function is called when your custom clock has to start running.

10.6.3.7 void MAPSBaseClock::SetAbsoluteTimeSpeed (int *absoluteTimeSpeed*)

Use this function to declare the maximum "real" timespeed your custom clock can reach.

Parameters

absolute-TimeSpeed	The maximum "real" timespeed your custom clock can reach. "real" time-speed means "in comparison with real time", the time of your watch. Declaring a max. timespeed which is much higher than the effective timespeed of your clock will result in little more CPU consumption. On the other hand, declaring a max. timespeed which is lower thant the effective timespeed of your clock will result in timers inacurracy within RTMaps (the Wait and Rest functions will be affected). A value of 1000 means real-time, a value of 500 means half the real-time for example.

10.6.3.8 virtual void MAPSBaseClock::ShutdownClock ( ) `[pure virtual]`

Overloading this function is necessary.

This function is called when your custom clock has to stop running.

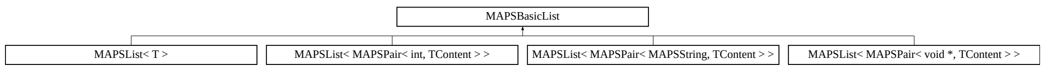
The documentation for this class was generated from the following file:

- maps.hpp

10.7 MAPSBasicList Class Reference

The base class for the MAPSList template class.

Inheritance diagram for MAPSBasicList:


MAPSBasicList
MAPSList< T >
MAPSList< MAPSPair< int, TContent > >
MAPSList< MAPSPair< MAPSString, TContent > >
MAPSList< MAPSPair< void *, TContent > >

Public Member Functions

- bool Empty () const

  Returns `true` *if the list is empty.*

- int Length () const

  Returns the length of the list (the number of elements in the list)

- int Size () const

  Returns the number of elements in the list.

- int Dim () const

  Returns the number of elements in the list.

- MAPSBasicList ()

  Default constructor.

Iteration management

- MAPSListIterator First () const

  Returns an iterator corresponding to the first element in the list (`NULL` *if the list is empty)*

- MAPSListIterator Iterator () const

  Returns an iterator corresponding to the first element in the list (`NULL` *if the list is empty)*

- MAPSListIterator Begin () const

  Returns an iterator corresponding to the first element in the list (`NULL` *if the list is empty)*

- MAPSListIterator Last () const

  Returns an iterator corresponding to the last element in the list (`NULL` *if the list is empty)*

- MAPSListIterator End () const

  Returns a `NULL` *iterator.*

- MAPSListIterator & Next (MAPSListIterator &it) const

  Increments iterator it*. Returns the next item in the list.*

- MAPSListIterator & Previous (MAPSListIterator &it) const

  Decrements iterator it. Returns the previous item in the list.

- bool Finished (MAPSListIterator it) const

  Has the iterator it reached the end of the list ?

- bool IsFirst (MAPSListIterator it) const

  Does the iterator it correspond to the first element in the list?

- bool IsLast (MAPSListIterator it) const

  Does the iterator it correspond to the last element in the list?

10.7.1 Detailed Description

The base class for the MAPSList template class. Should not be used directly.

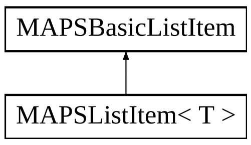
The documentation for this class was generated from the following file:

- maps.hpp

10.8 MAPSBasicListItem Class Reference

The basic list item class.

Inheritance diagram for MAPSBasicListItem:



Friends

- class MAPSListIterator

10.8.1 Detailed Description

The basic list item class. This is the base element of a MAPSBasicList.

Warning

Should not be used directly by the end user.

The documentation for this class was generated from the following file:

- maps.hpp

10.9 MAPSCANFrame Struct Reference

CAN Frames structure.

Public Attributes

- MAPSUInt32 arbitrationId

  The arbitration field for the frame (11 or 29 bits)

- bool isRemote

  Is this a remote frame ?

- MAPSInt8 dataLength

  The number of bytes in the frame, ranging from 1 to 8.

- MAPSUInt8 data [8]

  The n bytes of data (1<n<8)

Static Public Attributes

- static const MAPSUInt32 ExtendedId

  The constant to "or" with arbitrationId if EXTENDED identifier is to be used.

- static const MAPSUInt32 StandardId

  A 0 constant for STANDARD identifiers.

- static const MAPSUInt32 AddressMask

  The mask to use in order to get the id from arbitrationId member.

10.9.1 Detailed Description

CAN Frames structure. This is the RTMaps standard CAN frame type. It contains all the necessary members to describe completely a frame on a CAN bus

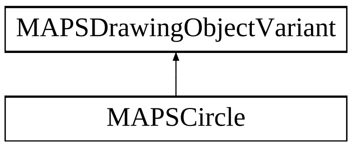
The documentation for this struct was generated from the following file:

- maps.hpp

10.10 MAPSCircle Struct Reference

The MAPSCircle structure.

Inheritance diagram for MAPSCircle:



Public Attributes

- int x

  The x-center of the circle.

- int y

  The y-center of the circle.

- int radius

  The radius of the circle.

10.10.1 Detailed Description

The MAPSCircle structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.11 MAPSComplex Struct Reference

Complex number structure.

Public Attributes

- MAPSFloat r

  Real part.

- MAPSFloat i

  Imaginary part.

10.11.1 Detailed Description

Complex number structure. This is the RTMaps standard structure for complex numbers handling

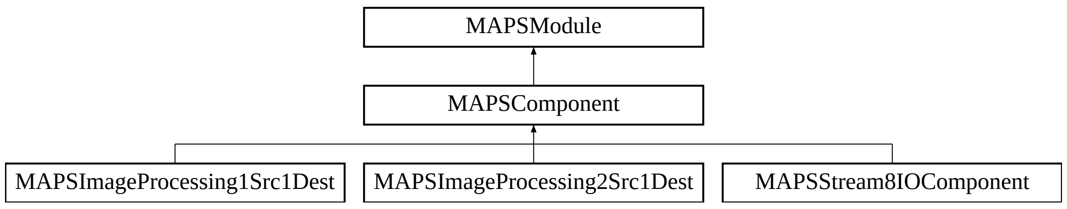
The documentation for this struct was generated from the following file:

- maps.hpp

10.12 MAPSComponent Class Reference

The base class for all RTMaps components.

Inheritance diagram for MAPSComponent:



Public Member Functions

- virtual void Set (MAPSProperty &p, bool value)

  Sets boolean value from boolean property p.

- virtual void Set (MAPSProperty &p, MAPSInt64 value)

  Sets integer value from integer property p.

- virtual void Set (MAPSProperty &p, MAPSFloat value)

  Sets float value from float property p.

- virtual void Set (MAPSProperty &p, const MAPSString &value)

  Sets string value from string property p.

- virtual void Set (MAPSProperty &p, const MAPSEnumStruct &enumStruct)

  Sets enum struct enumStruct from enum property p.

- void Start ()

  Starts the component.

- void SetAutomaticStart (bool s)

  Sets automatic start mode.

Protected Member Functions

- MAPSInput & NewInput (int model, const char ∗name=NULL)

  Dynamically creates an input.

- MAPSInput & NewInput (const char ∗model, const char ∗name=NULL)

  Dynamically creates an input.

- MAPSOutput & NewOutput (int model, const char ∗name=NULL)

  Dynamically creates an output.

- MAPSOutput & NewOutput (const char ∗model, const char ∗name=NULL)

  Dynamically creates an output.

- MAPSProperty & NewProperty (int model, const char ∗name=NULL)

  Dynamically creates a property.

- MAPSProperty & NewProperty (const char ∗model, const char ∗name=NULL)

  Dynamically creates a property.

- MAPSAction & NewAction (int model, const char ∗name=NULL)

  Dynamically creates an action.

- MAPSAction & NewAction (const char ∗model, const char ∗name=NULL)

  Dynamically creates an action.

- void CreateThread (MAPSThreadFunction)

  Creates a new thread and associates it to the component.

- virtual void FreeBuffers ()

  Frees the output buffers.

- void CommitSuicide (void)

  Must be called by the component itself.

- void CallDynamic ()

  Dynamic behaviour support.

- void DynamicConfirm ()

  Confirms all previously created inputs, outputs, actions and properties.

User provided functions

- virtual void Dynamic (void)

  User provided function for dynamic input/output/property generation.

- virtual void Core (void)=0

  The component main function. Its brain.

- virtual void Birth (void)

  The Birth function.

- virtual void Banzai (void)

  The Banzai function.

- virtual void Death (void)

  The Death function.

10.12.1 Detailed Description

The base class for all RTMaps components. Some useful macros are available for easier development of components. They are detailed in Component design.

10.12.2 Member Function Documentation

10.12.2.1 virtual void MAPSComponent::Banzai (void) `[protected, virtual]`

The Banzai function.

It is called as soon as someone asks the component to die. Must be defined only in very rare cases (generally when we need to kill a blocked on hardware related thread). For experts only.

10.12.2.2 virtual void MAPSComponent::Birth (void) `[protected, virtual]`

The Birth function.

Called only once, just before the first call to Core, when the component starts. Note that the birth function is executed in the very same thread as the Core function.

10.12.2.3 void MAPSComponent::CallDynamic ( ) `[protected]`

Dynamic behaviour support.

Can be called so that inputs or outputs can be dynamically changed according to a user-supplied Dynamic() function

10.12.2.4 virtual void MAPSComponent::Core (void) [protected, pure virtual]

The component main function. Its brain.

Must be overloaded. Obviously.

In threaded mode, the code in Core() is executed in the main thread of the component, the one created automatically. In sequential mode, the code in Core() is executed each time data arrives on one of the component inputs.

10.12.2.5 void MAPSComponent::CreateThread (MAPSThreadFunction) [protected]

Creates a new thread and associates it to the component.

Takes the function to execute in parameter. This function takes itself no parameter, but is a function of an object, so the `this` pointer to this object is given as a hidden parameter.
MAPSThreadFunction is defined as :

```
typedef void (MAPSModule::*MAPSThreadFunction)(void);
```

To ensure portability, if the thread function is a member of MAPSMyComponentClass, it must be called like this:

```
CreateThread((MAPSThreadFunction)&MAPSMyComponentClass::memberFunction);
```

Reimplemented from MAPSModule.

10.12.2.6 virtual void MAPSComponent::Death (void) [protected, virtual]

The Death function.

Called when the component dies, after the call the Banzai() and the death of all threads. The Death function is executed in the very same thread as the Core function.

10.12.2.7 virtual void MAPSComponent::Dynamic (void) [protected, virtual]

User provided function for dynamic input/output/property generation.

If your component dynamically creates inputs, outputs or properties through some calls to MAPSComponent::NewInput, ..., these calls must be made in a Dynamic() overloaded function, so that the RTMaps engine can update its view of the component. By default, confirm any previous inputs, outputs, properties and actions creation.

10.12.2.8 void MAPSComponent::DynamicConfirm ( ) [protected]

Confirms all previously created inputs, outputs, actions and properties.

Can be called in Dynamic(), if no change is detected.

10.12.2.9 virtual void MAPSComponent::FreeBuffers ( ) `[protected, virtual]`

Frees the output buffers.

Note that this can be overloaded for very specific buffer allocations, like allocation of frames in DMA memory (by frame grabbers).

10.12.2.10 MAPSAction& MAPSComponent::NewAction (int *model*, const char ∗ *name* = `NULL`) `[protected]`

Dynamically creates an action.

Parameters

model	model number (its order in the MAPS_ACTIONS_DEFINITION description)
name	the name for the new action. If `NULL`, the model name is used.

Returns

A reference to the newly created MAPSAction object.

10.12.2.11 MAPSAction& MAPSComponent::NewAction (const char ∗ *model*, const char ∗ *name* = `NULL`) `[protected]`

Dynamically creates an action.

Parameters

model	model name (its name in the MAPS_ACTIONS_DEFINITION description)
name	the name for the new action. If `NULL`, the model name is used.

Returns

A reference to the newly created MAPSAction object.

10.12.2.12 MAPSInput& MAPSComponent::NewInput (const char ∗ *model*, const char ∗ *name* = `NULL`) `[protected]`

Dynamically creates an input.

Parameters

model	model name (its name in the MAPS_INPUTS_DEFINITION description)
name	the name for the new input. If `NULL`, the model name is used.

Returns

A reference to the newly created MAPSInput object.

10.12.2.13 MAPSInput& MAPSComponent::NewInput (int *model*, const char ∗ *name* = NULL) [protected]

Dynamically creates an input.

Parameters

model	model number (its order in the MAPS_INPUTS_DEFINITION description)
name	the name for the new input. If NULL, the model name is used.

Returns

A reference to the newly created MAPSInput object.

10.12.2.14 MAPSOutput& MAPSComponent::NewOutput (int *model*, const char ∗ *name* = NULL) [protected]

Dynamically creates an output.

Parameters

model	model number (its order in the MAPS_OUTPUTS_DEFINITION description)
name	the name for the new output. If NULL, the model name is used.

Returns

A reference to the newly created MAPSOutput object.

10.12.2.15 MAPSOutput& MAPSComponent::NewOutput (const char ∗ *model*, const char ∗ *name* = NULL) [protected]

Dynamically creates an output.

Parameters

model	model name (its name in the MAPS_OUTPUTS_DEFINITION description)
name	the name for the new output. If NULL, the model name is used.

Returns

A reference to the newly created MAPSOutput object.

10.12.2.16 MAPSProperty& MAPSComponent::NewProperty (const char ∗ *model*, const char ∗ *name* = NULL) [protected]

Dynamically creates a property.

Parameters

model	model name (its name in the MAPS_PROPERTIES_DEFINITION description)
name	the name for the new property. If `NULL`, the model name is used.

Returns

A reference to the newly created MAPSProperty object.

10.12.2.17 MAPSProperty& MAPSComponent::NewProperty (int *model*, const char ∗ *name* = NULL) [protected]

Dynamically creates a property.

Parameters

model	model number (its order in the MAPS_PROPERTIES_DEFINITION description)
name	the name for the new property. If `NULL`, the model name is used.

Returns

A reference to the newly created MAPSProperty object.

10.12.2.18 void MAPSComponent::SetAutomaticStart (bool *s*)

Sets automatic start mode.

Usually, a component starts automatically on general run, or when the system is already running on creation. Sometimes, this is not the case, One wants to start the component later.

See also

MAPS::CreateComponent

10.12.2.19 void MAPSComponent::Start ( )

Starts the component.

A component is started automatically if RTMaps is running, except when start is set to `false` in the call to MAPS::CreateComponent. If so, a call to Start is necessary to effectively start the component.

Reimplemented from MAPSModule.

The documentation for this class was generated from the following file:

- maps.hpp

10.13 MAPSCouple< T > Class Template Reference

Template class for couple data.

Public Member Functions

- T & first ()

  Returns a reference to the first element.

- T & second ()

  Returns a reference to the second element.

- T & operator[] (int i)

  Unsafe! Use it at your own risk.

Related Functions

(Note that these are not member functions.)

- template< typename T >
  MAPSStreamedString & operator<< (MAPSStreamedString &out, MAPSCouple< T > &c)

  Standard operator.

10.13.1 Detailed Description

template<typename T> class MAPSCouple< T >

Template class for couple data.

10.13.2 Friends And Related Function Documentation

10.13.2.1 template< typename T > MAPSStreamedString & operator<< (MAPSStreamedString & *out*, MAPSCouple< T > & *c*) `[related]`

Standard operator.

The documentation for this class was generated from the following file:

- maps.hpp

10.14 MAPSDrawingObject Struct Reference

The MAPSDrawingObject : a standard for simple overlay shapes.

Public Attributes

- int kind

  Specifies the kind of drawing object.

- int color

  Use the MAPS_RGB macro to set the color of this drawing object.

- int width

  A width for the drawing.

- int id

  An optional id for the element.

- MAPSLine line

  The drawing object is a line.

- MAPSRectangle rectangle

  The drawing object is a rectangle.

- MAPSCircle circle

  The drawing object is a circle.

- MAPSText text

  The drawing object is some text.

- MAPSSpot spot

  The drawing object is a spot.

- MAPSEllipse ellipse

  The drawing object is an ellipse.

Static Public Attributes

- static const int Line

  One kind of drawing object.

- static const int Rectangle

  Another kind of drawing object.

- static const int Circle

  Another kind of drawing object.

- static const int Text

  Another kind of drawing object.

- static const int Spot

  Another kind of drawing object.

- static const int Ellipse

  Another kind of drawing object.

User defined information

- int misc1

  Miscellaneous information 1.

- int misc2

  Miscellaneous information 2.

- int misc3

  Miscellaneous information 3.

- void $*$ userdata

  TO USE WITH CARE! C++ EXPERTS ONLY.

10.14.1 Detailed Description

The MAPSDrawingObject : a standard for simple overlay shapes. This structure can also be used as a standard to return sensor information.

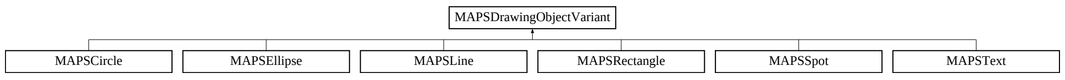
The documentation for this struct was generated from the following file:

- maps.hpp

10.15 MAPSDrawingObjectVariant Struct Reference

All drawing objects in RTMaps (circles, etc.) inherit from MAPSDrawingObjectVariant.

Inheritance diagram for MAPSDrawingObjectVariant:



10.15.1 Detailed Description

All drawing objects in RTMaps (circles, etc.) inherit from MAPSDrawingObjectVariant.

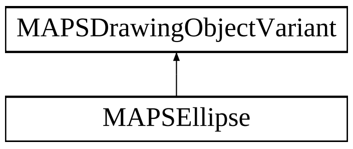
The documentation for this struct was generated from the following file:

- maps.hpp

10.16 MAPSEllipse Struct Reference

The MAPSEllipse structure.

Inheritance diagram for MAPSEllipse:



Public Attributes

- int x

  The x-center of the circle.

- int y

  The y-center of the circle.

- int sx

  The horizontal size of the ellipse.

- int sy

  The vertical size of the ellipse.

10.16.1 Detailed Description

The MAPSEllipse structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.17 MAPSEnumStruct Struct Reference

Enumeration structure.

Public Member Functions

- MAPSEnumStruct ()

  Default constructor: creates an empty enumeration.

- MAPSEnumStruct (const MAPSEnumStruct &enumStruct)

  Copy constructor.

- void FromString (const MAPSString &str, const bool selectIndexOnly=false)

  Counterparts of MAPSEnumStruct::ToString() unless selectIndexOnly is `true`; in this case, only the index is changed.

- MAPSEnumStruct & operator= (const MAPSEnumStruct &enumStruct)

  Assignement operator.

- MAPSInt32 GetSelected () const

  Retrieves the index of the selected property.

- bool Select (const MAPSString &value)

  Sets selectedEnum to the index of the string value in the collection.

- MAPSString ToString () const

  Converts the enumeration and its associated index to a single string.

Static Public Member Functions

- static bool IsEnumString (const MAPSString &str)

  Returns `true` if the string argument is a well-formatted enumeration string.

Public Attributes

- MAPSInt32 selectedEnum

  The selection index. -1 means no item has been selected yet.

- MAPSArray< MAPSString > ∗ enumValues

  A pointer to the collection of strings.

10.17.1 Detailed Description

Enumeration structure. This structure represents an array of strings with a selection index. The exchange string of a MAPSEnumStruct is like `"numberOfItems|selectedItem|item0|item1|...|itemN'`

10.17.2 Member Function Documentation

10.17.2.1 bool MAPSEnumStruct::Select (const MAPSString & *value*)

Sets *selectedEnum* to the index of the string *value* in the collection.

Returns `false` if *value* is an invalid choice.

The documentation for this struct was generated from the following file:

- maps.hpp

10.18 MAPSEvent Class Reference

The RTMaps event class.

Public Member Functions

- void Set ()

  Sets the event.

- void Reset ()

  Resets the event.

- int Wait (MAPSDelay timeout=-1)

  Waits for the event to be set by another thread.

- bool State ()

  Returns the current state of this event.

- bool SetTrigger (MAPSDelay delay)

Sets a trigger on this event.

- void ResetTrigger ()

  Resets a trigger;.

- HANDLE GetHandle ()

  Returns the Windows event handle corresponding to this MAPSEvent (Windows only function)

- MAPSEvent (bool initialState=false, const char ∗n=NULL)

  The MAPSEvent constructor.

- MAPSEvent (HANDLE h)

  Windows specific constructor.

- virtual ∼MAPSEvent ()

  The MAPSEvent destructor.

Static Public Member Functions

- static int Wait (int nCount, MAPSEvent ∗events[ ], MAPSDelay timeout=-1)

  Waits for one of a set of events to be set by another thread.

- static int MsgWait (int nCount, MAPSEvent ∗events[ ], MAPSDelay timeout=-1)

  Waits for one event to be set by another thread OR one message to arrive.

- static bool MsgInQueue ()

  Tells if there is a message in the queue for this thread.

Friends

- class MAPSWin32

10.18.1 Detailed Description

The RTMaps event class. An event is simply a 2-state variable, with wait and delayed set capabilities.

This is one of the most important classes in RTMaps. It is extensively used by the engine to rule all threads switching.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 MAPSEvent::MAPSEvent (bool *initialState* = `false`, const char ∗ *n* = `NULL`)

The MAPSEvent constructor.

Parameters

initialState	Initial state for the event. Default is `false`.
n	Optionnal name to give to the event.

10.18.2.2 MAPSEvent::MAPSEvent (HANDLE *h*)

Windows specific constructor.

Embeds a Windows event in a MAPSEvent. Useful if you want to combine Windows provided event with RTMaps provided events and use the RTMaps MAPSEvent functions to control them.

This a very low-level function, OS-specific, that must be used with care.

10.18.3 Member Function Documentation

10.18.3.1 HANDLE MAPSEvent::GetHandle ( )

Returns the Windows event handle corresponding to this MAPSEvent (Windows only function)

This must be used only for low level RTMaps programming, since this function is OS-specific.

10.18.3.2 static bool MAPSEvent::MsgInQueue ( ) `[static]`

Tells if there is a message in the queue for this thread.

This function has a meaning only for message based operating systems, like Microsoft Windows.

10.18.3.3 static int MAPSEvent::MsgWait (int *nCount*, MAPSEvent ∗ *events[ ]*, MAPSDelay *timeout* = `-1`) `[static]`

Waits for one event to be set by another thread OR one message to arrive.

This function exists only for message based operating systems, like Microsoft Windows. It allows to process some messages while processing some MAPSEvents. This is very useful to make some single-threaded Windows graphical components.

10.18.3.4 void MAPSEvent::ResetTrigger ( )

Resets a trigger;.

This will cancel any previously set trigger.

10.18.3.5 bool MAPSEvent::SetTrigger (MAPSDelay *delay*)

Sets a trigger on this event.

This function is very important. It automatically sets the event after the expiry of a given delay. This means that is any thread is waiting on this event, the latter thread will be unblocked after a finite amount of time. This enables the scheduling of threads in RTMaps.

Warning

This functions uses multimedia timers. Under windows NT, 2000, XP, the number of multimedia timers is limited to 16. If the function fails, the function returns `false`.

10.18.3.6 int MAPSEvent::Wait (MAPSDelay *timeout* = `-1`)

Waits for the event to be set by another thread.

The thread that called Wait will block until the event is set.

Warning

This is a very dangerous function since if it is badly used, it may block infinitely your RTMaps system. If you develop some components, please use MAPSComponent::Wait4Event. It is a safe function that will never dead lock your RTMaps system.

10.18.3.7 static int MAPSEvent::Wait (int *nCount*, MAPSEvent $*$ *events[ ]*, MAPSDelay *timeout* = `-1`) `[static]`

Waits for one of a set of events to be set by another thread.

The thread that called Wait will block until one of the events are set.

Warning

This is a very dangerous function since if it is badly used, it may block infinitely your RTMaps system. If you develop some components, please use MAPSComponent::Wait4Events. It is a safe function that will never dead lock your RTMaps system.

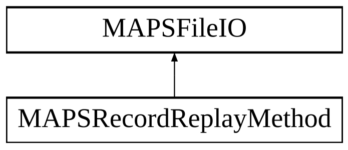
The documentation for this class was generated from the following file:

- maps.hpp

10.19 MAPSFileIO Class Reference

The RTMaps File I/O support class.

Inheritance diagram for MAPSFileIO:



Static Public Attributes

- static const char ∗ Endl

  End of line sequence for text files.

Protected Member Functions

- virtual void Unlock (void ∗)

  Unlocking function, for asynchronous I/O.

File I/O flags

Constants to use in MAPSFileIO::OpenFile4Writing and MAPSFileIO::OpenFile4Reading

- static const int Asynchronous

  Sets an asynchronous mode of operation.

- static const int NoBuffering

  Sets a no buffering mode of operation. The data are sent directly to disk. No caching.

- static const int Append

  Opens the file in append mode. It means that it won't destroy the previous content if the fils existed.

- static const int RandomAccess

  Specifies that the file will be accessed from any point. Optimizes the cache management for random access.

- static const int SequentialAccess

  Specifies that the file will always be accessed sequentially. Optimizes the cache management. (Random access is still possible but slower)

Fast file write functions

These functions take full advantage of the underlying OS and they collect automatically some performance data that are very important for the fine tuning of a RTMaps system. Please never use the functions provided by the OS for file I/O.

- MAPSFileWriteHandle ∗ FileOpen4Writing (const char ∗name, int options=0)

  Opens a file for writing.

- bool FileWrite (MAPSFileWriteHandle ∗fileHandle, const void ∗buffer, int size, void ∗elt2Unlock=NULL)

  Writes data to a file.

- bool FileWriteText (MAPSFileWriteHandle ∗fileHandle, const MAPSString &text, bool mayContainCarriageReturns=true)

  Writes text data to a file.

- void FileClose (MAPSFileWriteHandle ∗fileHandle)

  Closes the file.

File read functions

(That should have read-ahead capability)

All these functions work synchronously. No asynchronous mode is supported for reading operations.

- MAPSFileReadHandle ∗ FileOpen4Reading (const char ∗name, int options=0)

  Opens a file for reading.

- bool FileRead (MAPSFileHandle ∗fileHandle, void ∗buffer, int size, int ∗nbRead=NULL)

  Reads some data from the file.

- void FileClose (MAPSFileReadHandle ∗fileHandle)

  Closes the file.

General purpose functions

- bool FileSetPos (MAPSFileHandle ∗fileHandle, MAPSInt64 pos)

  Sets the file pointer.

- bool FileMovePos (MAPSFileHandle ∗fileHandle, int move)

  Moves to another position in the file, relative to the current position.

- MAPSInt64 FileGetPos (MAPSFileHandle ∗fileHandle)

  Returns the current position of the file pointer in the file.

- bool FileNextLine (MAPSFileHandle ∗fileHandle, MAPSString &record, MAPSInt64 ∗oldPos=NULL)

  Returns the next line in the text file.

- bool FilePreviousLine (MAPSFileHandle ∗fileHandle, MAPSString &record, MAPSInt64 ∗oldPos=NULL)

  Returns the previous line in the text file.

- bool FileFind (MAPSFileHandle ∗fileHandle, const MAPSString &lookFor, char delimiter=0, MAPSInt64 ∗pos=NULL)

  Finds a string in a file.

- bool FileFind (MAPSFileHandle ∗fileHandle, const MAPSString &lookFor, MAPSString &result, char delimiter=0, MAPSInt64 ∗pos=NULL)

  Finds a string in a file.

- bool FileRewind (MAPSFileHandle ∗fileHandle)

  Rewinds the file.

- bool FileGo2End (MAPSFileHandle ∗fileHandle)

  Goes to the end of the file.

- MAPSInt64 FileSize (MAPSFileHandle ∗fileHandle)

  Returns the size of the file.

Error management functions

- void FileWriteError ()

  Called when a write error occurs. Gets an explanation to the OS and reports it.

- void FileReadError ()

  Called when a read error occurs. Gets an explanation to the OS and reports it.

10.19.1 Detailed Description

The RTMaps File I/O support class. Provides the functions that any RTMaps module MUST use so that it can properly run in the RTMaps environment. Please forget about fopen and fclose stuff.

10.19.2 Member Function Documentation

10.19.2.1 void MAPSFileIO::FileClose (MAPSFileWriteHandle ∗ *fileHandle*) [protected]

Closes the file.

Parameters

fileHandle	Specifies the file to close

10.19.2.2 void MAPSFileIO::FileClose (MAPSFileReadHandle ∗ *fileHandle*) [protected]

Closes the file.

Parameters

fileHandle	Specifies the file to close

10.19.2.3 bool MAPSFileIO::FileNextLine (MAPSFileHandle ∗ *fileHandle,* MAPSString & *record,* MAPSInt64 ∗ *oldPos* = NULL) [protected]

Returns the next line in the text file.

Removes the line feed/carriage return character from the read line of text.

10.19.2.4 MAPSFileReadHandle∗ MAPSFileIO::FileOpen4Reading (const char ∗ *name,* int *options* = 0) [protected]

Opens a file for reading.

Parameters

name	The name of the file to open
options	Can be a combination of the following flags : • RandomAccess : Optimization flag. Specifies that the file will be accessed from any point. • SequentialAccess : Optimization flag. Specifies that the file will always be accessed sequentially.

Returns

The file handle that can be used with fileReadXXX functions. Returns `NULL` is the file does not exist.

10.19.2.5 MAPSFileWriteHandle∗ MAPSFileIO::FileOpen4Writing (const char ∗ *name*, int *options* = `0`) `[protected]`

Opens a file for writing.

