Type : **SPEC**	Projet RNTL **MORSE**	Réf : **SPEC-LfP-C++**
Date : **08/07/2005**		Sous-projet n° : **4**
Version : **1.0**		Tâche n° : **4.1**
Auteurs : **Antony Hubervic**		
Relecteurs : **Marc Richard-Foy**		

Génération de code C++

Liste des révisions

	Paragraphe	Commentaire
Version : 1.0 *Date* : 2005/07/08 *Auteur* : Antony Hubervic		Version initiale

Table des matières

1 Introduction

Ce document spécifie le code C++ à générer à partir d'un modèle LfP exprimé sous forme textuelle. Les chapitres suivants reprennent les constructions du langage LfP et établissent les règles de génération. Pour chaque cas un exemple est fourni présentant le code LfP et le code C++ associé. Dans le code C++, les éléments issus directement de LfP sont indiqués en **gras**. En annexe se trouve la documentation du runtime C++ pour LfP qui a permis de définir les règles de génération.

2 Diagramme d'architecture

2.1 Ports LfP

Les ports sont représentés par instanciation de la classe générique LfP::Port en utilisant les discriminants comme argument du type générique.

Pour un port sans discriminant, le type est défini en instanciant le type LfP::Port sans argument (la valeur par défaut de l'argument est utilisée).

```
type simple_port is port ;
```

```
/**
 * simple_port port type
 */
typedef LfP::Port<> simple_port;
```

Pour un port avec des discriminants, la génération comprend un type utilitaire contenant tous les discriminants du port. Ce type permet de sérialiser les discriminants pour transmission. Le type LfP::Port est instancié avec le type utilitaire pour créer le type de port.

```
type simple_port is port(integer) ;
```

```
/**
 * discriminant for simple_port port
 */
struct simple_port_discriminant_type
{
  // liste des discriminants
  LfP::integer integer_ ;
  /**
   * @brief Serialize the discriminant
   * @param[in,out] os stream where the object is written
   */
  void serialize(std::ostream& os)
  {
     os << "integer " << integer_;
  }
  /**
   * @brief Deserialize the discriminant
   * @param[in,out] is stream from which the object is read
   */
   void deserialize(std::istream& is)
   {
      std::string temp;
         is >> temp >> integer_;
   }
} ;
/**
 * simple_port port type
 */
typedef LfP::Port<simple_port_discriminant_type> simple_port;
```

2.2 Binders LfP

Le code généré pour les binders LfP est associé à la variable de type port sur lequel le binder est associé.
Pour chaque variable de type port d'une classe LfP, une classe C++ est créée contenant le binder en tant qu'attribut. Cette classe hérite du type C++ correspondant au port (cf §2.1).

```
binder client.itf IS FIFO(7)
       itf |-> rpc, client;
       itf <-| rpc, client;
end;

class client is
      itf : simple_port;
end;
```

```
class client_itf : public simple_port
{
private:
  LfP::GenericBinder<LfP::fifo<7>, simple_port_discriminant_type> binder_;
public:
  client_itf()
  {
    setBinder(binder_);
  }
};
```

Le type de l'attribut binder_ est une instanciation du type LfP::GenericBinder en remplaçant les paramètres génériques par :

- le type de file d'attente. Le type de file d'attente est spécifié en instanciant les classes génériques de la librairie LfP avec la taille de la file :
    - si la file est de type fifo, le type LfP::fifo doit être instancié avec la taille de la file
    - si la fille est de type bag, le type LfP::bag doit être instancié avec la taille de la file
- le type de discriminant utilisé par le port (cf §2.1)

Si le binder est de multiplicité all (binder static), l'attribut généré doit être statique (dans ce cas, l'attribut doit être défini dans le fichier .cpp)
L'association entre le port et le binder est fait dans le constructeur du port par appel à la méthode setBinder de la classe LfP::Port, avec en paramètre l'adresse de l'attribut binder.

2.3 Médias LfP

Les médias LfP sont représentés par des classes C++ dérivant de la classe LfP::Media.

```
media rpc ;
```

```
class rpc: public LfP::Media
{
public:
   rpc ();
   virtual ~rpc ();
   void run(void *arg=0) ;
};
```

2.4 Classes LfP

Les classes LfP sont représentées par des classes C++ dérivant de la classe LfP::Class.

```
class client ;
```

```
class client : public LfP::Class
{
public:
   client();
   virtual ~client();
   void run(void *arg=0) ;

};
```

2.5 Sérialisation

Lors de la transmission des messages, le discriminant et les données sont sérialisées par appel au méthodes suivantes :

- void serialize (std::ostream& os)
- void deserialize (std::istream& is)

Ces méthodes doivent être implémentées dans tous les types potentiellement transmissibles. Elles sont automatiquement générées pour les types non opaques.

3 Diagramme de comportement

Pour les classes et médias LfP, le diagramme de comportement conduit à la génération de code suivante :

- les variables du diagramme de comportement sont implémentées sous forme d'attributs de la classe C++ générée
- les fonctions et procédures sont implémentées sous forme de méthodes de la classe C++ générée
- le diagramme principal de comportement est implémenté dans la méthode run de la classe C++. Cette méthode run correspond à la fonction principale du thread associé à la classe.

3.1 Type enumérés

Un type enuméré LfP est traduit par une classe C++ contenant un type énuméré C++. La classe C++ contient de plus les méthodes de sérialisation du type énuméré et des méthodes utilitaires facilitant la conversion entre le type énuméré et la classe.
Le type énuméré interne à la classe reprend les valeurs du type énuméré LfP.

```
type t_msg_type is enum (diffuse, quit, join);
```

```
class t_msg_type {
public:
    enum t_msg_type_inner_enum {
        diffuse,
        quit,
        join
    };

private:
    t_msg_type::t_msg_type_inner_enum v_;

public:
    t_msg_type() {}
    // constructeur de conversion
    t_msg_type(const t_msg_type::t_msg_type_inner_enum& val) : v_(val){}

    // operateur de conversion
    operator t_msg_type_inner_enum() const { return v_; }
    // functions de serialisation
    void serialize(std::ostream& os) const { os << v_; }
    void deserialize(std::istream& is) { is >> v_; }
    bool operator ==(const t_msg_type& o) const { return v_==o.v_; }