Parameters

name	The file name to create or open
options	Can be a combination of the following flags : • Asynchronous : Sets an asynchronous mode of operation. This means that all write operations will return immediately, before the write operation has really finished. With this mode of operation, you must specify an element to unlock (parameter elt2Unlock) to any write operation, so that RTMaps can know when the data can be disposed of. • NoBuffering : Sets a no buffering mode of operation. This mode is very fast on Windows NT/2000 for huge transfers to disk. It does not use any cache (so saves many memory transfers) but puts the data directly on the disk. Note that the size of the data must be proportional to the size of sector on the disk. • Append : Tells not to overwrite the file if it already exists, but to append any new data to the end of the file. • RandomAccess : Optimization flag. Specifies that the file will be accessed from any point. • SequentialAccess : Optimization flag. Specifies that the file will always be accessed sequentially.

Returns

The file handle that can be used with fileWriteXXX functions. Returns `NULL` is the file creation or opening failed.

Note that this function also opens the file for reading too.

10.19.2.6 bool MAPSFileIO::FilePreviousLine (MAPSFileHandle ∗ *fileHandle*, MAPSString & *record*, MAPSInt64 ∗ *oldPos* = `NULL`) `[protected]`

Returns the previous line in the text file.

Removes the line feed/carriage return character from the read line of text.

10.19.2.7 bool MAPSFileIO::FileRead (MAPSFileHandle ∗ *fileHandle*, void ∗ *buffer*, int *size*, int ∗ *nbRead* = `NULL`) `[protected]`

Reads some data from the file.

Parameters

fileHandle	Specifies the file to read from

buffer	Specifies the buffer to transfer the read bytes to
size	Max size of data to read
nbRead	Pointer to a variable that will get the resulting number of read bytes. If set to `NULL`, the function will not return this information. If ∗*nbRead* is 0 and the function returns `true`, then the end of file was reached before the call to the function.

Returns

Returns `false` if an error was detected. Note that if the end of file is reached, this function returns `true`, but the *nbRead* result may be less than *size*.

10.19.2.8 bool MAPSFileIO::FileWrite (MAPSFileWriteHandle ∗ *fileHandle*, const void ∗ *buffer*, int *size*, void ∗ *elt2Unlock* = `NULL`) `[protected]`

Writes data to a file.

Parameters

fileHandle	Specifies the file to write to
buffer	Specifies where the data must be taken from
size	The size of data to write
elt2Unlock	The function automatically calls MAPSFileIO::Unlock with this parameter when the file write has finished. Should be `NULL` with synchronous write operations.

Returns

`true` if the operation has succeeded, `false` otherwise.

10.19.2.9 bool MAPSFileIO::FileWriteText (MAPSFileWriteHandle ∗ *fileHandle*, const MAPSString & *text*, bool *mayContainCarriageReturns* = `true`) `[protected]`

Writes text data to a file.

Parameters

fileHandle	Specifies the file to write to
text	The string to write to the file
mayContain-CarriageRe-turns	Indicates that the string may contains '\n' characters. If so, RTMaps will convert all '\n' characters to the MAPSFileIO::Endl string which depends on the underlying operating system, so that the resulting text file can be read with a text editor. If you do know that your file does not contain '\n' characters, set this parameter to false and RTMaps will spare some time as it will not try to find out the '\n' in your string.

Returns

`true` if the operation has succeeded, `false` otherwise.

10.19.2.10 virtual void MAPSFileIO::Unlock (void *) `[protected, virtual]`

Unlocking function, for asynchronous I/O.

Default implementation : do nothing

10.19.3 Member Data Documentation

10.19.3.1 const char* MAPSFileIO::Endl `[static]`

End of line sequence for text files.

The presence of the carriage return ('\r') character depends on the OS. This sequence of characters defines the behaviour for text files in RTMaps.

For linux and QNX, this is set to '\n'.

For the Windows operating systems, this is set to '\r\n'

The documentation for this class was generated from the following file:

- maps.hpp

10.20 MAPSFileReadHandle Class Reference

The class that is used to manage all RTMaps read file operations.

Inherits MAPSFileHandle.

Friends

- class MAPSFileIO

10.20.1 Detailed Description

The class that is used to manage all RTMaps read file operations.

The documentation for this class was generated from the following file:

- maps.hpp

10.21 MAPSFileWriteHandle Class Reference

The class that is used to manage all RTMaps write file operations.

Inherits MAPSFileHandle.

Friends

- class MAPSFileIO

10.21.1 Detailed Description

The class that is used to manage all RTMaps write file operations.

The documentation for this class was generated from the following file:

- maps.hpp

10.22 MAPSHashTable< TKey, TContent, H > Class Template Reference

Inherits MAPSBasicHashTable.

Public Member Functions

- void SetAutoAdjust (const bool x)

  Sets the auto size adjustment behaviour of the hash table.

- int HashSize () const

  Returns the size of the hash table.

- int Size () const

  Returns the number of elements currently in the hash table.

- int NbElts () const

  Returns the number of elements currently in the hash table.

- float Efficiency () const

  Computes the number of buckets per stored element. When <1, the hash table should be resized.

- void Insert (const TKey &key, const TContent &content)

  Inserts element content with key key into the hash table.

- TContent & Insert (const TKey &key)

  Inserts a new element with key key into the hash table. Returns a reference to the content that you can further set up.

- TContent ∗ Lookup (const TKey &key) const

  Finds the element with key key. Returns `NULL` if that element cannot be found in the hash table.

- TContent Remove (const TKey &key)

  Returns the element the key of which is key and removes it from the hash table.

- MAPSHashTable & operator-= (const TKey &key)

  Removes element the key of which is key and returns the hash table itself (faster than Remove())

- TContent Remove (MAPSHashTableIterator &it)

  Removes element corresponding to iterator it from the hash table, and returns its content.

- MAPSHashTable & operator-= (MAPSHashTableIterator &it)

  Removes element corresponding to iterator it from the hash table. Returns the hash table itself (faster than Remove())

- MAPSHashTableIterator Iterator () const

  Returns a new iterator.

- void ∗ End () const

  STL-like End()

- TKey & Key (const MAPSHashTableIterator &it) const

  Returns the current key of an iterator.

- TContent & Content (const MAPSHashTableIterator &it) const

  Returns the current content of an iterator.

- TContent & operator[] (const MAPSHashTableIterator &it) const

  Returns the current content of an iterator.

- MAPSPair< TKey, TContent > & Pair (const MAPSHashTableIterator &it) const

  Returns the current pair (key, content) of an iterator.

- MAPSHashTableIterator & Next (MAPSHashTableIterator &it) const

  Iterates...

- void Resize (const int size)

  Resizes the hash table to size (number of buckets)

- bool Adjust ()

  Adjusts the size of the hash table if needed (smart function): double the current size of the hash while the efficiency is <1.

- void Clear (const int n=16)

  Clears the content of the hash table.

- MAPSHashTable (const int n=16, const bool adjust=true)

  Constructor.

10.22.1 Detailed Description

template<typename TKey, typename TContent, typename H> class MAPSHashTable< TKey, TContent, H >

The RTMaps hash table class. You should not expect any order of the items. Hash table initialization example :

```
MAPSHashTable<MAPSString,int,MAPSStringHashFunction> hash_table(1000);
```

Here is an example of hash table use :

```
MAPSStringHashTable<int> HashTable(500);
HashTable.Insert("Hello",20);
HashTable.Insert("World",30);
HashTable.Insert("!",40);
int *res=HashTable.Lookup("Hello");
std::cout<<"res : "<<*res<<std::endl;
res=HashTable.Lookup("World");
std::cout<<"res : "<<*res<<std::endl;
HashTable.Remove("Hello");
res=HashTable.Lookup("Hello"); // Now res is NULL
MAPSHashTableIterator it(HashTable);
while (it) {
    std::cout<<HashTable.Key(it)<<" = "<<HashTable.Content(it)<<std::endl;
    it++;
}
```

10.22.2 Constructor & Destructor Documentation

10.22.2.1 template<typename TKey , typename TContent , typename H > MAPSHashTable< TKey, TContent, H >::MAPSHashTable (const int *n* = `16`, const bool *adjust* = `true`) `[explicit]`

Constructor.

Parameters

n	must be set so that the number of elements that will be put in the hash table should be less than 3*n, unless the hash table will work but will lose its efficiency. The default value for *n* is 16. We strongly recommend that you provide your own convenient value.

adjust	when set to `true`, the size of the hash table is automatically adjusted when needed with Adjust() function.

10.22.3 Member Function Documentation

10.22.3.1 template<typename TKey , typename TContent , typename H > void MAPSHashTable< TKey, TContent, H >::Clear (const int *n* = `16`)

Clears the content of the hash table.

Parameters

n	Same as *n* parameter in the constructor MAPSHashTable::MAPSHashTable(int,bool)

10.22.3.2 template<typename TKey, typename TContent, typename H> void MAPSHashTable< TKey, TContent, H >::Insert (const TKey & *key,* const TContent & *content*)

Inserts element *content* with key *key* into the hash table.

Warning

This function does not check if an element with the same key exists. If it happens to be the case, the hash table content will be inconsistent.

10.22.3.3 template<typename TKey, typename TContent, typename H > TContent & MAPSHashTable< TKey, TContent, H >::Insert (const TKey & *key*)

Inserts a new element with key *key* into the hash table. Returns a reference to the content that you can further set up.

Warning

This function does not check if an element with the same key exists. If it happens to be the case, the hash table content will be inconsistent.

10.22.3.4 template<typename TKey, typename TContent, typename H> MAPSHashTableIterator& MAPSHashTable< TKey, TContent, H >::Next (MAPSHashTableIterator & *it*) const

Iterates...

Note that the elements in the hash table are NOT sorted.

The documentation for this class was generated from the following file:

- maps.hpp

10.23 MAPSHashTableIterator Class Reference

The RTMaps hash table iterator.

Public Member Functions

- MAPSHashTableIterator & operator++ ()

    The ++ operator : goes to the next element in the associated hash table.

- MAPSHashTableIterator operator++ (int)

    The ++ operator : goes to the next element in the associated hash table.

- void Reset ()

    Resets the iterator.

- MAPSHashTableIterator ()

    Default constructor. Not that you cannot use the iterator directly. You must associate it to a hash table through a call to MAPSHashTable::Iterator or MAPSHashTable::Next.

- MAPSHashTableIterator (const MAPSBasicHashTable &hash_table)

    Another constructor that associates this iterator to a hash table.

10.23.1 Detailed Description

The RTMaps hash table iterator.

The documentation for this class was generated from the following file:

- maps.hpp

10.24 MAPSImage Struct Reference

The RTMaps Image type.

Public Attributes

- MAPSInt32 width

    The width of the image in pixels.

- MAPSInt32 height

    The height of the image in pixels.

- MAPSInt32 imageSize

The number of bytes the stored image actually consumes.

- MAPSInt32 bufferSize

  The number of bytes allocated in the buffer.

- void ∗ imageId

  Points to an identifier for the image.

- void ∗ imageId2

  Supplements imageId.

- char ∗ imageData

  Points to the actual bytes composing the image.

- char imageCoding [4]

  4 characters describing the coding used for the image

- char imageType [4]

  4 characters that supplement imageCoding (generally "COLR" or "MONO")

10.24.1 Detailed Description

The RTMaps Image type. The IplImage structure must be preferably used when dealing with images, since these structures can be directly used with the IPL (Image Processing Library) or the CV lib (Open Source Computer Vision Library).

However, the IplImage structure was not designed to contain some compressed images or some images coded neither pixel or planar oriented (for instance YUYV images). To deal with these kind of images, the MAPSImage structure should be used.

The MAPSImage structure is much simpler than the IplImage structure : it retains only the main parameters, i.e. the width or the height of the image, and the imageSize parameters. Some new members appeared :

- bufferSize, which contains the size in bytes of the buffer actually allocated
- imageId, that can point to an identifier of the content, for instance a string.
- imageId2, which can supplement imageId
- imageCoding, which can contain four characters specifying the coding used for the image, for instance "JPEG" or "YUYV"
- imageType, which can supplement imageCoding. Generally, it is "COLR" or "MONO"

See also

IplImage

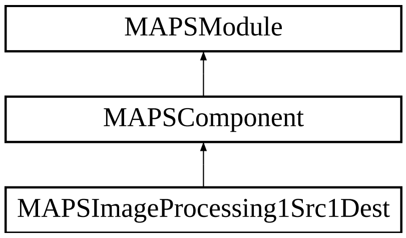
The documentation for this struct was generated from the following file:

- maps.hpp

10.25 MAPSImageProcessing1Src1Dest Class Reference

Image Processing component base class with 1 input and 1 output images.

Inheritance diagram for MAPSImageProcessing1Src1Dest:



10.25.1 Detailed Description

Image Processing component base class with 1 input and 1 output images.

The documentation for this class was generated from the following file:

- MAPSImageProcessing1Src1Dest.h

10.26 MAPSImageProcessing1Src1DestDefinition Struct Reference

Definition structure for an image processing algorithm (1 Src 1 Dest operation).

Public Attributes

- int transform

  Algorithm identifier.

- const char * str

  Algorithm name.

- int nbIntParams

  Required number of integer parameters for this algorithm.

- int nbDoubleParams

  Required number of double parameters for this algorithm.

- const char ∗ colorModel

  Output color model for auto-allocation feature.

- int nbClicksUsed

  Specify the number of 2D points the algorithm may expect.

10.26.1 Detailed Description

Definition structure for an image processing algorithm (1 Src 1 Dest operation).

The documentation for this struct was generated from the following file:

- MAPSImageProcessing1Src1Dest.h

10.27 MAPSImageProcessing1Src1DestParams Struct Reference

Parameter structure for an image processing algorithm (1 Src 1 Dest operation).

Public Attributes

- int ∗ intParams

  Integer parameters array.

- int nbIntParams

  Number of integer parameters in the intParams array.

- double ∗ dblParams

  Double parameters array.

- int nbDblParams

  Number of double parameters in the dblParams array.

- MAPSPoint2D ∗ clicks

  2D points array.

- int nbClicks

  Number of 2D points in the clicks array.

- MAPSTimestamp ts

  Timestamp of input image.

- MAPSTimestamp toi

  TimeOfIssue of input image.

10.27.1 Detailed Description

Parameter structure for an image processing algorithm (1 Src 1 Dest operation).

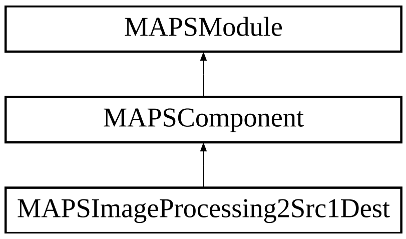
The documentation for this struct was generated from the following file:

- MAPSImageProcessing1Src1Dest.h

10.28 MAPSImageProcessing2Src1Dest Class Reference

Image Processing component base class with 2 input and 1 output images.

Inheritance diagram for MAPSImageProcessing2Src1Dest:



10.28.1 Detailed Description

Image Processing component base class with 2 input and 1 output images.

The documentation for this class was generated from the following file:

- MAPSImageProcessing2Src1Dest.h

10.29 MAPSImageProcessing2Src1DestDefinition Struct Reference

Definition structure for an image processing algorithm (2 Src 1 Dest operation).

Public Attributes

- int transform

  Algorithm identifier.

- const char ∗ str

  Algorithm name.

- int nbIntParams

  Required number of integer parameters for this algorithm.

- int nbDoubleParams

  Required number of double parameters for this algorithm.

- const char ∗ colorModel

  Output color model for auto-allocation feature.

- int nbClicksUsed

  Specify the number of 2D points the algorithm may expect.

10.29.1 Detailed Description

Definition structure for an image processing algorithm (2 Src 1 Dest operation).

The documentation for this struct was generated from the following file:

- MAPSImageProcessing2Src1Dest.h

10.30 MAPSImageProcessing2Src1DestParams Struct Reference

Parameter structure for an image processing algorithm (2 Src 1 Dest operation).

Public Attributes

- int ∗ intParams

  Integer parameters array.

- int nbIntParams

  Number of integer parameters in the intParams array.

- double ∗ dblParams

  Double parameters array.

- int nbDblParams

  Number of double parameters in the dblParams array.

- MAPSPoint2D ∗ clicks

  2D points array.

- int nbClicks

  Number of 2D points in the clicks array.

- MAPSTimestamp ts [2]

Timestamps of input images (array of 2 timestamps)

- MAPSTimestamp toi [2]

  TimeOfIssues of input images (array of 2 timestamps)

10.30.1 Detailed Description

Parameter structure for an image processing algorithm (2 Src 1 Dest operation).

The documentation for this struct was generated from the following file:

- MAPSImageProcessing2Src1Dest.h

10.31 MAPSInput Class Reference

The RTMaps Module Input class.

Public Member Functions

- virtual bool IsConnected ()=0

  Returns `true` if the input is connected to another component output.

- virtual MAPSOutput ∗ ConnectedOutput ()=0

  Returns a pointer to the output (of type MAPSOutput) connected to this input.

- virtual MAPSTypeInfo & Type ()=0

  Gets the type of the output to which this input is connected.

- virtual MAPSString & Name ()=0

  Returns the name of the input.

- virtual const char ∗ ShortName ()=0

  Returns the short name of the input.

- virtual bool DataAvailableInFIFO ()=0

  Checks if any data is available in the FIFO connected to this input.

10.31.1 Detailed Description

The RTMaps Module Input class.

10.31.2 Member Function Documentation

10.31.2.1 virtual MAPSOutput* MAPSInput::ConnectedOutput ( ) `[pure virtual]`

Returns a pointer to the output (of type MAPSOutput) connected to this input.

Returns `NULL` if no output is connected

10.31.2.2 virtual bool MAPSInput::IsConnected ( ) `[pure virtual]`

Returns `true` if the input is connected to another component output.

It is useless to try to get some data from an input that is not connected.

See also

MAPSComponent::StartReading

10.31.2.3 virtual MAPSString& MAPSInput::Name ( ) `[pure virtual]`

Returns the name of the input.

In the form `componentName.inputName`

10.31.2.4 virtual const char* MAPSInput::ShortName ( ) `[pure virtual]`

Returns the short name of the input.

This is the name after the dot

10.31.2.5 virtual MAPSTypeInfo& MAPSInput::Type ( ) `[pure virtual]`

Gets the type of the output to which this input is connected.

Note that generally this function is used after a call to MAPSComponent::StartReading, that is after getting some data from the connected output, so that it is not generally necessary to check the connection.

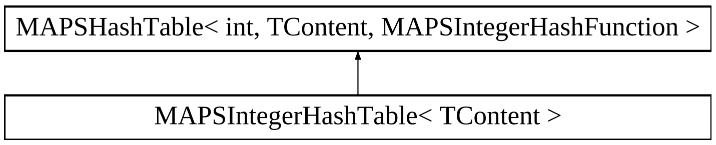
The documentation for this class was generated from the following file:

- maps.hpp

10.32 MAPSIntegerHashTable< TContent > Class Template Reference

The RTMaps integer hash table template class.

Inheritance diagram for MAPSIntegerHashTable< TContent >:


MAPSHashTable< int, TContent, MAPSIntegerHashFunction >
MAPSIntegerHashTable< TContent >

Public Member Functions

- MAPSIntegerHashTable (int n=16)

  Constructor.

10.32.1 Detailed Description

template<typename TContent> class MAPSIntegerHashTable< TContent >

The RTMaps integer hash table template class. As its name says, the key is an integer.

The documentation for this class was generated from the following file:

- maps.hpp

10.33 MAPSIOElt Class Reference

The RTMaps I/O Buffer Element.

Public Member Functions

- virtual void ∗& Data ()=0

  Returns a pointer to the data contained in the buffer element.

- virtual MAPSTimestamp & TimeOfIssue ()=0

  Returns the time of issue of the I/O element.

- virtual MAPSTimestamp & Timestamp ()=0

  Gets/Sets the Timestamp of the data contained in the buffer element.

- virtual MAPSInt32 & VectorSize ()=0

  Gets/Sets the variable size of a vector (must be less than the allocated size (the max size) returned by MAPSIOElt::Size()

- virtual MAPSInt32 & Size ()=0

  Returns the allocated size for the buffer element.

- virtual MAPSInt32 & BufferSize ()=0

  Returns the allocated size for the buffer element.

- virtual MAPSInt64 & Frequency ()=0

  Gets/Sets the frequency of the data contained in the buffer element.

- virtual MAPSInt32 & Quality ()=0

  Gets/Sets the quality of the data contained in the buffer element.

- virtual MAPSTypeInfo & Type ()=0

  Returns the type of the data contained in the buffer element.

- virtual MAPSInt32 & Integer (int index=0)=0

  Gets/Sets the value of the data contained in the buffer element (must be of integer type)

- virtual MAPSInt64 & Integer64 (int index=0)=0

  Gets/Sets the value of the data contained in the buffer element (must be of MAPSInt64 type)

- virtual MAPSFloat & Float (int index=0)=0

  Gets/Sets the value of the data contained in the buffer element (must be of MAPSFloat type)

- virtual char ∗ Text ()=0

  Gets/Sets the text content of the data contained in the buffer element (must be of TextAscii type)

- virtual char ∗ TextAscii ()=0

  Gets/Sets the text content of the data contained in the buffer element (must be of TextAscii type)

- virtual wchar_t ∗ TextUnicode ()=0

  Gets/Sets the text content of the data contained in the buffer element (must be of TextUnicode type)

- virtual ::IplImage & IplImage ()=0

  Gets a reference to the data contained in the buffer element (must be of IplImage type)

- virtual ::MAPSImage & Image ()=0

  Gets a reference to the data contained in the buffer element (must be of MAPSImage type)

- virtual ::MAPSImage & MAPSImage ()=0

  Gets a reference to the data contained in the buffer element (must be of MAPSImage type)

- virtual MAPSCANFrame & CANFrame (int index=0)=0

  Gets a reference to the data contained in the buffer element (must be of MAPSCAN-Frame type)

- virtual MAPSMatrix & Matrix ()=0

  Gets a reference to the data contained in the buffer element (must be of MAPSMatrix type)

- virtual MAPSRealObject & RealObject (int index=0)=0

  Gets a reference to the data contained in the buffer element (must be of MAPSRe-alObject type)

- virtual MAPSDrawingObject & DrawingObject (int index=0)=0

  Gets a reference to the data contained in the buffer element (must be of MAPS-DrawingObject type)

- virtual MAPSTriangles3D & Triangles3D ()=0

  Gets a pointer to the stream packet contained in the buffer element (must be of MAP-STriangles3D type)

- virtual MAPSUInt8 ∗ Stream8 ()=0

  Gets a pointer to the stream packet contained in the buffer element (must be of MAPSStream8 type)

- virtual MAPSUInt16 ∗ Stream16 ()=0

  Gets a pointer to the stream packet contained in the buffer element (must be of MAPSStream16 type)

- virtual MAPSUInt32 ∗ Stream32 ()=0

  Gets a pointer to the stream packet contained in the buffer element (must be of MAPSStream32 type)

- virtual MAPSInt32 & Misc ()=0

  Gets/Sets additional information associated to the buffer element.

- virtual MAPSInt32 & Misc1 ()=0

  Gets/Sets additional information associated to the buffer element.

- virtual MAPSInt32 & Misc2 ()=0

  Gets/Sets additional information associated to the buffer element.

- virtual MAPSInt32 & Misc3 ()=0

  Gets/Sets additional information associated to the buffer element.

- virtual operator char ∗ ()=0

  Casts the data in the buffer element to `char*` (only for TestAscii type data)

- virtual operator wchar_t ∗ ()=0

    Casts the data in the buffer element to `wchar_t*` *(only for TestUnicode type data)*

- virtual operator MAPSInt32 & ()=0

    Casts the data in the buffer element to `int&` *(only for integer type data)*

- virtual operator MAPSInt64 & ()=0

    Casts the data in the buffer element to `MAPSInt64&` *(only for MAPSInt64 type data)*

- virtual operator MAPSFloat & ()=0

    Casts the data in the buffer element to `MAPSFloat&` *(only for MAPSFloat type data)*

- virtual operator::IplImage ∗ ()=0

    Casts the data in the buffer element to IplImage∗ (only for IplImage type data)

- virtual operator::MAPSImage & ()=0

    Casts the data in the buffer element to MAPSImage (only for MAPSImage type data)

- virtual operator MAPSCANFrame & ()=0

    Casts the data in the buffer element to MAPSCANFram& (only for MAPSCANFrame type data)

- virtual operator MAPSMatrix & ()=0

    Casts the data in the buffer element to MAPSMatrix& (only for MAPSMatrix type data)

- virtual operator MAPSRealObject & ()=0

    Casts the data in the buffer element to MAPSRealObject& (only for MAPSRealObject type data)

- virtual operator MAPSDrawingObject & ()=0

    Casts the data in the buffer element to MAPSRealObject& (only for MAPSDrawingObject type data)

- virtual operator MAPSTriangles3D & ()=0

    Casts the data in the buffer element to MAPSRealObject& (only for MAPSTriangles3D type data)

- virtual int IOEltBufferSizeInBytes (bool ∗isMemContiguous)=0

    Retrieves the size of the allocated buffer in bytes.

- virtual int IOEltUsedSizeInBytes (bool ∗isMemContiguous)=0

    Retrieves the size that is actually used in the allocated buffer in bytes.

- virtual int Serialize (void ∗dest, unsigned int maxSize)=0

    Copies the MAPSIOElt object to a continuous zone in memory.

- virtual int Deserialize (void ∗serializedIoElt, bool doAllocateIfNecessary=false)=0

  Deserializes a memory zone from a serialized MAPSIOElt to the current MAPSIOElt.

Static Public Member Functions

- static int SerializedIOEltSizeInBytes (void ∗serializedIoElt)

  Retrieves the memory size needed to deserialize a MAPSIOElt.

- static bool IsDeserializable (void ∗serializedIoElt)

  Checks if serializedIOElt can be deserialized in the current MAPSIOElt.

10.33.1 Detailed Description

The RTMaps I/O Buffer Element. This class is of the highest interest for all RTMaps developers. It's the basic element of communication between RTMaps modules.

10.33.2 Member Function Documentation

10.33.2.1 virtual MAPSInt32& MAPSIOElt::BufferSize ( ) `[pure virtual]`

Returns the allocated size for the buffer element.

Returns the allocated size for the buffer element.

The meaning of this info depends on the data type in the buffer :

- For vectors (of integers, MAPSFloat, MAPSCANFrame, MAPSRealObject), sets the max size of the vector
- For text (TextAscii, TextUnicode), sets the max length of the string in the buffer
- For MAPSMatrix buffers, sets the max size of the matrix, i.e. the max m∗n elements in the matrix.
- For IplImage buffers, retrieves the max size of the imageData zone of the IplImage (call to MAPSOutput::AllocOutputBufferIplImage).
- For MAPSImage buffers, retrieves the max size of the imageData zone of the MAPSImage (call to MAPSOutput::AllocOutputBufferMAPSImage).

See also

MAPSOutput::AllocOutputBuffer MAPSOutput::AllocOutputBufferIplImage MAPSOutput::AllocOutputBufferMAPSImage

10.33.2.2 virtual void∗& MAPSIOElt::Data ( ) [pure virtual]

Returns a pointer to the data contained in the buffer element.

Beware : Do not set this pointer yourself. Please let RTMaps set it itself during buffer allocation. The setting is allowed only for "expert" cases

10.33.2.3 virtual int MAPSIOElt::Deserialize (void ∗ *serializedIoElt*, bool *doAllocateIfNecessary* = false) [pure virtual]

Deserializes a memory zone from a serialized MAPSIOElt to the current MAPSIOElt.

Parameters

serializedIoElt	Pointer to the memory zone containing the serialized MAPSIOElt.

Warning

The validity of this pointer cannot be checked entirely. Use it at your own risk.

Parameters

doAllocateIfNecessary	If `true` and if the current MAPSIOElt (the destination one) is not allocated, the allocation is performed automatically. Deallocation then has to be done manually by the programmer. Usually let this parameter to `false`.

10.33.2.4 virtual MAPSInt64& MAPSIOElt::Frequency ( ) [pure virtual]

Gets/Sets the frequency of the data contained in the buffer element.

The frequency can be used by replay facilities to perfectly link data packets one to the others, when the time stamping are not accurate enough.

The value of frequency must be set such that 1000 means 1Hz. This implies that no signal slower than 0.001Hz can be considered, but signals with very high frequencies (GHz...) can.

Note that the frequency is considered as a valid value only when the flag MAPS::FrequencyFlag is set in the type specification.

See also

MAPSIOElt::Type

10.33.2.5 virtual int MAPSIOElt::IOEltBufferSizeInBytes (bool ∗ *isMemContiguous*) [pure virtual]

Retrieves the size of the allocated buffer in bytes.

Parameters

isMemContiguous	This boolean is set to `true` if the data contained in the MAPSIOElt is kept in a contiguous memory zone. This is not the case for exemple for a MAPSIOElt containing an IplImage: the imageData can be stored anywhere in memory.

Returns

the size of the allocated buffer in bytes, or -1 if the function fails.

10.33.2.6 virtual int MAPSIOElt::IOEltUsedSizeInBytes (bool ∗ *isMemContiguous*) `[pure virtual]`

Retrieves the size that is actually used in the allocated buffer in bytes.

Parameters

isMemContiguous	This boolean is set to `true` if the data contained in the MAPSIOElt is kept in a contiguous memory zone. This is not the case for exemple for a MAPSIOElt containing an IplImage: the imageData can be stored anywhere in memory.

Returns

the size used in the allocated buffer in bytes, or -1 if the function fails.

10.33.2.7 static bool MAPSIOElt::IsDeserializable (void ∗ *serializedIoElt*) `[static]`

Checks if serializedIOElt can be deserialized in the current MAPSIOElt.

Parameters

serializedIoElt	Pointer to the serialized MAPSIOElt

10.33.2.8 virtual MAPSIOElt::operator MAPSFloat & ( ) `[pure virtual]`

Casts the data in the buffer element to `MAPSFloat&` (only for MAPSFloat type data)

Use `MAPSFloat& MAPSIOElt::Float(int index=0)` for MAPSFloat vectors

10.33.2.9 virtual MAPSIOElt::operator MAPSInt32 & ( ) `[pure virtual]`

Casts the data in the buffer element to `int&` (only for integer type data)

Use `int& MAPSIOElt::Integer(int index=0)` for integer vectors

10.33.2.10 virtual MAPSIOElt::operator MAPSInt64 & ( ) `[pure virtual]`

Casts the data in the buffer element to `MAPSInt64&` (only for MAPSInt64 type data)

Use `MAPSInt64& MAPSIOElt::Integer64(int index=0)` for integer64 vectors

10.33.2.11 virtual MAPSInt32& MAPSIOElt::Quality ( ) `[pure virtual]`

Gets/Sets the quality of the data contained in the buffer element.

The quality contains an additional information about the data, for instance a signal/noise ratio or compression ratio.

Note that the quality is considered as a valid value only when the flag MAPS::QualityFlag is set in the type specification.

See also

MAPSIOElt::Type

10.33.2.12 virtual int MAPSIOElt::Serialize (void $*$ *dest,* unsigned int *maxSize*) `[pure virtual]`

Copies the MAPSIOElt object to a continuous zone in memory.

Warning

Since MAPSIOElt structure can change with the version of RTMaps, do not use this function for recording purpose: your records may not be compatible with future versions.

Parameters

dest	Pointer to the destination memory zone. This zone has to be already allocated.
maxSize	Determines the size of the memory zone pointed by dest. The function fails if the zone is too small to contain the entire MAPSIOElt.

Returns

the size of the memory that has been used to serialize the MAPSIOElt, or -1 if the function fails (*maxSize* is too small for example).

10.33.2.13 static int MAPSIOElt::SerializedIOEltSizeInBytes (void $*$ *serializedIoElt*) `[static]`

Retrieves the memory size needed to deserialize a MAPSIOElt.

sizeof(MAPSIOElt) is not taken into account there. So add it if you need.

Parameters

serialize-dIoElt	Pointer to the serialized MAPSIOElt

Returns

Returns -1 if the function fails.

10.33.2.14 virtual MAPSInt32& MAPSIOElt::Size ( ) `[pure virtual]`

Returns the allocated size for the buffer element.

The meaning of this info depends on the data type in the buffer :

- For vectors (of integers, MAPSFloat, MAPSCANFrame, MAPSRealObject), sets the max size of the vector
- For text (TextAscii, TextUnicode), sets the max length of the string in the buffer
- For MAPSMatrix buffers, sets the max size of the matrix, i.e. the max m∗n elements in the matrix.
- For IplImage buffers, retrieves the max size of the imageData zone of the IplImage (call to MAPSOutput::AllocOutputBufferIplImage).
- For MAPSImage buffers, retrieves the max size of the imageData zone of the MAPSImage (call to MAPSOutput::AllocOutputBufferMAPSImage).

See also

MAPSOutput::AllocOutputBuffer MAPSOutput::AllocOutputBufferIplImage MAPSOutput::AllocOutputBufferMAPSImage

10.33.2.15 virtual MAPSTimestamp& MAPSIOElt::TimeOfIssue ( ) `[pure virtual]`

Returns the time of issue of the I/O element.

The time of issue is the timestamp corresponding to the instant when the element was issued by the writing component. This might not be the same number as the timestamp of the data itself (which can be older and may have flowed through several components, or which may have been time stamped by a hardware device).

Not that you should not set this variable, since the time of issue time stamping is done automatically by the RTMaps system itself. In fact, setting this parameter can be useful in very "expert" cases.

10.33.2.16 virtual MAPSTimestamp& MAPSIOElt::Timestamp ( ) [pure virtual]

Gets/Sets the Timestamp of the data contained in the buffer element.

The timestamp is the real data time stamping. It is generally set by the acquisition component that grabbed the data.

If you do generate some data in your own components, you can time stamp them at your will, for instance by using "IOElt->Timestamp()=MAPS::CurrentTime()". If you do not timestamp yourself your data, RTMaps will do it for you and the timestamp will match the time of issue timestamp (i.e. the "IOElt->Timestamp()=MAPS::CurrentTime()" is executed within the call to StopWriting).

If you process some data in you own components, do not forget to propagate the timestamp of the incoming data, generally by writing "IOEltWriting->Timestamp()=IOEltReading->Timestamp()". This is very important, because if you never do that, you will not be able to trace back the origin of your data in the data flow.

10.33.2.17 virtual MAPSInt32& MAPSIOElt::VectorSize ( ) [pure virtual]

Gets/Sets the variable size of a vector (must be less than the allocated size (the max size) returned by MAPSIOElt::Size()

For variable length vectors, this function allows to set or retrieve the actual size of the vector, as opposed to the max size of the vector (the size allocated in memory for the buffer).

The vector size is automatically set to Size() by RTMaps if you do not set it. This means that the vector is considered by default to be full.

The documentation for this class was generated from the following file:

- maps.hpp

10.34 MAPSList< T >::MAPSIterator Class Reference

Iterator on MAPSList objects.

Public Member Functions

- MAPSIterator ()

  Default constructor.

- MAPSIterator (const MAPSIterator &iter)

  Copy constructor.

- MAPSIterator (const MAPSList< T > &li)

  Constructor associated to a MAPSList.

- T & operator∗ ()

  Direct access to the data of the list item.

- bool HasNext ()

  Returns `true` if the iterator has a successor.

- bool End ()

  Returns `true` if the iterator corresponds to the end of the list.

- bool Initialized ()

  Returns `true` if the iterator is associated to a real list item.

- void Reset ()

  Puts the iterator in an uninitialized state.

10.34.1 Detailed Description

template<typename T> class MAPSList< T >::MAPSIterator

Iterator on MAPSList objects. This is a more efficient alternative to MAPSListIterator. Its usage is close to its STL counterpart. Following is an example that displays the content of a list of `int`:

```
MAPSList<int> li;
// ... populate li ...
// Get an iterator it on the list li
MAPSList<int>::MAPSIterator it(li);
// And walk through the list
for(;!it.End();++it) {
   std::cout << (*it) << std::endl;
}
```

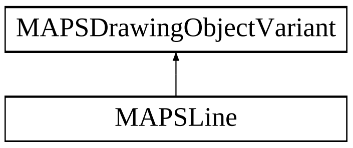
The documentation for this class was generated from the following file:

- maps.hpp

10.35 MAPSLine Struct Reference

The MAPSLine structure.

Inheritance diagram for MAPSLine:



Public Attributes

- int x1

  First point x.

- int y1

  First point y.

- int x2

  Second point x.

- int y2

  Second point y.

10.35.1 Detailed Description

The MAPSLine structure.

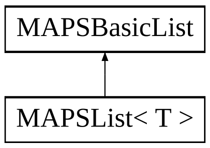
The documentation for this struct was generated from the following file:

- maps.hpp

10.36 MAPSList< T > Class Template Reference

The RTMaps double linked list template class.

Inheritance diagram for MAPSList< T >:



Classes

- class MAPSIterator

  Iterator on MAPSList objects.

Public Member Functions

- T & operator[] (MAPSListIterator it) const

Returns the element corresponding to iterator it.

- T & operator[] (int n) const

  Returns the nth element in the list (starting from 0)

- MAPSIterator Iter ()

  Gets an iterator on the first element of the list.

- MAPSIterator IterFirst ()

  Gets an iterator on the first element of the list.

- MAPSIterator IterLast ()

  Gets an iterator on the last element of the list.

- void Clear ()

  Destroys the content of the list.

- MAPSListItem< T > & Item (MAPSListIterator it)

  Access to the MAPSListItem from an iterator. Try to avoid this method.

- MAPSListItem< T > & Item (MAPSBasicListItem ∗it)

  Access to the MAPSListItem from a MAPSBasicListItem∗. Try to avoid this method.

- MAPSList< T > & operator= (const MAPSList< T > &L)

  List duplication.

- MAPSList< T > & operator+= (const MAPSList< T > &L)

  List concatenation.

- void Permutations (MAPSList< MAPSList< T > > &L)

  Computes all the possible permutations in a list.

- void Couples (MAPSList< MAPSCouple< T > > &L)

  Computes all the couples of elements of the list.

- void Bubblesort (int(∗compare_function)(T ∗t1, T ∗t2))

  Bubblesorts the list.

- MAPSList ()

  Default constructor.

- virtual ∼MAPSList ()

  Destructor.

- MAPSList (const MAPSList< T > &L)

  Copy constructor.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  MAPSStreamedString & operator<< (MAPSStreamedString &out, MAPSList< T > &L)

  Standard operator.

Adds an element to the list

- MAPSListItem< T > & Insert ()

  Inserts a new empty element in the front of the list (at position 0)

- void PushFront ()

  Inserts a new empty element in the front of the list (at position 0)

- MAPSListIterator Insert (const T &elt)

  Inserts element elt in the front of the list (at position 0)

- void PushFront (const T &elt)

  Inserts element elt in the front of the list (at position 0)

- MAPSListItem< T > & Append ()

  Appends a new empty element to the end of the list (the tail)

- void PushBack ()

  Appends a new empty element to the end of the list (the tail)

- MAPSListIterator Append (const T &elt)

  Appends element elt to the end of the list (the tail)

- void PushBack (const T &elt)

  Appends element elt to the end of the list (the tail)

- MAPSList< T > & operator<< (const T &elt)

  Appends element elt to the end of the list (the tail)

Element insertion

- MAPSListIterator InsertAfter (const MAPSListIterator itx, const T &elt)

  Inserts element elt just after the element corresponding to the iterator itx.

- MAPSListItem< T > ∗ InsertAfter (MAPSListItem< T > &it, const T &elt)

  Inserts element elt just after the element corresponding to the iterator itx.

- void InsertAfter (MAPSIterator &it, const T &elt)

  Inserts element elt just after the element corresponding to the iterator itx.

- MAPSListItem< T > & InsertAfter (const MAPSListIterator itx)

  Inserts a new empty element just after the element corresponding to the iterator itx.

- T & InsertAfter (MAPSIterator &it)

  Inserts a new empty element just after the element corresponding to the iterator itx.

- MAPSListIterator InsertBefore (const MAPSListIterator itx, const T &elt)

  Inserts element elt just before the element corresponding to the iterator itx.

- MAPSListItem< T > ∗ InsertBefore (MAPSListItem< T > &it, const T &elt)

  Inserts element elt just before the list item it.

- void InsertBefore (MAPSIterator &it, const T &elt)

  Inserts element elt just before the element corresponding to the iterator itx.

- MAPSListItem< T > & InsertBefore (MAPSListIterator itx)

  Inserts a new empty element just before the element corresponding to the iterator itx.

- T & InsertBefore (MAPSIterator &it)

  Inserts a new empty element just before the element corresponding to the iterator itx.

Deletion

- MAPSList< T > & operator-= (MAPSListIterator itx)

  Deletes the item corresponding to iterator itx.

- MAPSList< T > & operator-= (MAPSList< MAPSListIterator > &L)

  Deletes all the items in list L.

- T Delete (MAPSListIterator itx)

  Deletes the item corresponding to iterator itx.

- void Delete (MAPSIterator &it)

  Deletes the item corresponding to iterator it.

- T Remove (MAPSListIterator it)

  Deletes the item corresponding to iterator it.

- MAPSList< T > & Remove (MAPSList< MAPSListIterator > &L)

Deletes all the items in list L.

- MAPSList< T > & Delete (MAPSList< MAPSListIterator > &L)

  Deletes all the items in list L.

- T Pop ()

  Removes and returns the last item in the list (Perl-like Pop)

- T Shift ()

  Removes and returns the first element in the list (Perl-like Shift)

Order manipulation

- MAPSList< T > & Move2End (MAPSListIterator itx)

  Moves an item to the end of the list.

- void Move2End (MAPSIterator &it)

  Moves an item to the end of the list.

- MAPSList< T > & Move2Front (MAPSListIterator itx)

  Moves an item to the front of the list.

- void Move2Front (MAPSIterator &it)

  Moves an item to the front of the list.

- MAPSList< T > & Swap (MAPSListIterator itx1, MAPSListIterator itx2)

  Swaps two items.

- void Swap (MAPSIterator &it1, MAPSIterator &it2)

  Swaps two items.

10.36.1 Detailed Description

template<typename T> class MAPSList< T >

The RTMaps double linked list template class. MAPSLists have now a true iterator MAPSList::MAPSIterator. Future developments should always prefer this iterator to MAPSListIterator when possible. Methods that use MAPSListItem objects should be avoided as direct access to the list item might be removed in the next release.

10.36.2 Member Function Documentation

10.36.2.1 template<typename T> MAPSListItem<T>& MAPSList< T >::Append ( )

Appends a new empty element to the end of the list (the tail)

Prefer MAPSList::PushBack() when possible

10.36.2.2 template<typename T> MAPSListIterator MAPSList< T >::Append (const T & *elt*)

Appends element *elt* to the end of the list (the tail)

Prefer MAPSList::PushBack(const T&) when possible

10.36.2.3 template<typename T> MAPSListItem<T>& MAPSList< T >::Insert ( )

Inserts a new empty element in the front of the list (at position 0)

Prefer MAPSList::PushFront() when possible

10.36.2.4 template<typename T> MAPSListIterator MAPSList< T >::Insert (const T & *elt*)

Inserts element *elt* in the front of the list (at position 0)

Prefer MAPSList::PushFront(const T&) when possible

10.36.2.5 template<typename T> MAPSListIterator MAPSList< T >::InsertAfter (const MAPSListIterator *itx*, const T & *elt*)

Inserts element *elt* just after the element corresponding to the iterator *itx*.

Prefer MAPSList::InsertAfter(MAPSIterator&, const T&) when possible

10.36.2.6 template<typename T> MAPSListItem< T > * MAPSList< T >::InsertAfter (MAPSListItem< T > & *it*, const T & *elt*)

Inserts element *elt* just after the element corresponding to the iterator *itx*.

Prefer MAPSList::InsertAfter(MAPSIterator&, const T&) when possible

10.36.2.7 template<typename T> MAPSListItem< T > & MAPSList< T >::InsertAfter (const MAPSListIterator *itx*)

Inserts a new empty element just after the element corresponding to the iterator *itx*.

Prefer MAPSList::InsertAfter(MAPSIterator&) when possible

10.36.2.8 template<typename T> MAPSListIterator MAPSList< T >::InsertBefore (const MAPSListIterator *itx*, const T & *elt*)

Inserts element *elt* just before the element corresponding to the iterator *itx*.

Prefer MAPSList::InsertBefore(MAPSIterator&, const T&) when possible

10.36.2.9 template<typename T> MAPSListItem< T > ∗ MAPSList< T >::InsertBefore (MAPSListItem< T > & *it*, const T & *elt*)

Inserts element *elt* just before the list item *it*.

Prefer MAPSList::InsertBefore(MAPSIterator&, const T&) when possible

10.36.2.10 template<typename T> MAPSListItem< T > & MAPSList< T >::InsertBefore (MAPSListIterator *itx*)

Inserts a new empty element just before the element corresponding to the iterator *itx*.

Prefer MAPSList::InsertBefore(MAPSIterator&) when possible

10.36.3 Friends And Related Function Documentation

10.36.3.1 template<typename T > MAPSStreamedString & operator<< (MAPSStreamedString & *out*, MAPSList< T > & *L*) `[related]`

Standard operator.

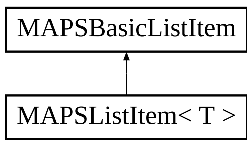
The documentation for this class was generated from the following file:

- maps.hpp

10.37 MAPSListItem< T > Class Template Reference

The list item template class.

Inheritance diagram for MAPSListItem< T >:



10.37.1 Detailed Description

template< typename T > class MAPSListItem< T >

The list item template class. Should be only for internal use.

The documentation for this class was generated from the following file:

- maps.hpp

10.38 MAPSListIterator Class Reference

The list iterator class.

Public Member Functions

- MAPSListIterator & operator++ ()

  Prefix increment operator. Increments iterator and returns the new iterator.

- MAPSListIterator operator++ (int)

  Postfix increment operator. Increments iterator and returns the old iterator.

- MAPSListIterator & operator-- ()

  Prefix decrement operator. Decrements iterator and returns the new iterator.

- MAPSListIterator operator-- (int)

  Postfix decrement operator. Decrements iterator and returns the old iterator.

- MAPSListIterator operator= (MAPSBasicListItem ∗p)

  Replaces the current list item by p.

- bool operator== (MAPSBasicListItem ∗p)

  Ckecks if two list items are the same item (not only have the same content)

- MAPSListIterator ()

  Default constructor.

- MAPSListIterator (MAPSBasicListItem ∗p)

  Constructor from a basic list item p.

- MAPSListIterator (MAPSBasicList &list)

  Initializes the iterator to the first item of a MAPSBasicList or `NULL` if list is empty.

10.38.1 Detailed Description

The list iterator class. Use this iterator in conjunction with MAPSBasicList or MAPSList.

The documentation for this class was generated from the following file:

- maps.hpp

10.39 MAPSMatrix Struct Reference

MATLAB-Like matrix (Complex and columnwise, like in fortran)

Public Member Functions

- MAPSFloat64 & Real (int row, int column)

  Gives access to the real part of one element of the matrix.

- MAPSFloat64 & Im (int row, int column)

  Gives access to the imaginary part of one element of the matrix.

- void Zero ()

  Sets all the elements of a matrix to zero.

- void Free ()

  Frees the real and imaginary part arrays. Called by the destructor.

- void Alloc (int mm, int nn)

  Allocates the real and imaginary part arrays.

- MAPSMatrix & operator= (MAPSMatrix2$<$ double $>$ &mat)

  Copy operator, from a MAPSMatrix2 (easier to use)

- MAPSMatrix & operator= (MAPSMatrix &mat)

  Copy operator, from another MAPSMatrix.

- operator MAPSMatrix2$<$ double $>$ () const

  Cast to MAPSMatrix2. Very useful to process MAPSMatrix in input of components.

- MAPSMatrix ()

  Creates a non allocated MAPSMatrix. Size is set by default to 0.

- MAPSMatrix (MAPSMatrix2$<$ double $>$ &mat)

  Constructor that takes a MAPSMatrix2 in input.

- MAPSMatrix (MAPSMatrix &mat)

Copy constructor.

- MAPSMatrix (int mm, int nn)

  Creates an allocated MAPSMatrix. Arrays are not initialized.

- virtual ∼MAPSMatrix ()

  Destructor.

Public Attributes

- MAPSInt32 m

  Number of rows in the matrix.

- MAPSInt32 n

  Number of columns in the matrix.

- MAPSFloat64 ∗ pr

  Real part. Array of double.

- MAPSFloat64 ∗ pi

  Imaginary part. Array of double.

- bool owner

  Determines if the memory was allocated by the component itself (owner=true), or by RTMaps (owner=false).

10.39.1 Detailed Description

MATLAB-Like matrix (Complex and columnwise, like in fortran) This is the RTMaps standard matrix type designed to be used as an output type. It is a very simple structure designed to contain an array of complex numbers.

Usually, this structure is used only for inter RTMaps components communication, and thus is usually allocated through a call to MAPSOutput::AllocOutputBuffer. As a matter of fact, this structure also has some constructors, so it can be used as a general purpose fortran-like matrix as well. Be aware that a much more powerful class exists for matrices computation in RTMaps : the MAPSMatrix2 class. We recommend that you use the latter for matrices computation and convert it to MAPSMatrix to output results from your components.

10.39.2 Constructor & Destructor Documentation

10.39.2.1 MAPSMatrix::MAPSMatrix (MAPSMatrix2< double > & *mat*)

Constructor that takes a MAPSMatrix2 in input.

See also

MAPSMatrix2

10.39.3 Member Function Documentation

10.39.3.1 MAPSFloat64& MAPSMatrix::Im (int *row*, int *column*)

Gives access to the imaginary part of one element of the matrix.

This function allows to read or set the imaginary part of one element of the matrix

Parameters

row	the row of the element to get access to
column	the column of the element to get access to

Returns

a reference to the imaginary part of the (row, column) element of the matrix

10.39.3.2 MAPSMatrix::operator MAPSMatrix2< double > ( ) const

Cast to MAPSMatrix2. Very useful to process MAPSMatrix in input of components.

Note that this cast only converts the real part of the MAPSMatrix.

10.39.3.3 MAPSFloat64& MAPSMatrix::Real (int *row*, int *column*)

Gives access to the real part of one element of the matrix.

This function allows to read or set the real part of one element of the matrix

Parameters

row	the row of the element to get access to
column	the column of the element to get access to

Returns

a reference to the real part of the (row, column) element of the matrix

The documentation for this struct was generated from the following file:

- maps.hpp

10.40 MAPSMatrix2< T > Class Template Reference

The RTMaps MAPSMatrix2 class, a powerful matrix management class.

Public Member Functions

- int M () const
  Returns the number of rows.
- int N () const
  Returns the number of columns.
- int NbRows () const
  Returns the number of rows.
- int NbColumns () const
  Returns the number of columns.
- int SizeX () const
  Returns the number of columns.
- int SizeY () const
  Returns the number of rows.
- int Dim1 () const
  Returns the number of rows.
- int Dim2 () const
  Returns the number of columns.
- void To (T ∗t[ ]) const
  Maps to an array of pointers to each row, for handling by C functions.

Related Functions

(Note that these are not member functions.)

- template<typename T >
  MAPSMatrix2< T > operator∗ (const MAPSMatrix2< T > &A, const MAPSMatrix2< T > &B)
  MAPSMatrix2 multiplication operator.
- template<typename T >
  MAPSMatrix2< T > operator+ (const MAPSMatrix2< T > &A, const MAPSMatrix2< T > &B)
  MAPSMatrix2 addition operator.
- template<typename T >
  MAPSMatrix2< T > operator- (const MAPSMatrix2< T > &A, const MAPSMatrix2< T > &B)

MAPSMatrix2 substraction operator.

- template<typename T >
  MAPSMatrix2< T > operator∗ (const MAPSMatrix2< T > &A, const T &b)

  Multiplication of a MAPSMatrix2 with a scalar.

- template<typename T >
  MAPSMatrix2< T > operator/ (const MAPSMatrix2< T > &A, const T &b)

  Division of a MAPSMatrix2 by a scalar.

- template<typename T >
  MAPSStreamedString & operator<< (MAPSStreamedString &out, const MAPS-Matrix2< T > &v)

  Standard operator.

Operators

- T & operator() (int i, int j) const

  Returns element (i,j) of the matrix.

- MAPSArray< T > & operator[] (int i) const

  Returns row i of the matrix. 0<= i < M().

- MAPSMatrix2 & operator= (const MAPSMatrix2 &v)

  Warning! This operator does not make a copy (too slow).

- MAPSMatrix2 & operator= (T ∗v)

  Fills the matrix with data from C array v.

- MAPSMatrix2< T > operator∼ () const

  Transposition operator.

- MAPSMatrix2< T > operator! () const

  Matrix inversion operator.

- MAPSMatrix2< T > operator∗= (const T &v)

  Matrix multiplication by a scalar.

- MAPSMatrix2< T > operator/= (const T &v)

  Matrix division by a scalar.

Functions

- void Clone ()

  Makes a clone copy of the original matrix.

- void Clear (void)

  Sets all the elements of the matrix to 0.

- void Id ()

  Sets the current matrix to identity, i.e. all the elements of the matrix will be set to 0, except the diagonal elements that will be set to 1.

- MAPSMatrix2< T > Inverse () const

  Computes the inverse of the matrix.

- T Det () const

  Computes the determinant of the matrix.

- MAPSMatrix2< T > PseudoInverse () const

  Computes the pseudo inverse of the matrix, i.e ! `(~A*A)*~A`. This is the general formula. There are some optimizations if the matrix has special properties.

- MAPSArray< T > & Row (int i) const

  Returns row i, where 0<= i < M()

- MAPSArray< T > Col (int i) const

  Returns column i, where 0<= i < N()

Constructors and destructor

- MAPSMatrix2 ()

  Default constructor.

- MAPSMatrix2 (int m, int n)

  Constructor with size of the matrix.

- MAPSMatrix2 (const MAPSMatrix2 &v)

  Copy constructor.

- virtual ~MAPSMatrix2 ()

  Destructor.

10.40.1 Detailed Description

template<typename T> class MAPSMatrix2< T >

The RTMaps MAPSMatrix2 class, a powerful matrix management class.

10.40.2 Member Function Documentation

10.40.2.1 template<typename T > void MAPSMatrix2< T >::Clone ( )

Makes a clone copy of the original matrix.

When you use the operator = (such as in A=B), A is not really a copy of B. A and B share the same matrix in memory. If you do A.Clone(), the A will have its own image of the matrix in memory, i.e. any modification of A will not affect B any longer.

10.40.2.2 template<typename T > T& MAPSMatrix2< T >::operator() (int *i*, int *j*) const

Returns element (*i*,*j*) of the matrix.

Valid values are: 0<= *i* < M(), 0<= *j* < N()

10.40.2.3 template<typename T > MAPSMatrix2& MAPSMatrix2< T >::operator= (const MAPSMatrix2< T > & *v*)

Warning! This operator does not make a copy (too slow).

It is a real mathematical operator = (The MAPSMatrix2 class has a reference counter)

10.40.2.4 template<typename T > MAPSArray<T>& MAPSMatrix2< T >::operator[] (int *i*) const

Returns row *i* of the matrix. 0<= *i* < M().

This allows the use of `[i][j]` combination to retrieve an element of the matrix.

10.40.3 Friends And Related Function Documentation

10.40.3.1 template<typename T > MAPSStreamedString & operator<< (MAPSStreamedString & *out*, const MAPSMatrix2< T > & *v*) `[related]`

Standard operator.

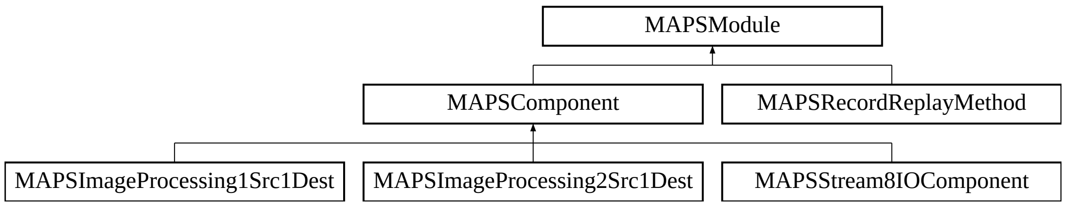
The documentation for this class was generated from the following file:

- maps.hpp

10.41 MAPSModule Class Reference

The base class for all RTMaps modules.

Inheritance diagram for MAPSModule:



Protected Member Functions

- int NbInputs (void)

  Returns the number of inputs of the module.

- int NbOutputs (void)

  Returns the number of outputs of the module.

- int NbProperties (void)

  Returns the number of properties of the module.

- int NbActions (void)

  Returns the number of actions of the module.

- MAPSInput & Input (const int inputnb)

  Gets a reference to the input inputnb.

- MAPSInput & Input (const char ∗name)

  Gets a reference to the input called name.

- MAPSOutput & Output (const int outputnb)

  Gets a reference to the output outputnb.

- MAPSOutput & Output (const char ∗name)

  Gets a reference to the output called name.

- MAPSProperty & Property (const int propertynb)

  Gets a reference to the property propertynb.

- MAPSProperty & Property (const char ∗name)

  Gets a reference to the property called name.

- MAPSAction & Action (const int actionnb)

  Gets a reference to the action actionnb.

- MAPSAction & Action (const char ∗name)

  Gets a reference to the action called name.

- bool PropertyChanged (MAPSProperty &p)

  Tells if the property value has changed.

- bool PropertyChanged (int propertynb)

  Tells if the property value has changed.

- bool PropertyChanged (const char ∗name)

  Tells if the property value has changed.

- void AcknowledgePropertyChanged (MAPSProperty &p)

  Acknowledges the property value change.

- void AcknowledgePropertyChanged (int propertynb)

  Acknowledges the property value change.

- void AcknowledgePropertyChanged (const char ∗name)

  Acknowledges the property value change.

- int RunningThreads ()

  Returns the number of threads currently associated to the component.

- int & ReaderHandle (int inputnb)

  Returns the FastI/O reader handle of input inputnb.

- bool IsConnected (MAPSInput &input)

  Returns `true` if input input is connected to another component output. cf. MAPSInput::IsConnected.

- bool DataAvailableInFIFO (MAPSInput &input)

  Returns `true` if some data are available in FIFO connected to this input.

- bool IsFIFOEmpty (MAPSOutput &output)

  Returns `true` if the FIFO connected is empty.

- bool IsFIFOFull (MAPSOutput &output)

  Returns `true` if the FIFO connected is full.

- void Wait4Handshake (MAPSOutput &output)

  Waits for a handshake on output.

MAPSEvent management

- bool Wait4Event (MAPSEvent ∗event, MAPSDelay timeout=MAPS::Infinite)

  Waits for an event to occur.

- bool Wait4Event (MAPSEvent &event, MAPSDelay timeout=MAPS::Infinite)

  Waits for an event to occur.

- int Wait4Events (int nCount, MAPSEvent ∗events[ ], MAPSDelay timeout=MAPS::Infinite, bool addIsDyingEvent=true)

  Waits for some events to occur.

- int MsgWait4Event (MAPSEvent ∗event, MAPSDelay timeout=MAPS::Infinite)

  Waits for an event to occur OR a message to come.

- int MsgWait4Events (int nCount, MAPSEvent ∗events[ ], MAPSDelay timeout=MAPS::Infinite, bool addIsDyingEvent=true)

  Waits for some events to occur OR a message to come.

Virtual time management functions

- bool Rest (MAPSDelay delay, MAPSEvent ∗cancel=NULL)

  Blocks the current thread for a delay of delay microseconds.

- bool Wait (MAPSTimestamp timestamp, MAPSEvent ∗cancel=NULL)

  Blocks the current thread until the time reaches timestamp.

Death management functions

- bool IsDying (void)

  Returns `true` *if the module is currently dying (but is not dead).*

- bool IsDead (void)

  Returns `true` *if the module is dead.*

Threads management functions

- void CreateThread (MAPSThreadFunction)

  Creates a thread associated to the module.

- void Die ()

Function to be called by the module itself during termination. Ask all the associated threads to die.

- void OneMoreLife (void)

  Adds a "life" to the module. The "life" number is a counter for the threads associated to the module.

- void LoseOneLife (void)

  Removes a "life" from the module. The "life" number is a counter for the threads associated to the module.

- void Start ()

  To be called when the module starts. It correctly resets the dying and death events and variables.

- bool Wait4Death ()

  Waits for the death of all associated threads.

Error management functions

- virtual void Error (const char ∗string, int importance=1)

  Emits an error.

Reporting management functions

- virtual void ReportError (const char ∗string, int importance=0)

  Reports an error.

- virtual void ReportWarning (const char ∗string, int importance=0)

  Reports a warning.

- virtual void ReportInfo (const char ∗string, int importance=0)

  Reports a piece of information.

- virtual void Report (const char ∗string, int type=MAPS::Info, int importance=0)

  General report function.

DirectSet functions

These functions do not check the `canBeModifierAfterStart` and `readOnly` parameters of properties, so the RTMaps modules can set their own properties with these functions in all situations. Usual MAPSModule::Set functions check the parameters of properties, and so cannot be used if for instance `readOnly` is set.

MAPSModule::DirectSet functions are protected, so they can only be used in RTMaps modules. On the contrary, MAPSModule::Set functions are public and thus can be called by anyone.

- void DirectSet (MAPSProperty &p, bool value)

  Sets boolean value for boolean property p.

- void DirectSet (MAPSProperty &p, MAPSInt64 value)

  Sets integer value for integer property p.

- void DirectSet (MAPSProperty &p, MAPSInt32 value)
- void DirectSet (MAPSProperty &p, double value)

  Sets float value for float property p.

- void DirectSet (MAPSProperty &p, const MAPSString &value)

  Sets string value for string property p.

- void DirectSet (MAPSProperty &p, const char ∗value)
- void DirectSet (MAPSProperty &p, const char ∗enumValues, MAPSInt32 selected)

  Sets enum values to enumValues and the selected enum to selected for enum property p.

- void DirectSet (MAPSProperty &p, const MAPSString &enumValues, MAPSInt32 selected)
- void DirectSet (MAPSProperty &p, const MAPSEnumStruct &enumStruct)

  Sets enum value to enumStruct for enum property p.

- void DirectSetProperty (int propertynb, bool value)

  Sets boolean value for boolean property at index propertynb.

- void DirectSetProperty (int propertynb, MAPSInt64 value)

  Sets integer value for integer property at index propertynb.

- void DirectSetProperty (int propertynb, MAPSInt32 value)

  Sets integer value for integer property at index propertynb.

- void DirectSetProperty (int propertynb, double value)

  Sets float value for float property at index propertynb.

- void DirectSetProperty (int propertynb, const char ∗value)

  Sets string value for string property at index propertynb.

- void DirectSetProperty (int propertynb, const MAPSString &value)

  Sets string value for string property at index propertynb.

- void DirectSetProperty (int propertynb, const char ∗enumValues, MAPSInt32 selected)

  Sets enum values to enumValues and the selected enum to selected for enum property at index propertynb.

- void DirectSetProperty (int propertynb, const MAPSEnumStruct &enumStruct)

  Sets enum value to enumStruct for enum property at index propertynb.

- void DirectSetProperty (const char ∗p, bool value)

  Sets boolean value for boolean property p.

- void DirectSetProperty (const char ∗p, MAPSInt64 value)

  Sets integer value for integer property p.

- void DirectSetProperty (const char ∗p, MAPSInt32 value)

  Sets integer value for integer property p.

- void DirectSetProperty (const char ∗p, double value)

  Sets float value for float property p.

- void DirectSetProperty (const char ∗p, const char ∗value)

  Sets string value for string property p.

- void DirectSetProperty (const char ∗p, const MAPSString &value)

  Sets string value for string property p.

- void DirectSetProperty (const char ∗p, const char ∗enumValues, MAPSInt32 selected)

  Sets enum values to enumValues and the selected enum to selected for enum property p.

- void DirectSetProperty (const char ∗p, const MAPSEnumStruct &enumStruct)

  Sets enum value to enumStruct for enum property p.

- virtual void PerformanceMonitoring ()

  Performance monitoring function.

- MAPSInput & SequentialInput ()

  For a sequential module (or component), returns a reference to the input that is the source of the current execution.

- MAPSModule (const char ∗n, bool perfMon=false, bool stdProperties=true)

  Constructor.

- virtual ∼MAPSModule ()

  Virtual destructor. Overloaded.

Get/Set property functions

- virtual void Get (MAPSProperty &p, bool &value)

  Gets boolean value from boolean property p.

- virtual void Get (MAPSProperty &p, MAPSInt64 &value)

  Gets integer value from integer property p.

- virtual void Get (MAPSProperty &p, MAPSFloat &value)

  Gets float value from float property p.

- virtual void Get (MAPSProperty &p, MAPSString &value)

  Gets string value from string property p.

- virtual void Get (MAPSProperty &p, MAPSEnumStruct &enumStruct)

  Gets enum struct enumStruct from enum property p.

- void Get (MAPSProperty &p, MAPSArray< MAPSString > &enumValues, MAPSInt32 &selectedValue)

  Gets enum struct values into enumValues and selected item into selectedValue from enum property p.

- void Get (MAPSProperty &p, MAPSInt32 &value)

  Gets integer value from integer property p.

- void GetProperty (const int propertynb, bool &value)

  Gets boolean value from boolean property propertynb.

- void GetProperty (const int propertynb, MAPSInt32 &value)

  Get integer value from integer property propertynb.

- void GetProperty (const int propertynb, MAPSInt64 &value)

  Gets integer value from integer property propertynb.

- void GetProperty (const int propertynb, MAPSFloat &value)

  Gets float value from float property propertynb.

- void GetProperty (const int propertynb, MAPSString &value)

  Gets string value from string property propertynb.

- void GetProperty (const int propertynb, MAPSArray< MAPSString > &enumValues, MAPSInt32 &selectedValue)

  Gets enum struct values into enumValues and selected item into selectedValue from enum property propertynb.

- void GetProperty (const int propertynb, MAPSEnumStruct &enumStruct)

  Gets enum struct value into enumStruct from enum property p.

- void GetProperty (const char ∗p, bool &value)

  Gets boolean value from boolean property p.

- void GetProperty (const char ∗p, MAPSInt32 &value)

  Gets integer value from integer property p.

- void GetProperty (const char ∗p, MAPSInt64 &value)

  Gets integer value from integer property p.

- void GetProperty (const char ∗p, MAPSFloat &value)

  Gets float value from float property p.

- void GetProperty (const char ∗p, MAPSString &value)

  Gets string value from string property p.

- void GetProperty (const char ∗p, MAPSArray< MAPSString > &enumValues, MAPSInt32 &selectedValue)

  Gets enum struct values into enumValues and selected item into selectedValue from enum property p.

- void GetProperty (const char ∗p, MAPSEnumStruct &enumStruct)

  Gets enum struct value into enumStruct from enum property p.

- bool GetBoolProperty (const int propertynb)

  Gets boolean value from boolean property propertynb.

- MAPSInt64 GetIntegerProperty (const int propertynb)

  Gets integer value from integer property propertynb.

- MAPSFloat GetFloatProperty (const int propertynb)

  Gets float value from float property propertynb.

- MAPSString GetStringProperty (const int propertynb)

  Gets string value from string property propertynb.

- MAPSEnumStruct GetEnumProperty (const int propertynb)

  Gets enum value from enum property propertynb.

- bool GetBoolProperty (const char ∗p)

  Gets boolean value from boolean property p.

- MAPSInt64 GetIntegerProperty (const char ∗p)

  Gets integer value from integer property p.

- MAPSFloat GetFloatProperty (const char ∗p)

Gets float value from float property p.

- MAPSString GetStringProperty (const char ∗p)

  Gets string value from string property p.

- MAPSEnumStruct GetEnumProperty (const char ∗p)

  Gets enum value from enum property p.

- bool GetBoolProperty (MAPSProperty &p)

  Gets boolean value from boolean property p.

- MAPSInt64 GetIntegerProperty (MAPSProperty &p)

  Gets integer value from integer property p.

- MAPSFloat GetFloatProperty (MAPSProperty &p)

  Gets float value from float property p.

- MAPSString GetStringProperty (MAPSProperty &p)

  Gets string value from string property p.

- MAPSEnumStruct GetEnumProperty (MAPSProperty &p)

  Gets enum value from enum property p.

- virtual void Set (MAPSProperty &p, bool value)

  Sets boolean value from boolean property p.

- virtual void Set (MAPSProperty &p, MAPSInt64 value)

  Sets integer value from integer property p.

- virtual void Set (MAPSProperty &p, MAPSFloat value)

  Sets float value from float property p.

- virtual void Set (MAPSProperty &p, const MAPSString &value)

  Sets string value from string property p.

- virtual void Set (MAPSProperty &p, const MAPSEnumStruct &enumStruct)

  Sets enum struct enumStruct from enum property p.

- void Set (MAPSProperty &p, MAPSInt32 value)
- void Set (MAPSProperty &p, const char ∗value)
- void Set (MAPSProperty &p, const char ∗enumValues, MAPSInt32 selected)
- void Set (MAPSProperty &p, const MAPSString &enumValues, MAPSInt32 selected)

  Sets enum values to enumValues and the selected enum to selected for enum property p.

- void SetProperty (int propertynb, bool value)

Sets boolean value from boolean property propertynb.

- void SetProperty (int propertynb, MAPSInt32 value)

  Sets integer value from integer property propertynb.

- void SetProperty (int propertynb, MAPSInt64 value)

  Sets integer value from integer property propertynb.

- void SetProperty (int propertynb, MAPSFloat value)

  Sets float value from float property propertynb.

- void SetProperty (int propertynb, const MAPSString &value)

  Sets float value from float property propertynb.

- void SetProperty (int propertynb, const char ∗value)

  Sets string value from string property propertynb.

- void SetProperty (int propertynb, const char ∗enumValues, MAPSInt32 selected)

  Sets enum values to enumValues and the selected enum to selected for enum property propertynb.

- void SetProperty (int propertynb, const MAPSEnumStruct &enumStruct)

  Sets enum value to enumStruct for enum property propertynb.

- void SetProperty (const char ∗p, bool value)

  Sets boolean value from boolean property p.

- void SetProperty (const char ∗p, MAPSInt32 value)

  Sets integer value from integer property p.

- void SetProperty (const char ∗p, MAPSInt64 value)

  Sets integer value from integer property p.

- void SetProperty (const char ∗p, MAPSFloat value)

  Sets float value from float property p.

- void SetProperty (const char ∗p, const MAPSString &value)

  Sets string value from string property p.

- void SetProperty (const char ∗p, const char ∗value)

  Sets string value from string property p.

- void SetProperty (const char ∗p, const char ∗enumValues, MAPSInt32 selected)

  Sets enum values to enumValues and the selected enum to selected for enum property p.

- void SetProperty (const char ∗p, const MAPSEnumStruct &enumStruct)

  Sets enum value to enumStruct for enum property p.

Start/Stop Reading/Writing functions

- MAPSIOElt ∗ StartWriting (MAPSOutput &output)

  The StartWriting function, one of the most important functions in the RTMaps component architecture.

- void StopWriting (MAPSIOElt ∗IOElt, bool discard=false)

  The StopWriting function, one of the most important functions in the RTMaps component architecture.

- MAPSIOElt ∗ StartReading (MAPSInput &input)

  The StartReading function, one of the most important functions in the RTMaps component architecture.

- MAPSIOElt ∗ StartReading (int nCount, MAPSInput ∗inputs[ ], int ∗inputThatAnswered, int nCountEvents=0, MAPSEvent ∗events[ ]=NULL)

  The multiple StartReading function, one of the most important functions in the RTMaps component architecture.

- MAPSIOElt ∗ MsgStartReading (MAPSInput &input)

  The MsgStartReading function.

- MAPSIOElt ∗ MsgStartReading (int nCount, MAPSInput ∗inputs[ ], int ∗inputThatAnswered, int nCountEvents=0, MAPSEvent ∗events[ ]=NULL)

  The multiple MsgStartReading function.

- MAPSTimestamp SynchroStartReading (int nb, MAPSInput ∗∗inputs, MAPSIOElt ∗∗IOElts, MAPSInt64 synchroTolerance=0, MAPSEvent ∗abortEvent=NULL)

  The synchronized StartReading function.

- bool NLastStartReading (MAPSInput &input, int nb, MAPSIOElt ∗∗IOElts)

  The N last MAPSIOElts StartReading function.

- void StopReading (MAPSInput &input)

  The StopReading function, one of the most important functions in the RTMaps component architecture.

Action execution functions

- void DoAction (MAPSAction &a)

  Executes action a.

- void ExecuteAction (MAPSAction &a)

  Executes action a.

- void DoAction (int nb)

  Executes action number nb.

- void DoAction (const char ∗name)

  Executes action called name.

- void ExecuteAction (int nb)

  Executes action a.

- void ExecuteAction (const char ∗name)

  Executes action a.

10.41.1 Detailed Description

The base class for all RTMaps modules. Defines a scriptable module with inputs, outputs, properties and actions.

10.41.2 Member Function Documentation

10.41.2.1 void MAPSModule::AcknowledgePropertyChanged (MAPSProperty & *p*) `[protected]`

Acknowledges the property value change.

Acknowledges the property value change.

Must be called regularly so that we can know the property has changed.

See also

MAPSProperty::PropertyChanged

10.41.2.2 void MAPSModule::AcknowledgePropertyChanged (int *propertynb*) `[protected]`

Acknowledges the property value change.

Acknowledges the property value change.

Must be called regularly so that we can know the property has changed.

See also

MAPSProperty::PropertyChanged

10.41.2.3 void MAPSModule::AcknowledgePropertyChanged (const char ∗ *name*) [protected]

Acknowledges the property value change.

Acknowledges the property value change.

Must be called regularly so that we can know the property has changed.

See also

MAPSProperty::PropertyChanged

10.41.2.4 void MAPSModule::Die ( ) [protected]

Function to be called by the module itself during termination. Ask all the associated threads to die.

The result is not immediate : some threads could refuse to die. Should be followed by a call to Wait4Death that waits for the death itself

10.41.2.5 void MAPSModule::DirectSet (MAPSProperty & *p*, MAPSInt32 *value*) [protected]

Sets integer *value* for integer property *p*.

10.41.2.6 void MAPSModule::DirectSet (MAPSProperty & *p*, const char ∗ *value*) [protected]

Sets string *value* for string property *p*.

10.41.2.7 void MAPSModule::DirectSet (MAPSProperty & *p*, const MAPSString & *enumValues*, MAPSInt32 *selected*) [protected]

Sets enum values to *enumValues* and the selected enum to *selected* for enum property *p*.

10.41.2.8 virtual void MAPSModule::Error (const char ∗ *string*, int *importance* = 1) [protected, virtual]

Emits an error.

This will kill definitely the module. Use MAPSModule::ReportError if you want to go on executing some code.

Parameters

string	Error string to log and display

importance	Importance of the error : • 0 : The error is minor. It will not have any impact on the overall system behaviour. Some actions have been taken that will fix the problem. • 1 : The error is important. Some actions have been taken that might fix the problem. The system can go on running, but it might not function properly. • 2 : The error is critical. The system will not function properly thereafter.

Reimplemented in MAPSRecordReplayMethod.

10.41.2.9 virtual void MAPSModule::Get (MAPSProperty & *p*, bool & *value*) [virtual]

Gets boolean *value* from boolean property *p*.

The first 4 Get/Set functions can be overloaded. The other functions always call these 4 basic functions.

10.41.2.10 bool MAPSModule::IsDying (void) [protected]

Returns `true` if the module is currently dying (but is not dead).

Some other component or the RTMaps kernel asked for its death.

10.41.2.11 bool MAPSModule::IsFIFOFull (MAPSOutput & *output*) [protected]

Returns `true` if the FIFO connected is full.

This information can be very useful. When you know the FIFO is full, you know that you don't process the data fast enough. You can then act and process the data faster (or skip some data in a smart way)

10.41.2.12 void MAPSModule::LoseOneLife (void) [protected]

Removes a "life" from the module. The "life" number is a counter for the threads associated to the module.

Do not use this function directly. Let the RTMaps kernel manage the life and death of module threads. Only for very experienced RTMaps developers.

10.41.2.13 MAPSIOElt∗ MAPSModule::MsgStartReading (MAPSInput & *input*)

The MsgStartReading function.

Returns `NULL` when a detachment or a message occured.

Warning

Detachment is not completly transparent to this version of StartReading since when there is a detachment, nothing happens, but the next answer can be `NULL` (that is, be a message) in that case, the last MAPSIOElt is no longer valid (there was an implicit StopReading due to the detachment)

Note

This function has a meaning only with message based OSes like Windows.

10.41.2.14 MAPSIOElt∗ MAPSModule::MsgStartReading (int *nCount*, MAPSInput ∗ *inputs[ ]*, int ∗ *inputThatAnswered*, int *nCountEvents* = `0`, MAPSEvent ∗ *events[ ]* = `NULL`)

The multiple MsgStartReading function.

The multiple StartReading returns `NULL` when a detachment or a message occured. In that case, *inputThatAnswered* indicates which input was detached, except if *inputThatAnswered* is equal to `MAPS::GotAMessage`, then the `NULL` answer corresponds to a Windows Message. Sequential components: Always returns `NULL` (should not be used)

Note

There is priority of inputs over events.
The order of priority among inputs is set by the inputs table *inputs*
This function has a meaning only with message based OSes like Windows.

10.41.2.15 int MAPSModule::MsgWait4Event (MAPSEvent ∗ *event*, MAPSDelay *timeout* = MAPS::Infinite) `[protected]`

Waits for an event to occur OR a message to come.

This function blocks until the event occurs OR a message comes to the thread. Note that this function has a meaning only for message based OS like Windows.

Parameters

event	Event to wait for
timeout	The wait aborts after *timeout* microseconds.

Returns

0 if the event occured (was set),MAPS::TimeOut if a timeout occured, MAPS::GotAMessage if a message arrived.

Note that using this function is safe. RTMaps knows how to unblock this function if the component dies, i.e. if the event never occurs, this function will not block your RTMaps system forever.

10.41.2.16 int MAPSModule::MsgWait4Events (int *nCount*, MAPSEvent ∗ *events[]*, MAPSDelay *timeout* = MAPS::Infinite, bool *addIsDyingEvent* = `true`) `[protected]`

Waits for some events to occur OR a message to come.

This function blocks until one of the events occurs OR a message comes to the thread. Note that this function has a meaning only for a message based OS like Windows.

Parameters

nCount	Number of events to wait on. It's the size of the events parameter array.
events	Array of pointers to MAPSEvent classes.
timeout	The wait aborts after *timeout* microseconds.
addIs-DyingEvent	

Returns

MAPS::TimeOut if a timeout occured, the index of the event that occured (was set) and led to the function return, MAPS::GotAMessage if a message arrived.

Note that using this function is safe. RTMaps knows how to unblock this function if the component dies, i.e. if no event ever occurs, this function will not block your RTMaps system forever.

10.41.2.17 int MAPSModule::NbProperties (void) `[protected]`

Returns the number of properties of the module.

The number of properties may be higher that expected due to the fact that this function takes into account the module inputs and outputs properties (such as `subsampling`, `fifosize` etc. and also some fixed number of "hidden" properties that are reserved for internal use. In order to retrieve the number of "user" properties, you might use the `MAPSModule::firstUserPropertyIndex` member like: `NbProperties() - firstUserPropertyIndex`

10.41.2.18 bool MAPSModule::NLastStartReading (MAPSInput & *input*, int *nb*, MAPSIOElt ∗∗ *IOElts*)

The N last MAPSIOElts StartReading function.

Returns `true` when all the MAPSIOElts are correctly fetched. In threaded mode, the function should always return `true`.

Returns `false` in sequential mode until the right number of elements are fetched, or when a dynamic detachment occurs.

10.41.2.19 void MAPSModule::OneMoreLife (void) `[protected]`

Adds a "life" to the module. The "life" number is a counter for the threads associated to the module.

Do not use this function directly. Let the RTMaps kernel manage the life and death of module threads. Only for very experienced RTMaps developers.

10.41.2.20 virtual void MAPSModule::PerformanceMonitoring ( ) `[protected, virtual]`

Performance monitoring function.

This function is called automatically by the RTMaps kernel along with the Core function of the component.

10.41.2.21 bool MAPSModule::PropertyChanged (MAPSProperty & *p*) `[protected]`

Tells if the property value has changed.

Tells if the property value has changed.

Returns `true` if the property value has changed since the last call to MAPSProperty::AcknowledgePropertyChanged

10.41.2.22 bool MAPSModule::PropertyChanged (int *propertynb*) `[protected]`

Tells if the property value has changed.

Tells if the property value has changed.

Returns `true` if the property value has changed since the last call to MAPSProperty::AcknowledgePropertyChanged

10.41.2.23 bool MAPSModule::PropertyChanged (const char ∗ *name*) `[protected]`

Tells if the property value has changed.

Tells if the property value has changed.

Returns `true` if the property value has changed since the last call to MAPSProperty::AcknowledgePropertyChanged

10.41.2.24 virtual void MAPSModule::Report (const char ∗ *string*, int *type* = MAPS::Info, int *importance* = `0`) `[protected, virtual]`

General report function.

Parameters

string	String to log and display
type	Type of report. Can be MAPS::Info, MAPS::Warning or MAPS::Error.
importance	Importance of the report (0: minor, 1: important, 2: critical)

10.41.2.25 virtual void MAPSModule::ReportError (const char ∗ *string*, int *importance* = 0) [protected, virtual]

Reports an error.

Parameters

string	Error string to log and display
importance	Importance of the error : • 0 : The error is minor. It will not have any impact on the overall system behaviour. Some actions have been taken that will fix the problem. • 1 : The error is important. Some actions have been taken that might fix the problem. The system can go on running, but it might not function properly. • 2 : The error is critical. The system will not function properly thereafter.

10.41.2.26 virtual void MAPSModule::ReportInfo (const char ∗ *string*, int *importance* = 0) [protected, virtual]

Reports a piece of information.

Parameters

string	Info string to log and display
importance	Importance of the information : • 0 : The information is minor. • 1 : The information is important. • 2 : The information is critical.

10.41.2.27 virtual void MAPSModule::ReportWarning (const char ∗ *string*, int *importance* = 0) [protected, virtual]

Reports a warning.

Parameters

string	Warning string to log and display

importance	Importance of the warning : • 0 : The warning is minor. It has no impact on the behaviour of the module. • 1 : The warning is important. It signals a malfunction in the module that could result in an error. • 2 : The warning is critical. It signals a major malfunction in the module. It will certainly result in an error.

10.41.2.28 bool MAPSModule::Rest (MAPSDelay *delay,* MAPSEvent ∗ *cancel* = `NULL`) `[protected]`

Blocks the current thread for a delay of *delay* microseconds.

The parameters of these functions are expressed in virtual microseconds. This means that these functions make full usage of the RTMaps virtual time management feature. You MUST use these functions when you design some time-related components.

Any absolute reference is lost with this function. It works perfectly when the the virtual time speed is negative.

The optional parameter *cancel* allows to asynchronously stop the wait.

Returns `true` when the delay has elapsed. Returns `false` if the wait was canceled by setting the *cancel* event.

Note that using this function is safe. RTMaps knows how to unblock this function if the component dies, i.e. this function will not block your RTMaps system until the delay expires, which can take some hours...

10.41.2.29 void MAPSModule::Set (MAPSProperty & *p,* MAPSInt32 *value*)

Sets integer *value* from integer property *p*.

10.41.2.30 void MAPSModule::Set (MAPSProperty & *p,* const char ∗ *enumValues,* MAPSInt32 *selected*)

Sets enum values to *enumValues* and the selected enum to *selected* for enum property *p*.

10.41.2.31 void MAPSModule::Set (MAPSProperty & *p,* const char ∗ *value*)

Sets string *value* from string property *p*.

10.41.2.32 MAPSIOElt∗ MAPSModule::StartReading (MAPSInput & *input*)

The StartReading function, one of the most important functions in the RTMaps component architecture.

Sequential components: returns `NULL` when no input is connected or when no data is directly available, or when a detachment was asked for.

Note

Dynamic detachment is transparent to this version of StartReading in threading mode, but not in sequential mode. Watch out for `NULL` return values when developping sequential mode compatible components.

10.41.2.33 MAPSIOElt∗ MAPSModule::StartReading (int *nCount,* MAPSInput ∗ *inputs[],* int ∗ *inputThatAnswered,* int *nCountEvents =* `0`*,* MAPSEvent ∗ *events[] =* `NULL`)

The multiple StartReading function, one of the most important functions in the RTMaps component architecture.

The multiple StartReading returns `NULL` when a detachment occured. In that case, *inputThatAnswered* indicates which input was detached.

This version of StartReading enables to simultaneously wait for data on inputs and wait for events, exactly like MAPSModule::Wait4Event does. If some events are specified, and one of these events is triggered, the function returns *NULL*. In order to discrimate between a detachment and an occured event, the user MUST test the *inputThatAnswered* variable. If *inputThatAnswered* is more than *nCount*, then obviously an event occured, and its number is given by `inputThatAnswered-ncount`, starting from 0.

Note

There is priority of inputs over events.
The order of priority among inputs is set by the inputs table *inputs*

10.41.2.34 void MAPSModule::StopWriting (MAPSIOElt ∗ *IOElt,* bool *discard =* `false`)

The StopWriting function, one of the most important functions in the RTMaps component architecture.

Parameters

IOElt	A pointer to the MAPSIOElt obtained by a previous call to StartWriting.
discard	If set to `true`, the write will not be done, that is the data will be lost.

10.41.2.35 MAPSTimestamp MAPSModule::SynchroStartReading (int *nb*, MAPSInput ∗∗ *inputs*, MAPSIOElt ∗∗ *IOElts*, MAPSInt64 *synchroTolerance* = `0`, MAPSEvent ∗ *abortEvent* = `NULL`)

The synchronized StartReading function.

The SynchroStartReading function can retrieve synchronized data on several different inputs. The function returns when input data with the same timestamp are found on the inputs (or approximately the same in case the parameter *synchroTolerance* is specified and more than 0). The corresponding MAPSIOElts are then returned via the provided *IOElts* array.

An *abortEvent* can be provided in order to force the function to return when the event is set. In this case, the returned timestamp is -1 and the IOElts in the array are set to `NULL`. This *abortEvent* can be used for example in order to update the *synchro-Tolerance* if the latest value has been too low and the function blocks since it cannot synchronize signals.

10.41.2.36 bool MAPSModule::Wait (MAPSTimestamp *timestamp*, MAPSEvent ∗ *cancel* = `NULL`) `[protected]`

Blocks the current thread until the time reaches *timestamp*.

Beware : the time might never reach *timestamp*, especially if the virtual time speed is negative (the time goes backward).

The optional parameter *cancel* allows to asynchronously stop the wait.

Returns `true` when the timestamp *timestamp* is reached. Returns `false` is the wait was canceled by setting the *cancel* event.

Note that using this function is safe. RTMaps knows how to unblock this function if the component dies, i.e. this function will not block your RTMaps system until *timestamp* is reached, which might never happen.

10.41.2.37 bool MAPSModule::Wait4Event (MAPSEvent ∗ *event*, MAPSDelay *timeout* = MAPS::Infinite) `[protected]`

Waits for an event to occur.

This function blocks until the event occurs.

Parameters

event	Event to wait for
timeout	The wait aborts after *timeout* microseconds.

Returns

`true` if the event occurred (was set), `false` if a timeout occured.

Note that using this function is safe. RTMaps knows how to unblock this function if the component dies, i.e. if the event never occurs, this function will not block your RTMaps system forever.

10.41.2.38 bool MAPSModule::Wait4Event (MAPSEvent & *event*, MAPSDelay *timeout* = MAPS::Infinite) `[protected]`

Waits for an event to occur.

This function blocks until the event occurs.

Parameters

event	Event to wait for
timeout	The wait aborts after *timeout* microseconds.

Returns

`true` if the event occurred (was set), `false` if a timeout occured.

Note that using this function is safe. RTMaps knows how to unblock this function if the component dies, i.e. if the event never occurs, this function will not block your RTMaps system forever.

10.41.2.39 int MAPSModule::Wait4Events (int *nCount*, MAPSEvent ∗ *events[ ]*, MAPSDelay *timeout* = MAPS::Infinite, bool *addIsDyingEvent* = `true`) `[protected]`

Waits for some events to occur.

This function blocks until one of the events occurs.

Parameters

nCount	Number of events to wait on. It is the size of the event parameter array.
events	Array of pointers to MAPSEvent instances.
timeout	The wait aborts after *timeout* microseconds.
addIs-DyingEvent	

Returns

MAPS::TimeOut if a timeout occured, the index of the event that occured (was set) and led to the function return.

Note that using this function is safe. RTMaps knows how to unblock this function if the component dies, i.e. if no event ever occurs, this function will not block your RTMaps system forever.

10.41.2.40 void MAPSModule::Wait4Handshake (MAPSOutput & *output*) `[protected]`

Waits for a handshake on output.

This function is useful for handshaking communication between components. It will block until one of the components connected on *output* have read the data recently output by MAPSModule::StopWriting

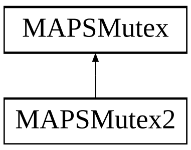
The documentation for this class was generated from the following file:

- maps.hpp

10.42 MAPSMutex Class Reference

The RTMaps mutex class.

Inheritance diagram for MAPSMutex:



Public Member Functions

- MAPSMutex ()

  Standard constructor.

- virtual ∼MAPSMutex ()

  Standard destructor.

- void Lock ()

  Locks the mutex.

- void Release ()

  Releases the mutex.

- void Reset ()

  Tries to lock the mutex.

10.42.1 Detailed Description

The RTMaps mutex class. A mutex blocks concurrent threads if they both want to access shared data at the same time.

10.42.2 Member Function Documentation

10.42.2.1 void MAPSMutex::Lock ( )

Locks the mutex.

Any other thread will block until this thread calls MAPSMutex::Release

Reimplemented in MAPSMutex2.

10.42.2.2 void MAPSMutex::Release ( )

Releases the mutex.

Any other thread blocked on this monitor will be able to lock it.

Reimplemented in MAPSMutex2.

10.42.2.3 void MAPSMutex::Reset (void)

Tries to lock the mutex.

Resets the mutex

Reset is very important to avoid deadlocking : when a mutex is possibly still owned by a dead thread, then a call to reset in MAPSComponent::Death() will solve the problem. Not calling Reset() in that case will probably deadlock the component during its next start, on the first call to Lock().

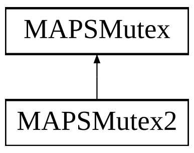
The documentation for this class was generated from the following file:

- maps.hpp

10.43 MAPSMutex2 Class Reference

The RTMaps advanced mutex class.

Inheritance diagram for MAPSMutex2:



Public Member Functions

- MAPSMutex2 ()

  Standard constructor. No options.

- void Lock ()

  Locks the mutex.

- bool TryLock ()

  Tries to lock the mutex. If it is already locked by somebody else, returns `false` immediatly.

- void ReleaseAll ()

  Releases all references on the mutex (if any)

- void Release ()

  Releases the mutex.

10.43.1 Detailed Description

The RTMaps advanced mutex class. This mutex has a reference counter and knows which thread owns it. Therefore, a thread can lock it several times without deadlocking it!

10.43.2 Member Function Documentation

10.43.2.1 void MAPSMutex2::ReleaseAll ( )

Releases all references on the mutex (if any)

Very useful when an exception occured and we are not sure about the number of references really held by the thread.

The documentation for this class was generated from the following file:

- maps.hpp

10.44 MAPSOutput Class Reference

The RTMaps Module Output class.

Public Member Functions

- virtual MAPSString & Name ()=0

  Returns the name of the output.

- virtual int NbFIFOOverflows ()=0

  Returns the number of FIFO overflows on this output (performance monitoring)

- virtual MAPSIOMonitor & Monitor ()=0

  Returns a pointer to the MAPSIOMonitor object associated to this output (the buffer management system)

- virtual void MakeShareable ()=0

  Makes this output buffer shareable among different RTMaps processes (expert only)

- virtual void MakeUnshareable ()=0

  Makes this output buffer unshareable among different RTMaps processes (expert only)

- virtual int IOEltSizeBytes ()=0

  Returns the size in bytes of one element of the buffer according to the allocation.

- virtual MAPSTypeInfo & Type ()=0

  Returns the type of the output.

- virtual void SetTypeUnit (const char ∗unit)=0

  Sets the unit of the output (dynamically modifies the type)

- virtual void SetUnit (const char ∗unit)=0

  Sets the unit of the output (dynamically modifies the type)

- virtual const char ∗ TypeUnit ()=0

  Gets the unit of the output ("m/s" for instance)

- virtual const char ∗ Unit ()=0

  Gets the unit of the output ("m/s" for instance)

- virtual void SetTypeName (const char ∗name)=0

  Sets the name parameter of the type of the output (dynamically modifies the type)

- virtual const char ∗ TypeName ()=0

  Gets the name parameter of the type of the output.

- virtual ∼MAPSOutput ()

  Destructor.

Buffer allocation functions

These are very important functions.

Note

There is no deallocation functions, since RTMaps automatically deallocates the output buffer when the output dies.

- virtual void AllocOutputBuffer (int size, bool doAllocate=true)=0

  Allocates an output buffer with an element size of size.

- virtual void AllocOutputBufferIplImage (IplImage &model, bool headerOnly=false)=0

  Allocates an IplImage output buffer.

- virtual void AllocOutputBufferImage (int size, int w=0, int h=0, MAPSUInt32 coding=MAPS_-IMAGECODING_UNKNOWN, MAPSUInt32 imageType=MAPS_IMAGETYPE_COLOR, bool headerOnly=false)=0

  Allocates a MAPSImage output buffer.

- virtual void AllocOutputBufferMAPSImage (int size, int w=0, int h=0, MAPSUInt32 coding=MAPS_IMAGECODING_UNKNOWN, MAPSUInt32 imageType=MAPS_-IMAGETYPE_COLOR, bool headerOnly=false)=0

  Allocates a MAPSImage output buffer.

- virtual void AllocOutputBufferMatrix (int m, int n)=0

  MAPSMatrix output buffer allocation.

10.44.1 Detailed Description

The RTMaps Module Output class. You must use this class in particular to allocate the data buffers in RTMaps. They are called "output buffers", because they are associated to the output. In the RTMaps component model, the outputs are responsible for the data buffering, and the components are responsible for their outputs.

10.44.2 Member Function Documentation

10.44.2.1 virtual void MAPSOutput::AllocOutputBuffer (int *size*, bool *doAllocate* = `true`) `[pure virtual]`

Allocates an output buffer with an element size of *size*.

This function is automatically called when an output is created, but it provides only basic output buffer capabilities, since it has only one parameter.

What this function really does depends on the type of the output:

- For "vectorizable" types, like integers, floats or CAN frames, this defines the size of the vector (*size* must be 1 for a scalar).

- For text types (MAPS::TextAscii or MAPS::TextUnicode), this function allocates some strings for the buffer of *size* characters. The default value MAPS::DefaultTextSize is generally used.

- For the MAPSMatrix type, an array of *size* elements is allocated for each buffer element, which means that the maximum $m*n$ size of the matrix must be less than *size*. *m* and *n* are set to 0 by default. A call to MAPSOutput::AllocOutputBufferMatrix can be more appropriate.
- For a MAPSImage type, an image buffer of *size* is allocated for each buffer element. All the parameters of the structure are set to 0 or unknown. A call to MAPSOutput::AllocOutputBufferImage or MAPSOutput::AllocOutputBufferMAPSImage might be more appropriate.
- For an IplImage type, no allocation is done whatever *size* is. You MUST call MAPSOutput::AllocOutputBufferIplImage to allocate the buffer.

If *size* is 0, no output buffer allocation is done, and another call to an output buffer allocation function will have to be done before the buffer can be actually used.

10.44.2.2 virtual void MAPSOutput::AllocOutputBufferImage (int *size*, int *w* = `0`, int *h* = `0`, MAPSUInt32 *coding* = `MAPS_IMAGECODING_UNKNOWN`, MAPSUInt32 *imageType* = `MAPS_IMAGETYPE_COLOR`, bool *headerOnly* = `false`) `[pure virtual]`

Allocates a MAPSImage output buffer.

Parameters

size	The size in bytes to allocate for the image buffer. Note that it can be less than $w*h$ since a MAPSImage can contain compressed images.
w	The width of the image
h	The height of the image
coding	Integer describing the code used by the image buffer. For instance `MAPS_IMAGECODING_JPEG` or `MAPS_IMAGECODING_YUYV`.
imageType	Additional information describing the type of the image. Generally, it is set to MAPS_IMAGETYPE_COLOR or MAPS_IMAGETYPE_MONO.
headerOnly	If set to `true`, this function will only allocate the header, and no memory will be allocated for the content of the image. This is useful to directly connect the IplImage structure to a frame buffer allocated by a hardware frame grabber. For experts only. *headerOnly* should usually be set to `false`.

10.44.2.3 virtual void MAPSOutput::AllocOutputBufferIplImage (IplImage & *model*, bool *headerOnly* = `false`) `[pure virtual]`

Allocates an IplImage output buffer.

Parameters

model	A model to replicate for each buffer element. This model can be provided by a call to MAPS::IplImageModel.

headerOnly	If set to `true`, this function will only allocate the header, and no memory will be allocated for the content of the image. This is useful to directly connect the IplImage structure to a frame buffer allocated by a hardware frame grabber. For experts only. headerOnly should usually be set to `false`.

10.44.2.4 virtual void MAPSOutput::AllocOutputBufferMAPSImage (int *size*, int *w* = `0`, int *h* = `0`, MAPSUInt32 *coding* = `MAPS_IMAGECODING_UNKNOWN`, MAPSUInt32 *imageType* = `MAPS_IMAGETYPE_COLOR`, bool *headerOnly* = `false`) `[pure virtual]`

Allocates a MAPSImage output buffer.

Allocates a MAPSImage output buffer.

Parameters

size	The size in bytes to allocate for the image buffer. Note that it can be less than *w*∗*h* since a MAPSImage can contain compressed images.
w	The width of the image
h	The height of the image
coding	Integer describing the code used by the image buffer. For instance `MAPS_IMAGECODING_JPEG` or `MAPS_IMAGECODING_YUYV`.
imageType	Additional information describing the type of the image. Generally, it is set to MAPS_IMAGETYPE_COLOR or MAPS_IMAGETYPE_MONO.
headerOnly	If set to `true`, this function will only allocate the header, and no memory will be allocated for the content of the image. This is useful to directly connect the IplImage structure to a frame buffer allocated by a hardware frame grabber. For experts only. *headerOnly* should usually be set to `false`.

10.44.2.5 virtual void MAPSOutput::AllocOutputBufferMatrix (int *m*, int *n*) `[pure virtual]`

MAPSMatrix output buffer allocation.

Parameters

m	The number of rows of the allocated matrix
n	The number of columns of the allocated matrix

10.44.2.6 virtual int MAPSOutput::IOEltSizeBytes ( ) `[pure virtual]`

Returns the size in bytes of one element of the buffer according to the allocation.

This is very useful to copy a buffer element. Beware that MAPS::Memcpy() cannot generally be used to copy buffers, in particular for IplImage and MAPSImage

10.44.2.7 virtual MAPSString& MAPSOutput::Name ( ) `[pure virtual]`

Returns the name of the output.

In the form `componentName.outputName`

10.44.2.8 virtual void MAPSOutput::SetTypeName (const char ∗ *name*) `[pure virtual]`

Sets the name parameter of the type of the output (dynamically modifies the type)

See also

MAPSTypeInfo

10.44.2.9 virtual void MAPSOutput::SetTypeUnit (const char ∗ *unit*) `[pure virtual]`

Sets the unit of the output (dynamically modifies the type)

See also

MAPSTypeInfo

10.44.2.10 virtual void MAPSOutput::SetUnit (const char ∗ *unit*) `[pure virtual]`

Sets the unit of the output (dynamically modifies the type)

See also

MAPSTypeInfo

10.44.2.11 virtual const char∗ MAPSOutput::TypeName ( ) `[pure virtual]`

Gets the name parameter of the type of the output.

This is completely different from the name of the output itself. Don't get confused.

See also

MAPSTypeInfo

The documentation for this class was generated from the following file:

- maps.hpp

10.45 MAPSPair< TKey, TContent > Class Template Reference

The RTMaps (key,content) pair template class.

Public Member Functions

- TKey & Key ()

  Returns the key stored in the pair.

- TContent & Content ()

  Returns the content stored in the pair.

- MAPSPair (TKey &kx, TContent &cx)

  Constructor.

- MAPSPair ()

  Default constructor.

10.45.1 Detailed Description

template<typename TKey, typename TContent> class MAPSPair< TKey, TContent >

The RTMaps (key,content) pair template class.

See also

MAPSHashTable

The documentation for this class was generated from the following file:

- maps.hpp

10.46 MAPSProperty Class Reference

The RTMaps Module Property class.

Public Member Functions

- virtual MAPSInt64 & IntegerValue ()=0

  Returns a reference to the integer content of the property.

- virtual MAPSString & StringValue ()=0

  Returns a reference to the string content of the property.

- virtual bool & BoolValue ()=0

  Returns a reference to the boolean content of the property.

- virtual MAPSFloat & FloatValue ()=0

  Returns a reference to the float content of the property.

- virtual MAPSEnumStruct & EnumValues ()=0

  Returns a reference to the enum structure of the property.

- virtual MAPSString & Name ()=0

  Returns the name of the property.

- virtual MAPSString & ShortName ()=0

  Returns the short name of the property.

- virtual const char ∗ Unit ()=0

  Returns the unit of the property.

- virtual MAPSModule & Owner ()=0

  Returns the owner of this property.

- virtual int Type ()=0

  Returns the type of this property.

- virtual bool PropertyChanged ()=0

  Tells if the property value has changed.

- virtual void AcknowledgePropertyChanged ()=0

  Acknowledges the property value change.

- virtual MAPSEvent ∗ GetPropertyChangedEvent ()=0

  Gets the MAPSEvent related to the change of property value event.

- virtual MAPSEvent & PropertyChangedEvent ()=0

  Gets the MAPSEvent related to the change of property value event.

The Set functions (to set the property value)

These functions call the MAPSModule::Set functions associated to the owner of the property. Behaviour might differ depending on the type of the property.

- virtual void Set (bool value)=0

  Sets the boolean property value to value for boolean properties.

- virtual void Set (MAPSInt64 value)=0

  Sets the integer property value to value for integer properties, or the selected item for enum properties.

- virtual void Set (MAPSInt32 value)=0
- virtual void Set (MAPSFloat value)=0

  Sets the float property value to value for floating point properties.

- virtual void Set (const char ∗value)=0

  Sets string property value to value for string properties, or fills the enum values for enum properties.

- virtual void Set (const MAPSString &value)=0
- virtual void Set (MAPSEnumStruct enumStruct)=0

  Sets the enum property value to enumStruct (enum strings and the selected item) for enum properties.

- virtual void Set (const MAPSString &enumValues, const int selected)=0

  Sets enum values to enumValues and the selected enum value to selected for enum properties.

The Get functions (to get the property value)

These functions call the MAPSModule::Get functions associated to the owner of the property. Behaviour might differ depending on the type of the property.

- virtual void Get (bool &value)=0

  Gets the boolean property value into value for boolean properties.

- virtual void Get (MAPSInt64 &value)=0

  Gets the integer property value into value for integer properties, or the selected item for enum properties.

- virtual void Get (MAPSInt32 &value)=0
- virtual void Get (MAPSFloat &value)=0

  Gets the float property value into value for float properties.

- virtual void Get (MAPSString &value)=0

  Gets the string property value int value for string properties, or the selected item for enum properties.

- virtual void Get (MAPSEnumStruct &enumStruct)=0

  Gets the enum property value into enumStruct for enum properies.

- virtual void Get (MAPSArray< MAPSString > &enumValues, MAPSInt32 &selectedValue)=0

Gets the enum property value for enum properties.

Properties callback management functions

For RTMaps experts only!

- virtual void SetSetCallback (MAPSPropertySetCallback cb)=0

  Sets a callback which will be called on a call to Set.

- virtual void SetGetCallback (MAPSPropertyGetCallback cb)=0

  Sets a callback which will be called on a call to Get.

10.46.1 Detailed Description

The RTMaps Module Property class.

10.46.2 Member Function Documentation

10.46.2.1 virtual void MAPSProperty::AcknowledgePropertyChanged ( ) [pure virtual]

Acknowledges the property value change.

Must be called regularly so that we can know the property has changed.

See also

MAPSProperty::PropertyChanged

10.46.2.2 virtual void MAPSProperty::Get (MAPSInt32 & *value*) [pure virtual]

Gets the integer property value into *value* for integer properties, or the selected item for enum properties.

10.46.2.3 virtual void MAPSProperty::Get (MAPSArray< MAPSString > & *enumValues,* MAPSInt32 & *selectedValue*) [pure virtual]

Gets the enum property value for enum properties.

The enum value is split into the array of strings, put into *enumValues*, and the selected value, put into *selectedValue*

10.46.2.4 virtual MAPSEvent∗ MAPSProperty::GetPropertyChangedEvent ( ) `[pure virtual]`

Gets the MAPSEvent related to the change of property value event.

This can be very useful to make some wait on property value change. You can do that with the MAPSModule::Wait4Event or MAPSModule::Wait4Events function.

10.46.2.5 virtual MAPSString& MAPSProperty::Name ( ) `[pure virtual]`

Returns the name of the property.

In the form `componentName.propertyName`

10.46.2.6 virtual bool MAPSProperty::PropertyChanged ( ) `[pure virtual]`

Tells if the property value has changed.

Returns `true` if the property value has changed since the last call to MAPSProperty::AcknowledgePropertyChanged

10.46.2.7 virtual MAPSEvent& MAPSProperty::PropertyChangedEvent ( ) `[pure virtual]`

Gets the MAPSEvent related to the change of property value event.

This can be very useful to make some wait on property value change. You can do that with the MAPSModule::Wait4Event or MAPSModule::Wait4Events function.

10.46.2.8 virtual void MAPSProperty::Set (MAPSInt32 *value*) `[pure virtual]`

Sets the integer property value to *value* for integer properties, or the selected item for enum properties.

10.46.2.9 virtual void MAPSProperty::Set (const MAPSString & *value*) `[pure virtual]`

Sets string property value to *value* for string properties, or fills the enum values for enum properties.

10.46.2.10 virtual MAPSString& MAPSProperty::ShortName ( ) `[pure virtual]`

Returns the short name of the property.

Returns the part of the name after the first dot (module name)

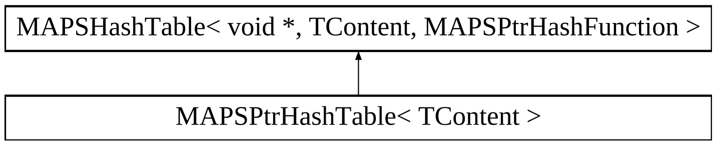
The documentation for this class was generated from the following file:

- maps.hpp

10.47 MAPSPtrHashTable< TContent > Class Template Reference

The RTMaps pointer hash table template class.

Inheritance diagram for MAPSPtrHashTable< TContent >:


MAPSHashTable< void *, TContent, MAPSPtrHashFunction >
MAPSPtrHashTable< TContent >

Public Member Functions

- MAPSPtrHashTable (int n=16)

    Constructor.

10.47.1 Detailed Description

template<typename TContent> class MAPSPtrHashTable< TContent >

The RTMaps pointer hash table template class. As its name says, the key is a pointer

The documentation for this class was generated from the following file:

- maps.hpp

10.48 MAPSRBRegion Class Reference

MAPSRBRegion class : Ring-Buffer region manipulation class.

Public Member Functions

- MAPSConstRBRegionState Check () const

    Check that the region is consistent and returns its state.

- int operator[] (int i) const

    A very important operator when manipulating regions.

Static Public Member Functions

- static const MAPSString StateToString (const MAPSConstRBRegionState &st)

  A very useful function to convert state into strings.

Public Attributes

- int fifoSize

  define the size of the ring-buffer.

- int offset

  define the start of the region in the ring-buffer.

- int size

  define the size of the region.

Constructors

- MAPSRBRegion ()

  Constructor of the class : just initialize all attributes to 0, and put the region to the MAPSConstRBRegionUninitialized state.

- MAPSRBRegion (const MAPSRBRegion &reg)

  Copy constructor of the class.

Initialization routines

- MAPSConstRBRegionState Init (int fifoSizex, int offsetx, int sizex)

  Initialize the region with default values.

- MAPSConstRBRegionState Reset ()

  Reset the region : offset=0, size=0.

- MAPSConstRBRegionState Full ()

  Fill entirely the region from current offset.

Attribute set methods.

- MAPSConstRBRegionState SetFifoSize (int value)

  Set the ring-buffer size of the current region.

- MAPSConstRBRegionState SetSize (int value)

  Set the size attribute.

- MAPSConstRBRegionState SetOffset (int value)

  Set the offset attribute.

Attributes or simple get methods

- int InverseSize () const

  Returns the complementary size of the region (fifoSize-size)

- int FifoSize () const

  Returns the size of the ring-buffer.

- int Size () const

  Get size attribute.

- int Size0 () const

  Get size attribute.

- int Size1 () const

  Get size attribute.

- int Offset () const

  Get offset attribute.

- int OffsetLastElement () const

  Get the last offset of the region.

Operation on the region itself

- MAPSConstRBRegionState Erode (MAPSConstRBRegionErosion type, int sizex)

  Erosion of a region.

- MAPSConstRBRegionState Dilate (MAPSConstRBRegionDilation type, int sizex)

  Dilation of a region.

- MAPSConstRBRegionState Crop (MAPSConstRBRegionCrop type, int offsetX, int sizex)

  Crop a region.

Transformation of the region into a new one

- MAPSConstRBRegionState SubRegion (MAPSRBRegion &dest, MAPSConstRBRegionSubRegion type, int offsetX, int sizex) const

  Extract a sub-region from current region.

- MAPSConstRBRegionState BesideRegion (MAPSRBRegion &dest, MAPSConstRBRegionBesideRegion type, int offsetX, int sizex) const

  Define a new region next to current region.

- MAPSConstRBRegionState InverseRegion (MAPSRBRegion &dest) const

  Define the complementary region of the current region (inversion).

Direct-access operations : Push, Pop, etc...

- MAPSConstRBRegionState Push (int nbData, MAPSRBRegion &destSubregion, bool enableOverwrite=true)

  Push some data in the ring-buffer (FIFO like operation)

- MAPSConstRBRegionState Pop (int nbData, MAPSRBRegion ∗srcSubregion=NULL)

  Pop some data from the ring-buffer (FIFO like operation)

- MAPSConstRBRegionState Popback (int nbData, MAPSRBRegion ∗srcSubregion=NULL)

  Dequeues some data from the ring-buffer (STACK like operation). The data is removed from the last.

Templates

- template<class T >
  int CopyFromRB (const T ∗fifoData, T ∗dest, int destMaxSize) const

  A template function designed to copy the content of a circular buffer to a temporary linear buffer.

- template<class T >
  int CopyToRB (T ∗fifoData, const T ∗src, int srcMaxSize)

  A template function designed to copy the content of a temporary linear buffer to a circular buffer region.

- template<class T >
  int DuplicateToRB (T ∗fifoData, T data)

  A template function designed to duplicate an element t of T into the whole region of dest.

- template<class T >
  int DuplicateToRB2 (T ∗fifoData, T data, T increment)

  A template function designed to duplicate an element t of T into the whole region of dest with regular increments.

- template<class T >
  int DuplicateToRB3 (T ∗fifoData, T data, T increment, T min_data_value)

  A template function designed to duplicate an element t of T into the whole region of dest with regular increments but with a minimum value.

10.48.1 Detailed Description

MAPSRBRegion class : Ring-Buffer region manipulation class. This class allows the user to manipulate regions of ring-buffers.

A region is built around the following variables :

- offset : define the start of the region in the ring-buffer.
- size : define the size of the region. size=size0+size1
- size0 : define the size of the region up to the end of the ring-buffer.
- size1 : define the size of the region from the beginning of the ring-buffer.
- fifoSize : define the size of the ring-buffer.

```
.                             fifoSize .
.  x-------------------------------------------------------------x .
. |++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
.
. |*********                 **************************************|
.
. |++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++|
.
.  x-------x                 x-----------------------------------x .
.    size1                   |                size0 .
.                          offset .
```

10.48.2 Member Function Documentation

10.48.2.1 MAPSConstRBRegionState MAPSRBRegion::BesideRegion (MAPSRBRegion & *dest*, MAPSConstRBRegionBesideRegion *type*, int *offsetX*, int *sizex*) const

Define a new region next to current region.

Parameters

dest	The destination region.
type	Define if the region must be taken from before or after the src region.
offsetX	Define an offset from the starting point (beginning or end).
sizex	Size of the new region.

Returns

The return value is the state of the new region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`

10.48.2.2 MAPSConstRBRegionState MAPSRBRegion::Crop (MAPSConstRBRegionCrop *type*, int *offsetX*, int *sizex*)

Crop a region.

Parameters

type	Define if the region must be croped from its beginning or its end
offsetX	Define an offset from the starting point (beginning or end).
sizex	Size of the new region (cropped).

Returns

The return value is the state of the new region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`

10.48.2.3 MAPSConstRBRegionState MAPSRBRegion::Dilate (MAPSConstRBRegionDilation *type*, int *sizex*)

Dilation of a region.

Parameters

type	Define if the region must be dilated from its beginning and/or its end
sizex	Size of the dilation element.

Returns

The return value is the state of the region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`

10.48.2.4 template$<$class T $>$ int MAPSRBRegion::DuplicateToRB2 (T $*$ *fifoData*, T *data*, T *increment*)

A template function designed to duplicate an element t of T into the whole region of dest with regular increments.

Usually used for timestamp data with frequency information (so we duplicate first the original timestamp, then increment it each step).

10.48.2.5 template<class T > int MAPSRBRegion::DuplicateToRB3 (T ∗ *fifoData*, T *data*, T *increment*, T *min_data_value*)

A template function designed to duplicate an element t of T into the whole region of dest with regular increments but with a minimum value.

Usually used for timestamp data with frequency information and monotonous constraints The min_data_value parameter should then be the last timestamp duplicated to the RB at the previous call to DuplicateToRB3. So we duplicate first the original timestamp, then increment it each step, provided it is not lower that min_data_value. Otherwise, the current value is bounded to min_data_value. WARNING: when using with timestamp, this can be nice for real-time operation, but problematic in replay mode when jumping backwards in time.

10.48.2.6 MAPSConstRBRegionState MAPSRBRegion::Erode (MAPSConstRBRegionErosion *type*, int *sizex*)

Erosion of a region.

Parameters

type	Define if the region must be eroded from its beginning and/or its end
sizex	Size of the erosion element.

Returns

The return value is the state of the region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`

10.48.2.7 MAPSConstRBRegionState MAPSRBRegion::Init (int *fifoSizex*, int *offsetx*, int *sizex*)

Initialize the region with default values.

Parameters

fifoSizex	The size of the ring-buffer
offsetx	The start of the region in the ring-buffer.
sizex	The region size

Returns

Returns the state of the defined region.

10.48.2.8 MAPSConstRBRegionState MAPSRBRegion::InverseRegion (MAPSRBRegion & *dest*) const

Define the complementary region of the current region (inversion).

Parameters

dest	The destination region.

Returns

The return value is the state of the new region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`

10.48.2.9 int MAPSRBRegion::operator[] (int *i*) const

A very important operator when manipulating regions.

This operator returns the offset `i` of a given region. If `i` is an incorrect offset and in debug mode, this method generate an `assert`. Otherwise, it returns the direct offset in the ring-buffer of the `i` element of the current region.

10.48.2.10 MAPSConstRBRegionState MAPSRBRegion::Pop (int *nbData*, MAPSRBRegion ∗ *srcSubregion* = `NULL`)

Pop some data from the ring-buffer (FIFO like operation)

Parameters

nbData	The number of element your want to pop from the FIFO
srcSubregion	Will contain the sub-region that contains data to pop.

Returns

The return value is the state of the new region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`.

10.48.2.11 MAPSConstRBRegionState MAPSRBRegion::Popback (int *nbData*, MAPSRBRegion ∗ *srcSubregion* = `NULL`)

Dequeues some data from the ring-buffer (STACK like operation). The data is removed from the last.

Parameters

nbData	The number of element your want to popback from the stack
srcSubregion	Will contain the sub-region that contains data to popback.

Returns

The return value is the state of the new region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`.

10.48.2.12 MAPSConstRBRegionState MAPSRBRegion::Push (int *nbData*, MAPSRBRegion & *destSubregion*, bool *enableOverwrite* = `true`)

Push some data in the ring-buffer (FIFO like operation)

Parameters

nbData	The number of element your want to push in the FIFO. nbData will contain the real number of pushed data in the region.
destSubregion	Will contain the sub-region that must receive data to push.
enableOverwrite	If set to true, a push with nbData too big will overwrite FIFO data. Else, nbData is adjusted to the maximum value without overwrite.

Returns

The return value is the state of the new region, if the operation succeeded. An overflow will return `MAPSConstRBRegionOverflowOp`. Else it returns `MAPSConstRBRegionInvalidOp`. !

10.48.2.13 MAPSConstRBRegionState MAPSRBRegion::SubRegion (MAPSRBRegion & *dest*, MAPSConstRBRegionSubRegion *type*, int *offsetX*, int *sizex*) const

Extract a sub-region from current region.

Parameters

dest	The destination region.
type	Define if the region must be extracted from the beginning or the end of src region.
offsetX	Define an offset from the starting point (beginning or end).
sizex	Size of the new region.

Returns

The return value is the state of the new region, if the operation succeeded. Else it returns `MAPSConstRBRegionInvalidOp`

The documentation for this class was generated from the following file:

- MAPSRBRegion.h

10.49 MAPSRealObject Struct Reference

The standard structure to transmit info about real objects.

Public Attributes

- int kind

  Specifies the kind of real object for this object.

- int id

  Identifier (for instance, obstacle number)

- MAPSFloat x

  The x coordinate of the object.

- MAPSFloat y

  The y coordinate of the object.

- MAPSFloat z

  The z coordinate of the object.

- int color

  The main color of the real object (RGB 24bits, Intel oriented : use MAPS_RGB macro)

- MAPSVehicle vehicle

  The real object is a vehicle.

- MAPSSign sign

  The real object is traffic sign.

- MAPSTree tree

  The real object is a tree.

Static Public Attributes

- static const int Vehicle

  One kind of real object.

- static const int Sign

  Another kind of real object.

- static const int Tree

  Another kind of real object.

User defined information

- int misc1

  Miscellaneous 1.

- int misc2

  Miscellaneous 2.

- int misc3

  Miscellaneous 3.

- void ∗ userdata

  TO USE WITH CARE! C++ EXPERTS ONLY.

10.49.1 Detailed Description

The standard structure to transmit info about real objects. The real content of the MAPSRealObject depends on the `kind` parameter.

By convention the coordinates of the object represent the position of the lower point in the middle of the object. The vehicle's coordinates represent the point on the floor at the middle of the back of the vehicle. A tree's coordinate is the coordinate of its trunk, at its base on the floor.

This structured type should evolve with the upcoming new RTMaps applications.

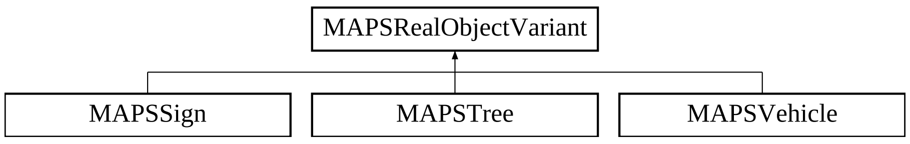
The documentation for this struct was generated from the following file:

- maps.hpp

10.50 MAPSRealObjectVariant Struct Reference

All real objects in RTMaps (vehicles, etc.) inherit from this structure.

Inheritance diagram for MAPSRealObjectVariant:



10.50.1 Detailed Description

All real objects in RTMaps (vehicles, etc.) inherit from this structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.51 MAPSRecordingStateListener Class Reference

The MAPSRecordingStateListener interface.

Public Member Functions

- virtual void CallbackRecordingStateChanged (const bool recording)=0

  Implement this function if you want to execute some code when the recording mode is activated or deactivated.

10.51.1 Detailed Description

The MAPSRecordingStateListener interface. Your class should inherit from this interface if you want it to be notified when RTMaps starts and stops recording.

Warning

If a MAPSComponent inherits this class, the Callback function is called from a different thread than the main thread of the component. Pay attention to threads synchronization. The callback function can be called as soon as the child class is constructed (in case of a MAPSComponent, the callback can be called even when the component is not running).

10.51.2 Member Function Documentation

10.51.2.1 virtual void MAPSRecordingStateListener::CallbackRecordingStateChanged (const bool *recording*) `[pure virtual]`

Implement this function if you want to execute some code when the recording mode is activated or deactivated.

Parameters

recording	It is set to `true` when the recording is activated, and to `false` when recording is stopped.

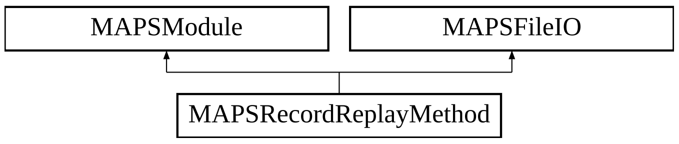
The documentation for this class was generated from the following file:

- maps.hpp

10.52 MAPSRecordReplayMethod Class Reference

The base class for all record/replay methods.

Inheritance diagram for MAPSRecordReplayMethod:



Protected Member Functions

- MAPSOutput & Output ()

  Returns a reference the output recorded or replayed.

- MAPSString FileName ()

  Returns the current file name.

- MAPSString FastFileName ()

  Returns the current fast file name.

- MAPSString DumpFileName (int dumpNb)

  Returns the dump file name (for the # dumpNb)

- MAPSString RecordPath ()

  Returns the RRM recordPath if specified, the recorder recordPath otherwise.

- MAPSString FastRecordPath ()

  Returns the RRM fastRecordPath if specified, the recorder fastRecordPath otherwise.

- int NbRecords ()

  Returns the current number of records.

- MAPSRecorder & Recorder ()

  Returns the recorder associated to this record/replay method.

- MAPSRecordReplayMethod (MAPSRecordReplayMethodDefinition &rmd, MAPSRecorder ∗recorder, MAPSOutput &output, bool record, bool copy)

  Constructor.

Buffer and pre-processing management

- virtual bool DoProcessBufferItem (MAPSIOElt &)

  Returns whether some processing (compression for instance) is needed before the item is stored. Generally overloaded.

- virtual void ProcessBufferItem (MAPSIOElt &source, MAPSIOElt &dest)

  The processing itself (for instance a compression algorithm). Generally overloaded.

Record management

- void RecordTime (MAPSIOElt &IOElt)

  Records the current record time in the .rec buffer.

- void RecordHeading (MAPSIOElt &IOElt)

  Records the heading in the .rec and .idy (the heading describes the output currently begin recorded)

- bool HeadingRecorded ()

  Has the heading already been recorded?

- const char ∗ HeadingHint ()

  Returns the heading hint (during replay)

- virtual void Store (MAPSIOElt &IOElt)=0

  The store operation (write to disk). Must be overloaded.

- virtual const char ∗ Hint (MAPSIOElt &IOElt)=0

  The hint found in the .rec file. Must be overloaded.

- virtual MAPSString HeadingHint (MAPSIOElt &)

  The function that sets the heading hint during record. Generally overloaded.

- void FileWrite (MAPSFileWriteHandle ∗fileHandle, const void ∗buffer, int size, MAPSIOElt ∗IOElt=NULL)

  An overload of MAPSFileIO::FileWrite, to ease the design of asynchronous recording methods.

- void Unlock (MAPSIOElt &IOElt)

  The MAPSFileIO::Unlock overload.

Trigger management

- void Set (MAPSProperty &p, bool value)

  Overload for set, taking into account the special "trigger" property.

- void Trigger (bool report=false)

  Sets the trigger.

Replay management

- virtual bool Replay (MAPSIOElt &IOElt, int rec, const char ∗hint, MAPSTypeInfo &type)=0

  The replay function. Must be overloaded.

Birth and death management

- virtual void BirthRecord (void)=0

  Birth in record mode. Must be overloaded.

- virtual void BirthReplay (void)

  Birth in replay mode. Default does nothing.

- virtual void DeathRecord (void)=0

  Death in record mode. Must be overloaded.

- virtual void DeathReplay (void)

  Death in replay mode. Default does nothing.

Blackbox management

- bool Blackboxed ()

  Are we in blackbox recording mode?

- virtual bool BlackboxCompatible (void)

  Is the component blackbox mode compatible? (default is no)

- virtual void BlackboxSwitch (MAPSString &)

  The blackbox switch function. Must be overloaded for blackbox compatible record methods.

- virtual void BlackboxDump (MAPSString &)

  The blackbox dump function.

Copy management

The basic database editing feature

- MAPSRecordReplayMethod & CopyDestination ()

  The record methods that is the destination of the current copy operation.

- bool Copying ()

  Am I in copy mode?

- virtual bool Copy (MAPSIOElt &IOElt, int rec, const char ∗hint, MAPSTypeInfo &type)

  The copy operation. Generally needs to be overloaded.

Error management

- void Error (const char ∗string, int importance=1)

  Emits an error.

10.52.1 Detailed Description

The base class for all record/replay methods. Examples of record/replay methods (RRMs) are raw, jpeg, jseq, wave, txt, mfile, etc. Some useful macros are available for easier development of RRMs. They are detailed in Record/replay method design.

10.52.2 Member Function Documentation

10.52.2.1 void MAPSRecordReplayMethod::Error (const char ∗ *string*, int *importance* = `1`) `[protected, virtual]`

Emits an error.

This will kill definitely the module. Use MAPSModule::ReportError if you want to go on executing some code.

Parameters

string	Error string to log and display
importance	Importance of the error : • 0 : The error is minor. It will not have any impact on the overall system behaviour. Some actions have been taken that will fix the problem. • 1 : The error is important. Some actions have been taken that might fix the problem. The system can go on running, but it might not function properly. • 2 : The error is critical. The system will not function properly thereafter.

Reimplemented from MAPSModule.

10.52.2.2 virtual MAPSString MAPSRecordReplayMethod::HeadingHint (MAPSIOElt &) [protected, virtual]

The function that sets the heading hint during record. Generally overloaded.

By default, returns an empty string (no hint).

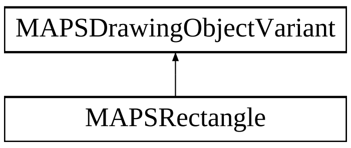
The documentation for this class was generated from the following file:

- maps.hpp

10.53 MAPSRectangle Struct Reference

The MAPSRectangle structure.

Inheritance diagram for MAPSRectangle:



Public Attributes

- int x1

  First point x.

- int y1

  First point y.

- int x2

  Second point x.

- int y2

  Second point y.

10.53.1 Detailed Description

The MAPSRectangle structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.54 MAPSRunnable< T > Class Template Reference

Helper class to create worker threads.

10.54.1 Detailed Description

template<typename T> class MAPSRunnable< T >

Helper class to create worker threads. A MAPSRunnableFunc function typically looks like: void∗ myFunc(void∗ dataPtr) { while(! myRunnable.IsStopRequested()) { do some stuff } } where dataPtr is the user data pointer given to the ctor or the Init function.

The documentation for this class was generated from the following file:

- maps.hpp

10.55 MAPSRunShutdownListener Class Reference

The MAPSRunShutdownListener interface.

Public Member Functions

- virtual void CallbackRun ()=0

  Implement this function to execute some code when diagram starts running.

- virtual void CallbackShutdown ()=0

  Implement this function to execute some code when diagram stops running.

10.55.1 Detailed Description

The MAPSRunShutdownListener interface. Your class should inherit from this interface if you want it to be notified of RTMaps Run/Shutdown events through functions CallbackRun() and CallbackShutdown();

Warning

If a MAPSComponent inherits this class, the Callback functions are called from a different thread than the main thread of the component. Pay attention to threads synchronization. The callback function can be called as soon as the child class is constructed (in case of a MAPSComponent, the callbacks can be called even when the component is not running).

10.55.2 Member Function Documentation

10.55.2.1 virtual void MAPSRunShutdownListener::CallbackRun ( ) `[pure virtual]`

Implement this function to execute some code when diagram starts running.

This function will be called by the RTMaps engine when the diagram starts to run, just before the diagram components start to run (i.e. just before their Birth() method are called).

10.55.2.2 virtual void MAPSRunShutdownListener::CallbackShutdown ( ) `[pure virtual]`

Implement this function to execute some code when diagram stops running.

This function will be called by the RTMaps engine when the diagram stops running, after the diagram components stop running (i.e. after their Death() method are called).

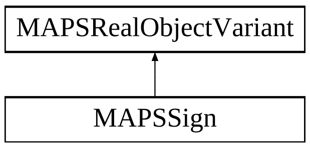
The documentation for this class was generated from the following file:

- maps.hpp

10.56 MAPSSign Struct Reference

The MAPSSign structure : a kind of MAPSRealObject.

Inheritance diagram for MAPSSign:



Public Attributes

- int type
  The type of the sign.
- int speedLimit
  Speed limit information.
- int trafficLightState
  State of a traffic light.
- char text [256]
  Text information.

Static Public Attributes

- static const int Stop

  Stop sign.

- static const int Yield

  Yield sign.

- static const int OneWay

  One Way sign.

- static const int TurnLeft

  Turn left sign.

- static const int TurnRight

  Turn right sign.

- static const int SpeedLimit

  Speed limit sign.

- static const int EndOfSpeedLimit

  End of speed limit sign.

- static const int TrafficLight

  A traffic light.

- static const int TextOnly

  Text only sign.

- static const int Red

  The traffic light is red.

- static const int Orange

  The traffic light is orange.

- static const int Green

  The traffic light is green.

- static const int OrangeBlinking

  The orange is blinking.

10.56.1 Detailed Description

The MAPSSign structure : a kind of MAPSRealObject.

See also

MAPSRealObject

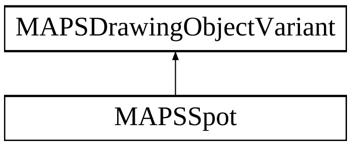
The documentation for this struct was generated from the following file:

- maps.hpp

10.57 MAPSSpot Struct Reference

The MAPSSpot structure.

Inheritance diagram for MAPSSpot:



Public Attributes

- int x

  The x coordinate of the spot.

- int y

  The y coordinate of the spot.

- int kind

  The kind of spot (MAPSSpot::Cross, etc.)

Static Public Attributes

- static const int Point

  This spot is simply a point.

- static const int Cross

  This spot is a cross.

- static const int Circle

This spot is a small circle.

- static const int CircledPoint

  This spot is a point with a circle around.

- static const int CircledCross

  This spot is a cross with a circle around.

10.57.1 Detailed Description

The MAPSSpot structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.58 MAPSStackedString< BUFFER_SIZE > Class Template Reference

The RTMaps flexible and fast string template class.

Public Member Functions

- int Length () const

  Returns the length of the string.

- int Len () const

  Returns the length of the string.

- char & operator[] (int pos) const

  Gets/Sets the character at position pos.

- operator const char ∗ () const

  Automatic cast to `const char*`

- MAPSStackedString & operator= (const MAPSStackedString &s)

  String copy operator. Returns a reference to the current string (`*this`*).*

- MAPSStackedString & operator= (const char ∗s)

  String copy operator. Returns a reference to the current string (`*this`*).*

- MAPSRegExp & operator= (MAPSRegExp &regexp)

  Match against a regular expression.

- const char * Beginning () const

  Returns a pointer to the beginning of the string.

- int Pos (const char *ptr) const

  Returns the position corresponding to a pointer to a character of the string.

- MAPSStackedString & Shorten (int pos)

  Reduces the length of the string to pos.

- void Clear ()

  Clears the content of the string.

- MAPSStackedString & UpdateLength ()

  Updates the length of the string by finding the first 0 character in the string buffer.

- MAPSStackedString & RemoveCRLF ()

  Removes the trailing `'\n'` *or* `'\r\n'` *characters.*

- MAPSStackedString & RemoveSpaces ()

  Removes spaces, tabs or carriage returns found at the beginning and at the end of the string.

- MAPSStackedString & Uppercase ()

  Sets the string to uppercase ('abc'->'ABC'). Returns a reference to the current string (`*this`*).*

- MAPSStackedString & FirstUppercase ()

  Sets the first character of the string to uppercase.

- MAPSStackedString & Substitute (char s, char d)

  Substitutes all s characters with d. Returns a reference to the current string (`*this`*).*

- MAPSStackedString & Reverse (const MAPSStackedString &s)

  Sets the current string to the reverse of string s. Returns a reference to the current string (`*this`*).*

Related Functions

(Note that these are not member functions.)

- template<int BUFFER_SIZE>
  MAPSStreamedString & operator<< (MAPSStreamedString &out, const MAPSStacked-String< BUFFER_SIZE > &str)

  Standard operator.

Strings comparison functions

- bool operator== (const MAPSStackedString &s) const

  String comparaison, equivalent to `strcmp`, i.e. case-sensitive.

- bool operator== (MAPSStackedString &s) const

  String comparaison, equivalent to `strcmp`, i.e. case-sensitive.

- bool operator!= (const MAPSStackedString &s) const

  String comparaison, equivalent to `strcmp`, i.e. case-sensitive.

- bool operator== (const char ∗s) const

  String comparaison, equivalent to `strcmp`, i.e. case-sensitive.

- bool operator!= (const char ∗s) const

  String comparaison, equivalent to `strcmp`, i.e. case-sensitive.

- bool operator< (const MAPSStackedString &s) const

  Simple string order, case sensitive.

- bool operator> (const MAPSStackedString &s) const

  Simple string order, case sensitive.

- bool Equals (const MAPSStackedString &s) const

  String comparison, equivalent to `strcmp`, i.e. case-sensitive.

- bool Iequals (const MAPSStackedString &s) const

  String comparaison, equivalent to `stricmp`, i.e. lowercase comparison of strings.

Strings concatenation functions

- MAPSStackedString & operator+= (const char ∗s)

  String concatenation operator. Returns a reference to the current string (∗`this`).

- MAPSStackedString & operator+= (unsigned char ∗s)

  String concatenation operator. Returns a reference to the current string (∗`this`).

- MAPSStackedString & operator+= (const MAPSStackedString &s)

  String concatenation operator. Returns a reference to the current string (∗`this`).

- MAPSStackedString & operator+= (const char c)

  Concatenates character c to the string. Returns a reference to the current string (∗`this`).

Substring extraction functions

All of these functions return a new string, or an array of strings.

- MAPSStackedString operator() (int start, int stop) const

  Extracts a part of the string.

- MAPSStackedString UpTo (char s, bool from_beginning=true) const

  Returns the part of the string up to the first occurence of character s.

- MAPSStackedString Up2 (char s, bool from_beginning=true) const

  Returns the part of the string up to the first occurence of character s.

- MAPSStackedString UpTo (int m, bool from_beginning=true) const

  Returns the characters starting from the beginning of the string to mth character, counting from the beginning or the end of the string.

- MAPSStackedString Up2 (int m, bool from_beginning=true) const

  Returns the characters starting from the beginning of the string to mth character, counting from the beginning or the end of the string.

- MAPSStackedString Tail (int m, bool from_beginning=true) const

  Returns the characters starting from the mth character, counting from the beginning or the end of the string, to the last character.

- MAPSStackedString Tail (char s, bool from_beginning=true) const

  Returns the characters found after the first occurence of character s, starting from the beginning or the end of the string.

- MAPSStackedString Part (int start, int stop) const

  Extracts a part of the string.

- MAPSStackedString Left (int length) const

  Extracts the left part of the string.

- MAPSStackedString Right (int length) const

  Extracts the right part of the string.

- MAPSStackedString Mid (int start, int length=-1) const

  Extracts a part of the string.

- MAPSArray< MAPSStackedString > Split (const char c) const

  Splits the string into several strings according to the specified separation character.

String tokenization functions

- MAPSStackedString & TokenReset ()

  Resets the token search. Returns a reference to the current string (∗`this`).

- bool Token (const char ∗delim, MAPSStackedString &s)

  Returns the next token.

- const char ∗ Strtok (const char ∗delim)

  Equivalent to standard C library `strtok`.

Find functions

- bool Find (char c, int &pos, int start=0) const

  Finds character c in the string, starting at position start.

- bool Find (const char ∗s, int &pos, int start=0) const

  Finds one of the characters in s in the string, starting at position start.

- int FindStr (const MAPSStackedString &s, int start=0) const

  Finds the substring s in the string, starting at position start.

- int FindStr (const char ∗s, int start=0) const

  Finds the substring s in the string, starting at position start.

Formatting

- MAPSStackedString & Format (const char ∗format_string, const char ∗s)

  Formats a string.

- MAPSStackedString & Format (const char ∗format_string, int m)

  Formats an integer.

- MAPSStackedString & Format (const char ∗format_string, double x)

  Formats a floating point number.

Constructors

- MAPSStackedString (const char ∗string_format, const char ∗s)

  Formatting constructor.

- MAPSStackedString (const char ∗string_format, int n)

Formatting constructor.

- MAPSStackedString (const char ∗string_format, double x)

  Formatting constructor.

- MAPSStackedString (const char ∗s)

  Constructs a string from another string in C format.

- MAPSStackedString (const char c)

  Constructs a string containing a single character.

- MAPSStackedString (const MAPSStackedString &s)

  Copy constructor.

- MAPSStackedString (int n, bool set_length=false)

  Preallocating constructor. By default, sets the length to 0. Optionally sets the length to n, setting the nth character to 0 (End of string)

- MAPSStackedString (int n, int nbdigits)

  Simple integer formatting. Fills with 0 if the number n to format is too short.

- MAPSStackedString ()

  Simple constructor.

10.58.1 Detailed Description

template<int BUFFER_SIZE> class MAPSStackedString< BUFFER_SIZE >

The RTMaps flexible and fast string template class. This template class is a very efficient and exhaustive string template class. In this template class, the parameter *BUFFER_-SIZE* gives the preallocated size in bytes for the string. The memory is preallocated in the stack, which avoids any slow call to malloc (and any memory fragmentation due to heap usage). Note that if the string actually contains more characters than the *BUFFER_SIZE* template parameter, the buffer is automatically moved to the heap via malloc.

A common value for *BUFFER_SIZE* is 40. The usual MAPSString class is defined as MAPSStackedString<40>.

Among other features, the MAPSStackedString class provides :

- string formatting features, based on the Microsoft familiar formatting features found in Visual Basic or Excel.
- operators and functions for basic string manipulation (concatenation, extraction, find, substitution)
- tokenization functions (strtok like)

See also

MAPSString MAPSStreamedString

10.58.2 Member Function Documentation

10.58.2.1 template<int BUFFER_SIZE> bool MAPSStackedString< BUFFER_SIZE >::Find (char *c*, int & *pos*, int *start* = `0`) const

Finds character *c* in the string, starting at position *start*.

Returns `true` if found, and updates the *pos* variable.

10.58.2.2 template<int BUFFER_SIZE> bool MAPSStackedString< BUFFER_SIZE >::Find (const char * *s*, int & *pos*, int *start* = `0`) const

Finds one of the characters in *s* in the string, starting at position *start*.

Returns `true` if found, and updates the *pos* variable.

Warning

This function does not look for the whole string *s*. If this is what you want, have a look at FindStr functions.

10.58.2.3 template<int BUFFER_SIZE> MAPSStackedString MAPSStackedString< BUFFER_SIZE >::Left (int *length*) const

Extracts the left part of the string.

Similar to Visual Basic `Left$` function.

Parameters

length	Indicates how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in the string, the entire string is returned.

10.58.2.4 template<int BUFFER_SIZE> MAPSStackedString MAPSStackedString< BUFFER_SIZE >::Mid (int *start*, int *length* = `-1`) const

Extracts a part of the string.

Similar to Visual Basic `Mid$` function.

Parameters

start	Character position in the string at which the part to be taken begins. If *start* is greater than the number of characters in the string, Mid returns a zero-length string ("").

length	(optional) Number of characters to return. If omitted or if there are fewer than *length* characters in the text (including the character at start), all characters from the start position to the end of the string are returned.

10.58.2.5 template<int BUFFER_SIZE> MAPSStackedString MAPSStackedString< BUFFER_SIZE >::operator() (int *start*, int *stop*) const

Extracts a part of the string.

Returns all the characters between positions *start* and *stop* (included).

10.58.2.6 template<int BUFFER_SIZE> MAPSStackedString MAPSStackedString< BUFFER_SIZE >::Part (int *start*, int *stop*) const

Extracts a part of the string.

Returns all the characters between positions *start* and *stop* (included).

10.58.2.7 template<int BUFFER_SIZE> MAPSStackedString MAPSStackedString< BUFFER_SIZE >::Right (int *length*) const

Extracts the right part of the string.

Similar to Visual Basic `Right$` function.

Parameters

length	Indicates how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in the string, the entire string is returned.

10.58.2.8 template<int BUFFER_SIZE> MAPSStackedString& MAPSStackedString< BUFFER_SIZE >::Shorten (int *pos*)

Reduces the length of the string to *pos*.

Puts a 0 (end of string) in position *pos*

10.58.2.9 template<int BUFFER_SIZE> bool MAPSStackedString< BUFFER_SIZE >::Token (const char ∗ *delim*, MAPSStackedString< BUFFER_SIZE > & *s*)

Returns the next token.

Parameters

delim	The set of characters in *delim* specifies possible delimiters of the token to be found in the string on the current call
s	Holds the substring

Returns

Returns `false` when no more tokens are found.

10.58.2.10 template<int BUFFER_SIZE> MAPSStackedString MAPSStackedString< BUFFER_SIZE >::Up2 (char *s*, bool *from_beginning* = `true`) const

Returns the part of the string up to the first occurence of character *s*.

Returns the part of the string up to the first occurence of character *s*.

Note : When *from_beginning* is `false`, i.e. when we are seeking the first occurence of character *s* in the string beginning from the end of the string, if there is no such character, then the returned string is empty. On the other hand, when *from_beginning* is `true`, i.e. when we are seeking the first occurence of character *s* in the string beginning from the first character of the string, if there is no such character, then the whole string is returned.

10.58.2.11 template<int BUFFER_SIZE> MAPSStackedString MAPSStackedString< BUFFER_SIZE >::UpTo (char *s*, bool *from_beginning* = `true`) const

Returns the part of the string up to the first occurence of character *s*.

Note : When *from_beginning* is `false`, i.e. when we are seeking the first occurence of character *s* in the string beginning from the end of the string, if there is no such character, then the returned string is empty. On the other hand, when *from_beginning* is `true`, i.e. when we are seeking the first occurence of character *s* in the string beginning from the first character of the string, if there is no such character, then the whole string is returned.

10.58.3 Friends And Related Function Documentation

10.58.3.1 template<int BUFFER_SIZE> MAPSStreamedString & operator<< (MAPSStreamedString & *out*, const MAPSStackedString< BUFFER_SIZE > & *str*) `[related]`

Standard operator.

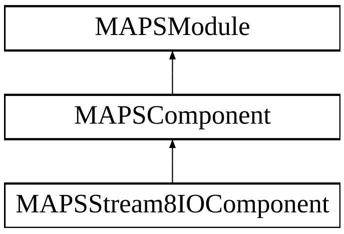
The documentation for this class was generated from the following file:

- maps.hpp

10.59 MAPSStream8IOComponent Class Reference

Processing component template with 1 Stream8 input and 1 Stream8 output.

Inheritance diagram for MAPSStream8IOComponent:



Protected Member Functions

- virtual void NewDataCallback (MAPSRBRegion &region, const unsigned char ∗data, const MAPSTimestamp ∗timestamp, const MAPSTimestamp ∗timeOfIssue)=0

  Processing function to implement in the child class.

Output writing functions.

- virtual bool **SendData** (const unsigned char ∗stream, int streamSize, MAPSTimestamp ts=0)
- virtual bool **SendData** (const char ∗stream, int streamSize, MAPSTimestamp ts=0)

10.59.1 Detailed Description

Processing component template with 1 Stream8 input and 1 Stream8 output.

The documentation for this class was generated from the following file:

- MAPSStream8IOComponent.h

10.60 MAPSStreamedString Class Reference

The RTMaps streamed string class.

Inherits MAPSStackedString$< 40 >$.

10.60.1 Detailed Description

The RTMaps streamed string class. The MAPSStreamedString class extends the MAPSString features. The main extension is the operator$<<$ support. Special formatting options are also available, in the namespace `MAPSSManip`:

- `dec`, `hex`, `oct` for integers. For example:

```
MAPSStreamedString sx("Formatting test:");
unsigned int i = 15;
sx << "i=" << i << ", 0" << MAPSSManip::oct << i << ", 0x" << MAPSSManip::hex <<
      i;
Now sx contains: "Formatting test: i=15, 017, 0xF"
```

- `endl`: adds just a '\n' to the end of the string
- `ends`: adds just a '\0' to the end of the string

Note

If you use intensively formatting options, you can add

```
using namespace MAPSSManip;
```

after the inclusion of maps.hpp, but you may have conflicts with STL names.

See also

MAPSString MAPSStackedString

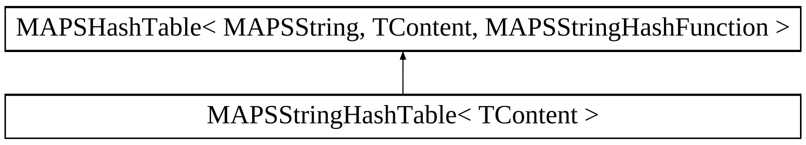
The documentation for this class was generated from the following file:

- maps.hpp

10.61 MAPSStringHashTable< TContent > Class Template Reference

The RTMaps string hash table class.

Inheritance diagram for MAPSStringHashTable< TContent >:


MAPSHashTable< MAPSString, TContent, MAPSStringHashFunction >
MAPSStringHashTable< TContent >

Public Member Functions

- MAPSStringHashTable (int n=16)

  Constructor.

10.61.1 Detailed Description

template<typename TContent> class MAPSStringHashTable< TContent >

The RTMaps string hash table class. As its name says, the key is a string. Most of the code of hashing function for strings is inspired of Bob Jenkins' work (`bob_jenkins@burtleburtle.net`). See `http://burtleburtle.net/bob/hash/evahash.html`

The documentation for this class was generated from the following file:

- maps.hpp

10.62 MAPSSynchronizer Class Reference

The RTMaps Synchroniser tool.

Public Member Functions

- void SignalSynchronizationCommand (MAPSTimestamp t)

  Sends a synchronization command to the synchronizable clock.

10.62.1 Detailed Description

The RTMaps Synchroniser tool. Provides the function that any RTMaps module can use to synchronize the RTMaps clock on an external signal/clock (GPS, another RTMaps...). Since there can be only one synchronizer at a time, the pointer to the synchronizer object can be requested through function MAPS::GetSynchronizer.

10.62.2 Member Function Documentation

10.62.2.1 void MAPSSynchronizer::SignalSynchronizationCommand (MAPSTimestamp *t*)

Sends a synchronization command to the synchronizable clock.

Parameters

t	Specifies the timestamp to synchronize the synchronizable clock on.

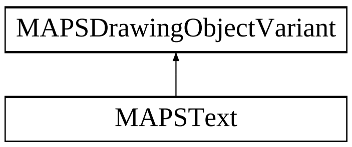
The documentation for this class was generated from the following file:

- maps.hpp

10.63 MAPSText Struct Reference

The MAPSText structure.

Inheritance diagram for MAPSText:



Public Attributes

- int x

  The leftmost coordinate.

- int y

  The topmost coordinate.

- int cwidth

  The width of one character (in pixels)

- int cheight

  The height of one character (in pixels)

- int orientation

  In degrees, around the upper left corner, counter clockwise.

- int bkcolor

  Background color (Use the MAPS_RGB macro to set the color).

- char text [32]

  The text to write.

10.63.1 Detailed Description

The MAPSText structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.64 MAPSTimer Class Reference

The MAPSTimer class for managing timers.

Public Member Functions

- void Start ()

  Starts the timer.

- void Stop ()

  Stops the timer.

- void Reset ()

  Resets the timer.

- MAPSTimestamp Time ()

  Returns the current time measure in microseconds.

- MAPSTimer ()

  Default constructor.

10.64.1 Detailed Description

The MAPSTimer class for managing timers.

The documentation for this class was generated from the following file:

- maps.hpp

10.65 MAPSTimeStateListener Class Reference

The MAPSTimeStateListener interface.

Public Member Functions

- virtual void CallbackTimeJumped (const MAPSTimestamp destTimestamp)=0

  Implement this function to execute some code each time the RTMaps current time jumps. (due to an action of the user on the VCR slider for example)

- virtual void CallbackTimeSpeedChanged (const int newTimeSpeed)=0

  Implement this function to execute some code each time the RTMaps changes during execution of a diagram.

- virtual void CallbackTimePaused (const bool paused)=0

  Implement this function to execute some code each time the RTMaps execution is paused, or the pause is released.

10.65.1 Detailed Description

The MAPSTimeStateListener interface. Your class should inherit from this interface if you want it to be notified of RTMaps clocks states changes such as changes of time-speed, jumps in time, or pausing events.

Warning

> If a MAPSComponent inherits this class, the Callback functions are called from a different thread than the main thread of the component. Pay attention to threads synchronization. The callback function can be called as soon as the child class is constructed (in case of a MAPSComponent, the callbacks can be called even when the component is not running).

10.65.2 Member Function Documentation

10.65.2.1 virtual void MAPSTimeStateListener::CallbackTimeJumped (const MAPSTimestamp *destTimestamp*) `[pure virtual]`

Implement this function to execute some code each time the RTMaps current time jumps. (due to an action of the user on the VCR slider for example)

Parameters

destTimestamp	The timestamp that has been reached during the jump in time. This function is called just after the destination timestamp has been taken into account by the RTMaps engine.

10.65.2.2 virtual void MAPSTimeStateListener::CallbackTimePaused (const bool *paused*) `[pure virtual]`

Implement this function to execute some code each time the RTMaps execution is paused, or the pause is released.

Parameters

paused	It is set to `true` if the execution is pause, and to `false` if the pause is released.

10.65.2.3 virtual void MAPSTimeStateListener::CallbackTimeSpeedChanged (const int *newTimeSpeed*) `[pure virtual]`

Implement this function to execute some code each time the RTMaps changes during execution of a diagram.

Parameters

newTime-Speed	The new timespeed that has been reached. This function is called just after the new timespeed has been taken into account by the RTMaps engine.

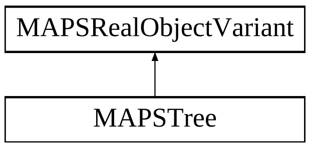
The documentation for this class was generated from the following file:

- maps.hpp

10.66 MAPSTree Struct Reference

The MAPSTree structure : a kind of MAPSRealObject.

Inheritance diagram for MAPSTree:



Public Attributes

- MAPSFloat height

  The height of the tree.

- MAPSFloat radius

  The radius of the tree.

- int shape

  The shape of the tree.

Static Public Attributes

- static const int Cone

  One possible shape.

- static const int Round

  Another possible shape.

- static const int Cylinder

  A third possible shape.

10.66.1 Detailed Description

The MAPSTree structure : a kind of MAPSRealObject.

See also

MAPSRealObject

The documentation for this struct was generated from the following file:

- maps.hpp

10.67 MAPSTriangles3D Struct Reference

The RTMaps structure for supporting 3D triangles output.

Classes

- struct Object

  The object structure.

- struct Point

  The 3D point structure.

- struct Triangle

  The triangle structure itself.

Public Attributes

- int nbObjects

  Number of objects in the scene.

- Object ∗ objects

  All the objects in the scene.

- int nbPoints

  Number of points in the scene.

- Point ∗ points

  All the points in the scene.

- int nbTriangles

  Number of triangles in the scene.

- Triangle ∗ triangles

  All the triangles in the scene.

10.67.1 Detailed Description

The RTMaps structure for supporting 3D triangles output.

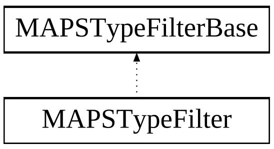
The documentation for this struct was generated from the following file:

- maps.hpp

10.68 MAPSTypeFilter Class Reference

The RTMaps type filter class.

Inheritance diagram for MAPSTypeFilter:



Public Member Functions

- MAPSTypeFilter (const MAPSTypeFilterBase &model, const char ∗filterName=NULL, const char ∗filterUnit=NULL)

  Constructs a type filter based on a model. Assigns a new name or unit filter (regular expression) if needed.

10.68.1 Detailed Description

The RTMaps type filter class.

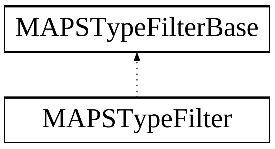
The documentation for this class was generated from the following file:

- maps.hpp

10.69 MAPSTypeFilterBase Struct Reference

The RTMaps type filter structure.

Inheritance diagram for MAPSTypeFilterBase:



Public Attributes

- MAPSTypeInfoValue mask

  First AND filter mask applied by MAPS::TypeFilter to the data type to pass through the filter.

- MAPSTypeInfoValue x

  The data type to pass through the filter is compared to x after applying the first AND mask.

- MAPSTypeInfoValue maskX

  The second AND filter mask.

- const char ∗ nameFilter

  The optional name filter.

- const char ∗ unitFilter

  The optional unit filter.

10.69.1 Detailed Description

The RTMaps type filter structure. A type filter structure must be provided for each input, since all data flowing within a RTMaps system do have a type associated to it and are filtered before getting in an input.

The RTMaps type filtering behaviour is the following :

- First of all, if a name or a unit filter is set, only the data whose unit or name matches the filter unit or name parameter pass through the filter.
- In a second phase, a AND mask is applied to the type (parameter mask). The result of this AND operation must be equal to the x parameter. Otherwise, the type is rejected. This enables to test some bits. For instance, if we want to let only vectors pass through the filter, we set mask to MAPS::VectorFlag and set x to MAPS::VectorFlag. This will require the bit MAPS::VectorFlag to be set. On the contrary, if we don't want to let vectors pass through the filter, we set mask to MAPS::VectorFlag and set x to 0.

- In a third and last phase, another AND mask is applied (parameter maskX). This mask is less restrictive, since if the result is different from 0, i.e. if any of the bits are set, the type will pass through the filter. For instance, if we want to let numbers pass through the filter, we can set maskX to MAPS::Integer | MAPS::Float.

This is actually the code of the MAPS::TypeFilter function :

```
bool MAPS::TypeFilter(const MAPSTypeInfo& type, MAPSTypeFilterBase& filter)
{
    if (filter.nameFilter) {
        if (filter.nameRegexp==NULL) filter.nameRegexp=new MAPSRegExp(filter.
      nameFilter);
        if ((type.name==NULL)||(filter.nameRegexp->Match(*(type.name))==false)) {

            return false;
        }
    }
    if (filter.unitFilter) {
        if (filter.unitRegexp==NULL) filter.unitRegexp=new MAPSRegExp(filter.
      unitFilter);
        if ((type.unit==NULL)||(filter.unitRegexp->Match(*(type.unit))==false)) {

            return false;
        }
    }
    return (((type.value&filter.mask)==(filter.x&filter.mask))&&(type.value&filte
      r.maskX));
}
```

See also

MAPS::TypeFilter

10.69.2 Member Data Documentation

10.69.2.1 const char* MAPSTypeFilterBase::nameFilter

The optional name filter.

Only the types matching this regular expression can pass through the filter. If set to `NULL`, no optional name filtering is done (any named data can pass through the filter).

10.69.2.2 const char* MAPSTypeFilterBase::unitFilter

The optional unit filter.

Only the types matching this regular expression can pass through the filter. If set to `NULL`, no unit filtering is done (any unit can pass through the filter).

The documentation for this struct was generated from the following file:

- maps.hpp

10.70 MAPSTypeInfo Struct Reference

The RTMaps type information structure.

Public Member Functions

- void SetName (const char ∗n)

  Dynamically sets the optional name of the output data.

- void SetUnit (const char ∗u)

  Dynamically sets the unit of the output data.

- MAPSTypeInfo & operator= (const MAPSTypeInfo &type)

  Copy operator.

- ∼MAPSTypeInfo ()

  Destructor.

Public Attributes

- MAPSTypeInfoValue value

  This is the 64-bit value describing the type (integer, MAPSFloat...) of the output data.

- MAPSString ∗ name

  An optional name associated to the output data type.

- MAPSString ∗ unit

  A string describing the unit of the output data.

10.70.1 Detailed Description

The RTMaps type information structure. A MAPSTypeInfo structure must be provided for each output, since all data flowing within an RTMaps must have a type associated to it.

10.70.2 Member Function Documentation

10.70.2.1 void MAPSTypeInfo::SetName (const char ∗ *n*)

Dynamically sets the optional name of the output data.

Note that the optional name is usually set statically in the output definitions.

See also

MAPS_OUTPUT

10.70.2.2 void MAPSTypeInfo::SetUnit (const char $*$ *u*)

Dynamically sets the unit of the output data.

Note that the unit is usually set statically in the output definitions.

See also

MAPS_OUTPUT

10.70.3 Member Data Documentation

10.70.3.1 MAPSString$*$ MAPSTypeInfo::name

An optional name associated to the output data type.

This is optional, except when value is set to MAPS::Structure, which means that the type of the output is a user-specific structure. In that case, a name must be associated to the structure in order to distinguish it among all the user-specific structures.

10.70.3.2 MAPSString$*$ MAPSTypeInfo::unit

A string describing the unit of the output data.

For instance, "km/h", or "m/s".

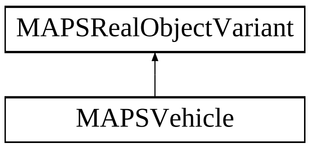
The documentation for this struct was generated from the following file:

- maps.hpp

10.71 MAPSVehicle Struct Reference

The MAPSVehicle structure : a kind of MAPSRealObject.

Inheritance diagram for MAPSVehicle:



Public Attributes

- MAPSFloat theta

  The vehicle's heading (in degrees)

- MAPSFloat speed

  The vehicle's speed.

- int kind

  The kind of vehicle.

- MAPSFloat width

  The width of the vehicle (in meters)

- MAPSFloat height

  The height of the vehicle (in meters)

- MAPSFloat length

  The length of the vehicle (in meters)

- int model

  Type of car.

- bool braking

  Is the vehicle currently breaking?

- MAPSFloat confidence

  Confidence in the information about this vehicle.

Static Public Attributes

- static const int Car

  The vehicle is a car (kind)

- static const int Bus

  The vehicle is a bus (kind)

- static const int Truck

  The vehicle is a truck (kind)

- static const int Bike

  The vehicle is a bike (kind)

- static const int Motorcycle

  The vehicle is a motorcycle (kind)

The vehicle position accuracy

- MAPSFloat dx

  Accuracy of coordinate along x.

- MAPSFloat dy

  Accuracy of coordinate along y.

- MAPSFloat dz

  Accuracy of coordinate along z.

10.71.1 Detailed Description

The MAPSVehicle structure : a kind of MAPSRealObject.

See also

MAPSRealObject

The documentation for this struct was generated from the following file:

- maps.hpp

10.72 MAPSWin32 Class Reference

The RTMaps Win32 support class.

Friends

- class MAPSEvent

Useful functions

- static bool OSIsWindowsNT ()

  Returns `true` if the OS is Windows 2000, XP or Server 2003, `false` otherwise.

- static int OSVersion ()

  Returns 5 for Windows 2000, XP or Server 2003, 4 for Windows NT4.

- static void AbsoluteTimeToSystemTime (MAPSAbsoluteTime &s, SYSTEMTIME &t)

  Translate MAPSAbsoluteTime structure to Windows SYSTEMTIME structure.

- static void SystemTimeToAbsoluteTime (SYSTEMTIME &t, MAPSAbsoluteTime &s)

  Translate Windows SYSTEMTIME structure to MAPSAbsoluteTime structure.

- static DWORD MainThreadId ()

  Returns the main thread Id (the one that processes WM_QUIT messages generally)

10.72.1 Detailed Description

The RTMaps Win32 support class.

The documentation for this class was generated from the following file:

- maps.hpp

10.73 MAPSTriangles3D::Object Struct Reference

The object structure.

Public Attributes

- int id

  An ID for the object.

- int texture

  Texture ID information.

- int firstTriangle

  The index of the first triangle among all the triangles.

- int nbOfTriangles

  The number of triangles for this object.

10.73.1 Detailed Description

The object structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.74 MAPSTriangles3D::Point Struct Reference

The 3D point structure.

Public Attributes

- float x

  x coordinate

- float y

  y coordinate

- float z

  z coordinate

- int landmark

  Optional. May be useful if you need to make space transformations. Just to keep an index to the reference location.

- int xt

  Texture x coordinate, for texture mapping.

- int yt

  Texture y coordinate, for texture mapping.

10.74.1 Detailed Description

The 3D point structure.

The documentation for this struct was generated from the following file:

- maps.hpp

10.75 MAPSTriangles3D::Triangle Struct Reference

The triangle structure itself.

Public Attributes

- int p1

  The 1st point constituting the triangle.

- int p2

The 2nd point constituting the triangle.

- int p3

  The 3rd point constituting the triangle.

- int color

  The color of the triangle if no texture is provided (use MAPS_RGB or MAPS_RGBA to set the color)

10.75.1 Detailed Description

The triangle structure itself.

The documentation for this struct was generated from the following file:

- maps.hpp

Chapter 11

File Documentation

11.1 maps.hpp File Reference

The RTMaps engine header file - Version 3.4 build 99.

Classes

- class MAPSCouple< T >

  Template class for couple data.

- class MAPSBasicListItem

  The basic list item class.

- class MAPSListIterator

  The list iterator class.

- class MAPSListItem< T >

  The list item template class.

- class MAPSBasicList

  The base class for the MAPSList template class.

- class MAPSList< T >

  The RTMaps double linked list template class.

- class MAPSList< T >::MAPSIterator

  Iterator on MAPSList objects.

- class MAPSStackedString< BUFFER_SIZE >

  The RTMaps flexible and fast string template class.

- class MAPSStreamedString

  The RTMaps streamed string class.

- class MAPSArray< T >

  The RTMaps array class.

- class MAPSMatrix2< T >

  The RTMaps MAPSMatrix2 class, a powerful matrix management class.

- class MAPSPair< TKey, TContent >

  The RTMaps (key,content) pair template class.

- class MAPSHashTableIterator

  The RTMaps hash table iterator.

- class MAPSHashTable< TKey, TContent, H >
- class MAPSStringHashTable< TContent >

  The RTMaps string hash table class.

- class MAPSIntegerHashTable< TContent >

  The RTMaps integer hash table template class.

- class MAPSPtrHashTable< TContent >

  The RTMaps pointer hash table template class.

- class MAPSMutex

  The RTMaps mutex class.

- class MAPSEvent

  The RTMaps event class.

- class MAPSFileWriteHandle

  The class that is used to manage all RTMaps write file operations.

- class MAPSFileReadHandle

  The class that is used to manage all RTMaps read file operations.

- class MAPSWin32

  The RTMaps Win32 support class.

- struct IplROI

  The IPL Region Of Interest structure.

- struct IplImage

  The famous IplImage structure, today's standard structure for image processing.

- struct MAPSAbsoluteTime

    Absolute time structure.

- struct MAPSCANFrame

    CAN Frames structure.

- struct MAPSRealObjectVariant

    All real objects in RTMaps (vehicles, etc.) inherit from this structure.

- struct MAPSVehicle

    The MAPSVehicle structure : a kind of MAPSRealObject.

- struct MAPSTree

    The MAPSTree structure : a kind of MAPSRealObject.

- struct MAPSSign

    The MAPSSign structure : a kind of MAPSRealObject.

- struct MAPSRealObject

    The standard structure to transmit info about real objects.

- struct MAPSDrawingObjectVariant

    All drawing objects in RTMaps (circles, etc.) inherit from MAPSDrawingObjectVariant.

- struct MAPSLine

    The MAPSLine structure.

- struct MAPSRectangle

    The MAPSRectangle structure.

- struct MAPSCircle

    The MAPSCircle structure.

- struct MAPSEllipse

    The MAPSEllipse structure.

- struct MAPSText

    The MAPSText structure.

- struct MAPSSpot

    The MAPSSpot structure.

- struct MAPSDrawingObject

    The MAPSDrawingObject : a standard for simple overlay shapes.

- struct MAPSTriangles3D

The RTMaps structure for supporting 3D triangles output.

- struct MAPSTriangles3D::Point
  The 3D point structure.
- struct MAPSTriangles3D::Triangle
  The triangle structure itself.
- struct MAPSTriangles3D::Object
  The object structure.
- struct MAPSComplex
  Complex number structure.
- struct MAPSMatrix
  MATLAB-Like matrix (Complex and columnwise, like in fortran)
- struct MAPSImage
  The RTMaps Image type.
- struct MAPSTypeInfo
  The RTMaps type information structure.
- struct MAPSTypeFilterBase
  The RTMaps type filter structure.
- class MAPSTypeFilter
  The RTMaps type filter class.
- struct MAPSEnumStruct
  Enumeration structure.
- class MAPSOutput
  The RTMaps Module Output class.
- class MAPSInput
  The RTMaps Module Input class.
- class MAPSProperty
  The RTMaps Module Property class.
- class MAPSAction
  The RTMaps Module Action class.
- class MAPSMutex2
  The RTMaps advanced mutex class.

- class MAPSFileIO

  The RTMaps File I/O support class.

- class MAPSIOElt

  The RTMaps I/O Buffer Element.

- class MAPSRunShutdownListener

  The MAPSRunShutdownListener interface.

- class MAPSTimeStateListener

  The MAPSTimeStateListener interface.

- class MAPSRecordingStateListener

  The MAPSRecordingStateListener interface.

- class MAPSBaseClock

  MAPSBaseClock class.

- class MAPSSynchronizer

  The RTMaps Synchroniser tool.

- class MAPSModule

  The base class for all RTMaps modules.

- class MAPSComponent

  The base class for all RTMaps components.

- class MAPSRecordReplayMethod

  The base class for all record/replay methods.

- class MAPSTimer

  The MAPSTimer class for managing timers.

- class MAPSRunnable< T >

  Helper class to create worker threads.

Namespaces

- namespace MAPS

  The main RTMaps namespace.

Defines

- #define MAPS_RGB(r, g, b)

    24-bit integer color representation

- #define MAPS_RGBA(r, g, b, a)

    24-bit integer color representation + 8-bit alpha value (transparency)

- #define MAPS_PI

    The Pi number.

- #define MAPS_DEG2RAD(x)

    Degrees to radians conversion macro.

- #define MAPS_SAFE_DELETE(p)

    Standard macro for safely deleting a pointer to an array.

- #define MAPS_SAFE_DELETE_ARRAY(p)

    Standard macro for safely deleting a single-element pointer.

- #define MAPSForallItems(it, L)
- #define MAPSForall(x, L)
- #define MAPSForallPtr(x, L)
- #define MAPS_BEGIN_INPUTS_DEFINITION(className)

    Starts the definition of the inputs of a module.

- #define MAPS_INPUT(namex, filter, typex)

    Basic definition of an input.

- #define MAPS_END_INPUTS_DEFINITION

    Ends the definition of the inputs of a module.

- #define MAPS_BEGIN_OUTPUTS_DEFINITION(className)

    Starts the definition of the outputs of a module.

- #define MAPS_OUTPUT(name, value, namex, unit, size)

    Basic definition of an output.

- #define MAPS_OUTPUT_FIFOSIZE(name, value, namex, unit, size, fifosize)

    Definition of an output with control of the FIFO size.

- #define MAPS_OUTPUT_USER_STRUCTURE(name, structureName)

    Definition of an output using a user structure.

- #define MAPS_END_OUTPUTS_DEFINITION

    Ends the definition of the outputs of a module.

- #define MAPS_BEGIN_PROPERTIES_DEFINITION(className)

  Starts the definition of the properties of a module.

- #define MAPS_PROPERTY(name, value, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_UNIT(name, value, unit, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_ENUM(name, enumstr, selected, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_ENUM_UNIT(name, enumstr, selected, unit, needs2BeInitialized, canBeChangedAfterInstantiation)

  Basic definition of a property.

- #define MAPS_PROPERTY_READ_ONLY(name, value)

  Definition of a read-only property.

- #define MAPS_PROPERTY_READ_ONLY_UNIT(name, value, unit)

  Definition of a read-only property with a specified unit.

- #define MAPS_END_PROPERTIES_DEFINITION

  Ends the definition of the properties of a module.

- #define MAPS_BEGIN_ACTIONS_DEFINITION(className)

  Starts the definition of the actions of a module.

- #define MAPS_ACTION(name, proc)

  Basic definition of an action.

- #define MAPS_ACTION2(name, proc, allowedWhenDead)

  Second definition of an action.

- #define MAPS_END_ACTIONS_DEFINITION

  Ends the definition of the actions of a module.

- #define MAPS_PACKAGE_DEFINITION(name, version)

  Use this macro to determine the version of your package file.

- #define MAPS_RECORD_REPLAY_METHOD_DEFINITION(rrm, namex, version, filter, recordThreaded, replayThreaded, nbPropertiesRecord, nbPropertiesReplay)

Required for the implementation of an RTMaps record/replay method.

- #define MAPS_RECORD_REPLAY_METHOD_STANDARD_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition.

- #define MAPS_RECORD_REPLAY_METHOD_BLACKBOX_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you intend to implement blackbox features.

- #define MAPS_RECORD_REPLAY_METHOD_PROCESS_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you need preprocessing before recording.

- #define MAPS_RECORD_REPLAY_METHOD_COPY_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you intend to implement special copy features.

- #define MAPS_RECORD_REPLAY_METHOD_HEADING_HINT_HEADER_CODE(rrm)

  Include this macro inside your RRM class definition if you intend to implement hint header.

- #define MAPSFilterUserStructure(structureName)

  Defines for type filters.

Typedefs

- typedef MAPSInt32 MAPSInteger

  The MAPSInteger should be a 32-bit integer, even on 64-bit architecture.

- typedef MAPSFloat64 MAPSFloat

  The MAPSFloat type is a double precision floating point number.

- typedef MAPSInt64 MAPSTimestamp

  The MAPSTimestamp type is a 64-bit integer.

- typedef MAPSInt64 MAPSDelay

  A 64-bit integer that specifies a delay in microseconds.

- typedef MAPSInt64 MAPSTypeInfoValue

  The MAPSTypeInfoValue type is a 64 bit-integer. All types in RTMaps are associated to a 64-bit value.

- typedef MAPSInt32 MAPS3States

  3-state value, generally used to tell the state of a thread.

- typedef MAPSStackedString< 40 > MAPSString

  The RTMaps flexible and fast string class.

Component definition macros

- #define MAPS_COMPONENT_DEFINITION(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions)

  One of the standard macros required for the definition of an RTMaps component.

- #define MAPS_COMPONENT_DEFINITION_DOC(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions, doc)

  One of the standard macros required for the definition of an RTMaps component.

- #define MAPS_COMPONENT_DEFINITION_UNIQUE(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions)

  One of the standard macros required for the definition of an RTMaps component.

- #define MAPS_COMPONENT_DEFINITION_REGISTRATION(component, name, version, priority, kind, defaultBehaviour, nbOfInputs, nbOfOutputs, nbOfProperties, nbOfActions)

  One of the standard macros required for the definition of an RTMaps component.

Component class declaration macros

- #define MAPS_COMPONENT_STANDARD_HEADER_CODE(component)
- #define MAPS_CLOCK_COMPONENT_HEADER_CODE(component)
- #define MAPS_COMPONENT_DYNAMIC_HEADER_CODE(component)
- #define MAPS_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(component)
- #define MAPS_COMPONENT_REGISTERING_HEADER_CODE(component)
- #define MAPS_CHILD_COMPONENT_HEADER_CODE(component, parent)

  Include this macro inside your component class definition if its parent is a descendant of MAPSComponent.

- #define MAPS_CHILD_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(component, parent)

  Include this macro inside your component class definition if its parent is a descendant of MAPSComponent and you need to implement your own constructor.

- #define MAPS_PARENT_COMPONENT_HEADER_CODE(component, parent)

  Include this macro inside a parent component class definition.

- #define MAPS_PARENT_COMPONENT_HEADER_CODE_WITHOUT_CONSTRUCTOR(component, parent)

  Include this macro inside a parent component class definition when you need to implement your own constructor.

RTMaps Typing related constants

- const MAPSTypeInfoValue MAPS::NoMask

  No mask.

- const MAPSTypeInfoValue MAPS::NoType

  No type.

- const MAPSTypeInfoValue MAPS::AnyType

  Any type.

- const MAPSTypeInfoValue MAPS::TypeMask

  Type mask.

- const MAPSTypeInfoValue MAPS::Structure

  User defined structure.

- const MAPSTypeInfoValue MAPS::Integer

  Integer (32 bits, signed)

- const MAPSTypeInfoValue MAPS::Float

  MAPSFloat (double : 64 bits, double precision)

- const MAPSTypeInfoValue MAPS::TextAscii

  Ascii characters (1 character = 8 bits).

- const MAPSTypeInfoValue MAPS::TextUnicode

  Unicode characters (1 character = 16 bits)

- const MAPSTypeInfoValue MAPS::IplImage

  IplImage (Image Processing Library image description structure)

- const MAPSTypeInfoValue MAPS::MAPSImage

  MAPSImage (RTMaps specific image structure)

- const MAPSTypeInfoValue MAPS::CANFrame

CAN frame.

- const MAPSTypeInfoValue MAPS::Matrix

  MATLAB-Like matrix.

- const MAPSTypeInfoValue MAPS::RealObject

  Real object (car, tree, etc.)

- const MAPSTypeInfoValue MAPS::DrawingObject

  Drawing object (line, circle, etc.)

- const MAPSTypeInfoValue MAPS::Triangles3D

  3D triangles for 3D scene rendering

- const MAPSTypeInfoValue MAPS::Stream8

  8-bit stream (sound, numerized data)

- const MAPSTypeInfoValue MAPS::Stream16

  16-bit stream (sound, numerized data)

- const MAPSTypeInfoValue MAPS::Stream32

  32-bit stream (numerized data)

- const MAPSTypeInfoValue MAPS::Integer64

  Integer (64 bits, signed)

- const MAPSTypeInfoValue MAPS::AnyText

  Some textual information, ascii or unicode.

- const MAPSTypeInfoValue MAPS::AnyInteger

  32 or 64 bits integers.

- const MAPSTypeInfoValue MAPS::VectorFlag

  Indicates that the piece of data is a vector (an array) of a basic type.

- const MAPSTypeInfoValue MAPS::FrequencyFlag

  Indicates that a frequency is provided with the data.

- const MAPSTypeInfoValue MAPS::QualityFlag

  Indicates that a quality is transmitted along with the data (for instance a compression or a noise ratio).

- const MAPSTypeInfoValue MAPS::MiscFlag

  Indicates that 3 miscellaneous integers are transmitted along with the data. See MAPSIOElt::Misc1(), Misc2() and Misc3()

RTMaps Inputs behaviours

- const int MAPS::FifoReader

  FIFO Reader behaviour.

- const int MAPS::NeverSkippingReader

  Never Skipping Reader behaviour.

- const int MAPS::LastOrNextReader

  Last or Next Reader behaviour.

- const int MAPS::Wait4NextReader

  Wait For Next Reader behaviour.

- const int MAPS::SamplingReader

  Sampling Reader behaviour.

RTMaps Properties types

- const int MAPS::BoolProperty

  Boolean property (`false` or `true`)

- const int MAPS::IntegerProperty

  Integer property (64-bit signed integer)

- const int MAPS::FloatProperty

  Floating point property (`double`: 64-bit floating point number)

- const int MAPS::StringProperty

  String property (ASCII string) Enum property (ASCII string)

- const int MAPS::EnumProperty

RTMaps Type Filters

- const MAPSTypeFilterBase MAPS::FilterStructure

  Filters any user-defined structure type.

- const MAPSTypeFilterBase MAPS::FilterInteger

  Filters integer type (same as FilterIntegers)

- const MAPSTypeFilterBase MAPS::FilterInteger64

  Filters 64 bits integer type (same as FilterIntegers64)

- const MAPSTypeFilterBase MAPS::FilterFloat

  Filters MAPSFloat type (same as FilterFloats)

- const MAPSTypeFilterBase MAPS::FilterNumber

  Filters integer or MAPSFloat type (same as FilterNumbers)

- const MAPSTypeFilterBase MAPS::FilterIntegers

  Filters integer scalars or vectors.

- const MAPSTypeFilterBase MAPS::FilterIntegers64

  Filters 64 bits integer scalars or vectors.

- const MAPSTypeFilterBase MAPS::FilterFloats

  Filters MAPSFloat scalars or vectors.

- const MAPSTypeFilterBase MAPS::FilterNumbers

  Filters integer or MAPSFloat scalars or vectors.

- const MAPSTypeFilterBase MAPS::FilterOneInteger

  Filters integer type (excludes vectors of integers)

- const MAPSTypeFilterBase MAPS::FilterOneInteger64

  Filters 64 bits integer type (excludes vectors of integers64)

- const MAPSTypeFilterBase MAPS::FilterOneFloat

  Filters MAPSFloat type (excludes vectors of MAPSFloat)

- const MAPSTypeFilterBase MAPS::FilterOneNumber

  Filters integer or MAPSFloat type (excludes vectors)

- const MAPSTypeFilterBase MAPS::FilterTextAscii

  Filters ASCII text string.

- const MAPSTypeFilterBase MAPS::FilterTextUnicode

  Filters Unicode (16 bits) text string.

- const MAPSTypeFilterBase MAPS::FilterImage

  Filters IplImages or MAPSImages.

- const MAPSTypeFilterBase MAPS::FilterIplImage

  Filters IplImages.

- const MAPSTypeFilterBase MAPS::FilterMAPSImage

  Filters MAPSImages.

- const MAPSTypeFilterBase MAPS::FilterCANFrame

Filters CANFrames.

- const MAPSTypeFilterBase MAPS::FilterMatrix

  Filters MATLAB-Like matrices.

- const MAPSTypeFilterBase MAPS::FilterRealObjects

  Filters RTMaps Real Objects.

- const MAPSTypeFilterBase MAPS::FilterDrawingObjects

  Filters RTMaps Drawing Objects.

- const MAPSTypeFilterBase MAPS::FilterTriangles3D

  Filters 3D triangles.

- const MAPSTypeFilterBase MAPS::FilterStream8

  Filters 8-bit data streams.

- const MAPSTypeFilterBase MAPS::FilterStream16

  Filters 16-bit data streams.

- const MAPSTypeFilterBase MAPS::FilterStream32

  Filters 32-bit data streams.

- const MAPSTypeFilterBase MAPS::FilterAudioSignal

  Filters audio signals (either MAPSFloat or Stream8)

- const MAPSTypeFilterBase MAPS::FilterAny

  Filters any kind of data.

RTMaps kinds of information outputs

- const int MAPS::Info

  Simple information.

- const int MAPS::Warning

  Warning.

- const int MAPS::Error

  Error.

- const int MAPS::ParserEcho

  Echo from the parser.

RTMaps replay modes

- const int MAPS::ReplayModeNormal

  Normal replay.

- const int MAPS::ReplayModeImmediate

  Immediate replay mode (replay ahead of real time)

- const int MAPS::ReplayModeTimestamp

  Replay using timestamp instead of time of issue.

RTMaps VCR Keys codes

- const int **MAPS::VCRKeyStop**
- const int **MAPS::VCRKeyPlay**
- const int **MAPS::VCRKeyRecord**
- const int **MAPS::VCRKeyPause**
- const int **MAPS::VCRKeyRewind**
- const int **MAPS::VCRKeyForward**
- const int **MAPS::VCRKeyNext**
- const int **MAPS::VCRKeyOrganize**
- const int **MAPS::VCRSlider**
- const int **MAPS::VCRAllKeys**

RTMaps Engine Keys codes

- const int **MAPS::KernelKeyRun**
- const int **MAPS::KernelKeyShutdown**
- const int **MAPS::DefaultKeysState**

Others RTMaps Constants

- const MAPSString MAPS::OperatingSystem

  Contains the operating system string information ("Win32","QNX" or "Linux" for instance)

- const int MAPS::OSBuild

  Contains the OS support build number information.

- const MAPSString MAPS::Distribution

  Contains the distribution information string ("3.0" for instance)

- const MAPSString MAPS::KernelVersion

  Contains the engine version string information ("1.0" for instance)

- const MAPSString MAPS::RTMapsMinorVersion

  Contains the RTMaps minor version (changes in the minor versions preserve backward compatibility with packages). "5" for example -> complete version string will look like "3.0.5".

- const int MAPS::KernelBuild

  Contains the engine build number information.

- const MAPSString MAPS::Copyright

  Contains the RTMaps copyright string information.

- const MAPSString MAPS::License

  Contains the RTMaps license grant string information (depends on the customer)

- const MAPSString MAPS::ProductName

  Contains the operating system string information ("Win32","QNX" or "Linux" for instance)

- const bool MAPS::BigEndian

  Tells if we are running on a big-endian platform (`true`) or a little-endian platform (`false`)

- const int MAPS::DefaultTextSize

  Default allocation size (in characters) for ascii text type outputs.

- const int MAPS::Infinite

  Infinite number.

- const int MAPS::ModuleDied

  State indicating a dead RTMaps module.

- const int MAPS::GotAMessage

  State indicating that RTMaps got a Windows message.

- const int MAPS::TimeOut

  State indicating a time out in RTMaps.

- const int MAPS::FatalKernelError

  State indicating a fatal engine error.

- const int MAPS::ErrorException

  State indicating an error in an RTMaps module.

- const int MAPS::Running

  State telling that RTMaps is running (that time is flowing)

- const int MAPS::Paused

  State telling that RTMaps is in pause mode.

- const int MAPS::ShuttingDown

  RTMaps is currently shutting down.

- const int MAPS::Resetting

  RTMaps is currently resetting.

- const int MAPS::WaitingForSynchBeforeRun

  RTMaps is currently waiting to be synchronized before running.

- const int MAPS::DeadState

  State indicating a dead thread or module.

- const int MAPS::DyingState

  State indicating a dying thread or module.

- const int MAPS::LivingState

  State indicating a living thread or module.

- const int MAPS::Threaded

  The component is threaded.

- const int MAPS::Sequential

  The component is sequential.

C standard library wrapper

These functions should be called instead of their C counterparts to ensures easier cross-platform design of components.

- int MAPS::Atoi (const char ∗a)

  Ascii to integer conversion.

- MAPSInt64 MAPS::Atoi64 (const char ∗a)

  Ascii to integer (64 bits) conversion.

- MAPSInt32 MAPS::Strtol (const char ∗s, char ∗∗endptr, int base)

  Converts the initial part of the string in s to a MAPSInt32 according to the given base.

- MAPSUInt32 MAPS::Strtoul (const char ∗s, char ∗∗endptr, int base)

  Converts the initial part of the string in s to a MAPSUInt32 according to the given base.

- MAPSInt64 MAPS::Strtoi64 (const char ∗s, char ∗∗endptr, int base)

Converts the initial part of the string in s to a MAPSInt64 according to the given base.

- MAPSUInt64 MAPS::Strtoui64 (const char ∗s, char ∗∗endptr, int base)

  Converts the initial part of the string in s to a MAPSUInt64 according to the given base.

- int MAPS::Strlen (const char ∗s)

  Calculates the length of the string s.

- const char ∗ MAPS::Strchr (const char ∗s, char ch)

  Finds character ch in s.

- char ∗ MAPS::Strchr (char ∗s, char ch)

  Finds character ch in s.

- int MAPS::Strcmp (const char ∗s1, const char ∗s2)

  String comparison.

- int MAPS::Stricmp (const char ∗s1, const char ∗s2)

  Lowercase comparison of strings.

- int MAPS::Strncmp (const char ∗s1, const char ∗s2, MAPSInt64 size)

  Compares the first size characters of two strings.

- char ∗ MAPS::Strstr (const char ∗s, const char ∗strSearch)

  Returns a pointer to the first occurrence of a search string strSearch in the string s.

- char ∗ MAPS::Strcpy (char ∗strDest, const char ∗strSrc)

  Copies strSrc into strDest.

- char ∗ MAPS::Strdup (const char ∗strSrc)

  Duplicate strings.

- char ∗ MAPS::Itoa (int val, char ∗buf, int radix=10)

  Integer to Ascii conversion.

- char ∗ MAPS::Itoa (unsigned long val, char ∗buf, int radix=10)

  Integer to Ascii conversion.

- char ∗ MAPS::Itoa (MAPSInt64 val, char ∗buf, int radix=10)

  Integer to Ascii conversion.

- bool MAPS::IsSpace (char c)

  Is the character c a space, a tab or a carriage return?

- bool MAPS::IsDigit (char c)

  Is the character c a digit?

- void ∗ MAPS::Memcpy (void ∗dest, const void ∗source, MAPSInt64 size, MAPSEvent ∗event=NULL)

  Memory copy.

- void ∗ MAPS::Memset (void ∗dest, int c, MAPSInt64 size)

  Sets buffers to a specified character c, repeated size times.

- void ∗ MAPS::Memmove (void ∗dest, const void ∗source, MAPSInt64 size)

  Moves one buffer to another.

- double MAPS::Modf (double x, double ∗intptr)

  Splits a floating-point value into fractional and integer parts.

- double MAPS::Fabs (double x)

  Calculates the absolute value of the floating-point argument.

RTMaps Main functions

- void MAPS::Exit ()

  Required after any use of RTMaps...

- bool MAPS::Run ()

  Starts the execution of the current session.

- bool MAPS::Shutdown ()

  Shutdowns the RTMaps session currently running.

- bool MAPS::Reset ()

  Resets the RTMaps system.

- bool MAPS::Parse (const char ∗s)

  Parses a string containing MAPSScript instructions.

- bool MAPS::ParseFile (const char ∗s)

  Parses a file containing MAPSScript instructions.

- void MAPS::LoadCoreFunction (const char ∗cf)

  Loads a core function and activates it.

- MAPSCoreFunctionInterface ∗ MAPS::CoreFunction (const char ∗cf)

  Returns a pointer to the core function named `cf` if it was previously loaded (returns NULL otherwise)

- void MAPS::RegisterPackage (const char ∗fileName)

Loads a package (set of components compiled as a shared object).

- void MAPS::UnregisterPackage (const char ∗fileName)

  Unloads a package (set of components compiled as a shared object).

- bool MAPS::IsRunning ()

  Is there a RTMaps system currently running?

- bool MAPS::IsPaused ()

  Is the RTMaps clock in paused state ?

- bool MAPS::IsReplaying ()

  Returns true if a file is open for replay.

- int MAPS::CheckLevel ()

  Returns the level of structure control in the current system (0=no control, fastest, 2=max control, slowest)

RTMaps functions for distribution and synchronization

- bool MAPS::IsDistributedAsMaster ()

  Is RTMaps acting as a Master on a network of distributed RTMaps.

- bool MAPS::IsDistributedAsSlave ()

  Is RTMaps acting as a Slave on a network of distributed RTMaps.

- bool MAPS::GetSynchronizer (void ∗owner, MAPSSynchronizer ∗∗ppSynchronizer)

  Requests a pointer on the synchronizer object.

- bool MAPS::ReleaseSynchronizer (void ∗module, MAPSSynchronizer ∗∗ppSynchronizer)

  Releases a pointer on the synchronizer object.

RTMaps Virtual Time management functions

- MAPSAbsoluteTime & MAPS::TimeReference ()

  Gets the time reference (the absolute time the timestamps always refer to)

- MAPSTimestamp MAPS::CurrentTime (bool lock=true, bool release=true)

  Gets the current time in the RTMaps system (virtual time)

- void MAPS::SetCurrentTime (MAPSTimestamp t)

  Sets the current time (jumps in time!)

- void MAPS::SetTimeSpeed (int speed)

  Sets the current time speed (1000 = real time)

- int MAPS::TimeSpeed ()

  Gets the current time speed (1000 = real time)

- int MAPS::TimeState ()

  Gets the current time state (MAPS::Running or MAPS::Paused)

RTMaps Index management

- void MAPS::SetIndex ()

  Adds an index now (during recording)

- void MAPS::Go2Index (int num)

  Goes to index num during replay.

- void MAPS::Go2PreviousIndex ()

  Goes to previous index (during replay)

- void MAPS::Go2NextIndex ()

  Goes to next index (during replay)

RTMaps Recording management

- void MAPS::StartRecording (void)

  Starts recording information (starts all)

- void MAPS::StopRecording (void)

  Stops recording information (stops all)

RTMaps Type Filtering management

- bool MAPS::TypeFilter (const MAPSTypeInfo &outputType, MAPSTypeFilterBase &filter)

RTMaps General purpose functions

- ::IplImage MAPS::IplImageModel (int width, int height, unsigned int channelSeq=MAPS_-CHANNELSEQ_BGR, unsigned int dataOrder=IPL_DATA_ORDER_PIXEL, unsigned int depth=IPL_DEPTH_8U)

Creates a model of an operational IplImage structure with the provided parameters.

- ::IplImage MAPS::IplImageModel (int width, int height, const char ∗channelSeq, unsigned int dataOrder=IPL_DATA_ORDER_PIXEL, unsigned int depth=IPL_DEPTH_-8U)

  Creates a model of an operational IplImage structure with the provided parameters.

- bool MAPS::IplImageCheck (::IplImage &image,::IplImage &model)

  Checks that an image is the same type and size as the model given in second parameter.

RTMaps Maintenance functions

- MAPSString MAPS::About (void)

RTMaps Reporting functions

For all these functions, the importance must be set between 0 (not important) to 2 (of the utmost importance)

- void MAPS::ReportInfo (const char ∗text, int importance=0)

  Reports a piece of information.

- void MAPS::ReportWarning (const char ∗text, int importance=0)

  Reports a warning.

- void MAPS::ReportError (const char ∗text, int importance=0)

  Reports an error.

- void MAPS::Report (const char ∗text, int type, int importance=0)

  Reports something (with user feedback + logging)

- void MAPS::MessageBox (const char ∗message, int type, int importance=0)

  Displays a modal MessageBox. To be used for debugging only since this function blocks until the user.

Time conversion and handling functions

- MAPSTimestamp MAPS::TimestampFromString (const char ∗s, char ∗∗endptr=NULL)

  Transforms a string of form hh:mm:ss.mmmuuu into a timestamp.

- MAPSString MAPS::TimeString (MAPSTimestamp t)

- MAPSString MAPS::Timestamp2String (MAPSTimestamp t)

  Transforms a timestamp into a human readable string of form hh:mm:ss.mmmmmm.

- void MAPS::Timestamp2AbsoluteTimeUTC (MAPSTimestamp t, MAPSAbsoluteTime ∗utctime)

  Transforms a timestamp in microseconds into an absolute time.

- MAPSTimestamp MAPS::AbsoluteTimeUTC2Timestamp (const MAPSAbsoluteTime ∗utctime)

  Transforms an absolute time into a timestamp.

- MAPSInt64 MAPS::AbsoluteTime2Integer (MAPSAbsoluteTime &time)
- void MAPS::Integer2AbsoluteTime (MAPSInt64 integer, MAPSAbsoluteTime &time)
- MAPSString MAPS::AbsoluteTime2String (MAPSAbsoluteTime absTime)

  Transforms an absolute time into a string of form yyyy/mm/dd hh:mm:ss.mmmuuu.

- void MAPS::GetAbsoluteTime (MAPSAbsoluteTime ∗localtime)

  Gets the absolute current time (local time). Deprecated. Prefer using MAPS::GetAbsoluteTimeLocal.

- void MAPS::GetAbsoluteTimeLocal (MAPSAbsoluteTime ∗localtime)

  Gets the absolute current local time.

- void MAPS::GetAbsoluteTimeUTC (MAPSAbsoluteTime ∗utctime)

  Gets the absolute current time (UTC)

- bool MAPS::AbsoluteTimeUTCToAbsoluteTimeLocal (const MAPSAbsoluteTime ∗utctime, MAPSAbsoluteTime ∗localtime)

  Converts a UTC time to a local time. Returns true in case of success, false otherwise.

- bool MAPS::AbsoluteTimeLocalToAbsoluteTimeUTC (const MAPSAbsoluteTime ∗localtime, MAPSAbsoluteTime ∗utctime)

  Converts a local time to a UTC time. Returns true in case of success, false otherwise.

- bool MAPS::SetSystemTime (const MAPSAbsoluteTime ∗utctime)

  Sets the computer system clock to the provided absolute time.

RTMaps Flow monitoring functions

- MAPSString MAPS::GetWriteStatistics ()

  Gets the detailed statistics about the data flow to the hard disks (write file operations).

- MAPSString MAPS::GetReadStatistics ()

  Gets the detailed statistics about the read file operations.

- int MAPS::GetRemainingTime ()

Gets the overall remaining recording time, if the flows stays constant.

- MAPSInt64 MAPS::GetWriteFlow ()

  Gets the total write flow (file write operations)

- MAPSInt64 MAPS::GetReadFlow ()

  Gets the total read flow (file read operations)

- void MAPS::RecordMemoryWriteFlow (MAPSInt64 size)

  Notifies RTMaps that such a memory write flow has occured. Do not use in conjonction with MAPS::Memcpy(...)

- void MAPS::RecordMemoryReadFlow (MAPSInt64 size)

  Notifies RTMaps that such a memory read flow has occured. Do not use in conjonction with MAPS::Memcpy(...)

- MAPSInt64 MAPS::GetMemoryReadFlow ()

  Gets the total memory read flow.

- MAPSInt64 MAPS::GetMemoryWriteFlow ()

  Gets the total memory write flow.

- MAPSInt64 MAPS::GetMemoryFlow ()

  Gets the total memory flow (read+write)

- MAPSInt64 MAPS::GetDiskFreeSpace (const char ∗path)

  Gets the free disk space on the disk containing `path`.

RTMaps OS wrapping functions

These functions give access to all the needed features of an OS except file I/O. These functions should be called instead of their OS-specific counterparts to ensure a cross-platform programming.

- MAPSTimestamp MAPS::GetSystemAccurateTiming (void)

  Gets an immediate timestamp (in microseconds).

- bool MAPS::CreateThread (void ∗(∗startAdress)(void ∗), void ∗argList)

  Starts a thread.

- void MAPS::SetCurrentThreadPriority (int priority)

  Sets the priority of the current thread (between 0 and 255)

- MAPSThreadId MAPS::GetCurrentThreadId ()

  Gets an id for the current thread.

- void MAPS::ReleaseCurrentThread (void)

    Releases the current thread.

- void MAPS::Sleep (MAPSDelay delay)

    Sleeps for a certain amount of time.

- void MAPS::Wait4AWhile (void)

    Waits for a while. Useful little function.

- MAPSString MAPS::GetCurrentDirectory ()

    Gets the current directory path.

- bool MAPS::SetCurrentDirectory (const MAPSString path)

    Sets the current directory path.

- bool MAPS::CreateDirectory (const char ∗dirName)

    Creates a directory.

- MAPSString MAPS::GetTempDirectory ()

    Gets a path to the temporary directory.

- MAPSString MAPS::GetInstallPath (const char ∗pathName)

    Gets a path refering to RTMaps installation.

- void ∗ MAPS::AllocSharedMemory (int size)

    Memory allocation. Assumes it is allocated in a way that it can be accessed by a RTMaps system running in another process.

- void MAPS::DeallocSharedMemory (void ∗ptr)

    Memory deallocation. Assumes it was allocated in a way that it can be accessed by a RTMaps system running in another process.

RTMaps C++ interface functions

These functions are designed to control the RTMaps engine directly through C++ calls. These are rather low-level functions that are not of any interest for a component developer.

- bool MAPS::GetBoolProperty (const char ∗s, bool ∗ok=NULL)

    Gets a boolean property.

- MAPSInt64 MAPS::GetIntegerProperty (const char ∗s, bool ∗ok=NULL)

    Gets an integer property or enum property selected index.

- MAPSString MAPS::GetStringProperty (const char ∗s, bool ∗ok=NULL)

    Gets a string property or enum property selected string.

- MAPSFloat MAPS::GetFloatProperty (const char ∗s, bool ∗ok=NULL)

    Gets a float property.

- MAPSEnumStruct MAPS::GetEnumsProperty (const char ∗s, bool ∗ok=NULL)

    Gets an enum property.

- MAPSComponent ∗ MAPS::Component (const char ∗s)

    Returns a pointer to component `s` *(returns NULL if it does not exist)*

- MAPSProperty ∗ MAPS::Property (const char ∗s)

    Returns a pointer to the property named `s` *(returns NULL if it does not exist)*

- MAPSComponent ∗ MAPS::CreateComponent (const char ∗modelName, const char ∗componentName, bool start=true)

    Component instantiation.

- void MAPS::StartComponent (const char ∗componentName)

    Starts the component if frozen.

- void MAPS::KillComponent (const char ∗componentName)

    Dynamically destroys a component.

- void MAPS::KillComponent (MAPSComponent &component)

    Dynamically destroys a component.

- void MAPS::RenameComponent (const char ∗componentName, const char ∗newName)

    Renames a component.

- void MAPS::Attach (MAPSOutput &output, MAPSInput &input)

    Dynamically attaches an input to an output.

- void MAPS::Attach (const char ∗outputName, const char ∗inputName)

    Dynamically attaches an input to an output.

- bool MAPS::Attach2 (const char ∗name, const char ∗inputName)

    Dynamically attaches an input to an output. Extended version.

- void MAPS::Detach (MAPSOutput &output, MAPSInput &input)

    Dynamically detaches an output from an input.

- void MAPS::Record (const char ∗outputName, const char ∗recorder=NULL, const char ∗method=NULL, bool neverskipping=false, bool useTimestamp=false)

Records an output.

- void MAPS::Open (const char ∗pattern, const char ∗nspace=NULL, MAPSInt64 offset=0, MAPSTimestamp beginning=0, MAPSTimestamp end=0)

  Opens a database to replay records from.

- void MAPS::Replay (const char ∗outputname)

  Replays some data.

- void MAPS::Copy (const char ∗outputname, const char ∗recorderName)

  Copies some data.

- void MAPS::StopCopy (const char ∗outputname)

  Aborts the copy of an output.

- void MAPS::SetTimeAdapt (int ta)

  Sets the time automatic adaptation algorithms.

- MAPSTimestamp MAPS::GetFirstTimestamp ()

  Returns the first timestamp of all the currently open databases.

- MAPSTimestamp MAPS::GetLastTimestamp ()

  Returns the last timestamp of all the currently open databases.

- MAPSEvent ∗ MAPS::GetShutdownEvent ()

  Gets a pointer to the shutdown event.

11.1.1 Detailed Description

The RTMaps engine header file - Version 3.4 build 99.

11.1.2 Define Documentation

11.1.2.1 #define MAPS_PACKAGE_DEFINITION(*name*, *version*)

Use this macro to determine the version of your package file.

Use this macro in one (and only one) of your package .cpp files in order to set a version for your RTMaps .pck file. If this macro is not present, the version of the package is 1.0.

11.1.2.2 #define MAPS_PI

The Pi number.

You should use this instead of M_PI, which is not defined on some plateforms.

11.1.2.3 #define MAPS_RGB(*r*, *g*, *b*)

24-bit integer color representation

Stricly equivalent to the Windows RGB macro. Please use this one in place of RGB for better portability of the code.

11.1.3 Typedef Documentation

11.1.3.1 typedef MAPSInt32 MAPS3States

3-state value, generally used to tell the state of a thread.

It can take the MAPS::DeadState, MAPS::DyingState and MAPS::LivingState values.

11.1.3.2 typedef double MAPSFloat

The MAPSFloat type is a double precision floating point number.

RTMaps always uses MAPSFloat (`double` = 64-bit floating point numbers). Please never use `float` (`float` = 32 bits floating point numbers)

11.1.3.3 typedef MAPSInt64 MAPSTimestamp

The MAPSTimestamp type is a 64-bit integer.

RTMaps always timestamps data with a 64-bit integer representing the amount of time in microseconds since the last Run command execution (the reference time)

11.2 MAPSImageProcessing1Src1Dest.h File Reference

The MAPSImageProcessing1Src1Dest component model.

Classes

- struct MAPSImageProcessing1Src1DestDefinition

  Definition structure for an image processing algorithm (1 Src 1 Dest operation).

- struct MAPSImageProcessing1Src1DestParams

  Parameter structure for an image processing algorithm (1 Src 1 Dest operation).

- class MAPSImageProcessing1Src1Dest

  Image Processing component base class with 1 input and 1 output images.

11.2.1 Detailed Description

The MAPSImageProcessing1Src1Dest component model.

11.3 MAPSImageProcessing2Src1Dest.h File Reference

The MAPSImageProcessing2Src1Dest component model.

Classes

- struct MAPSImageProcessing2Src1DestDefinition

  Definition structure for an image processing algorithm (2 Src 1 Dest operation).

- struct MAPSImageProcessing2Src1DestParams

  Parameter structure for an image processing algorithm (2 Src 1 Dest operation).

- class MAPSImageProcessing2Src1Dest

  Image Processing component base class with 2 input and 1 output images.

11.3.1 Detailed Description

The MAPSImageProcessing2Src1Dest component model.

11.4 MAPSRBRegion.h File Reference

The RTMaps Ring-buffer region class and associated types.

Classes

- class MAPSRBRegion

  MAPSRBRegion class : Ring-Buffer region manipulation class.

Enumerations

- enum MAPSConstRBRegionState {

  MAPSConstRBRegionEmpty, MAPSConstRBRegionReady, MAPSConstRBRegionWarnings, MAPSConstRBRegionUninitialized,

  MAPSConstRBRegionOverflowOp, MAPSConstRBRegionErrors, MAPSConstRBRegionInvalid, MAPSConstRBRegionInvalidOp }

Flags for ring-buffer region state.

- enum MAPSConstRBRegionErosion { MAPSConstRBRegionErosionBeginning, MAPSConstRBRegionErosionEnd }

    Flags for the MAPSRBRegion::Erode method.

- enum MAPSConstRBRegionDilation { MAPSConstRBRegionDilationBeginning, MAPSConstRBRegionDilationEnd }

    Flags for the MAPSRBRegion::Dilate method.

- enum MAPSConstRBRegionCrop { MAPSConstRBRegionCropBeginning, MAP-SConstRBRegionCropEnd }

    Flags for the MAPSRBRegion::Crop method.

- enum MAPSConstRBRegionSubRegion { MAPSConstRBRegionSubRegionFromBe-ginning, MAPSConstRBRegionSubRegionFromEnd }

    Flags for the MAPSRBRegion::SubRegion method.

- enum MAPSConstRBRegionBesideRegion { MAPSConstRBRegionBesideRegion-Previous, MAPSConstRBRegionBesideRegionNext }

    Flags for the MAPSRBRegion::BesideRegion method.

11.4.1 Detailed Description

The RTMaps Ring-buffer region class and associated types.

11.4.2 Enumeration Type Documentation

11.4.2.1 enum MAPSConstRBRegionBesideRegion

Flags for the MAPSRBRegion::BesideRegion method.

Enumerator:

MAPSConstRBRegionBesideRegionPrevious Take the region before the source region.

MAPSConstRBRegionBesideRegionNext Take the region after the source region.

11.4.2.2 enum MAPSConstRBRegionCrop

Flags for the MAPSRBRegion::Crop method.

Enumerator:

MAPSConstRBRegionCropBeginning Crop from the beginning of the region.

MAPSConstRBRegionCropEnd Crop from the end of the region.

11.4.2.3 enum MAPSConstRBRegionDilation

Flags for the MAPSRBRegion::Dilate method.

Enumerator:

MAPSConstRBRegionDilationBeginning Dilate from the beginning of the region.

MAPSConstRBRegionDilationEnd Dilate from the end of the region.

11.4.2.4 enum MAPSConstRBRegionErosion

Flags for the MAPSRBRegion::Erode method.

Enumerator:

MAPSConstRBRegionErosionBeginning Erode from the beginning of the region.

MAPSConstRBRegionErosionEnd Erode from the end of the region.

11.4.2.5 enum MAPSConstRBRegionState

Flags for ring-buffer region state.

Enumerator:

MAPSConstRBRegionEmpty When the region has been initialized, but is empty.

MAPSConstRBRegionReady When the region has been initialized and contains some data.

MAPSConstRBRegionWarnings Warnings common flag.

MAPSConstRBRegionUninitialized When the region has not been initialized.

MAPSConstRBRegionOverflowOp When the region has been initialized, is full, and the last operation returned an overflow. This could never be the state of a region, but the state of an operation on a region.

MAPSConstRBRegionErrors Errors common flag.

MAPSConstRBRegionInvalid When the region is invalid (offset, size, fifoSize are inconsistent)

MAPSConstRBRegionInvalidOp Only returned by an operation when the operation is invalid on the current region. This could never be the state of a region, but the state of an operation on a region.

11.4.2.6 enum MAPSConstRBRegionSubRegion

Flags for the MAPSRBRegion::SubRegion method.

Enumerator:

MAPSConstRBRegionSubRegionFromBeginning Extract the sub-region from the beginning of the source region.

MAPSConstRBRegionSubRegionFromEnd Extract the sub-region from the end of the source region.

11.5 MAPSSoundProcessingComponent.h File Reference

The MAPSSoundProcessingComponent component model Format of IO element:

- Frequency() holds frequency info (!!! in mHz !!!!)
- Misc1() holds the number of channels
- Misc2() holds the bits info.

11.5.1 Detailed Description

The MAPSSoundProcessingComponent component model Format of IO element:

- Frequency() holds frequency info (!!! in mHz !!!!)
- Misc1() holds the number of channels
- Misc2() holds the bits info.

11.6 MAPSStream8IOComponent.h File Reference

The MAPSStream8IOComponent component model.

Classes

- class MAPSStream8IOComponent

  Processing component template with 1 Stream8 input and 1 Stream8 output.

11.6.1 Detailed Description

The MAPSStream8IOComponent component model.

Index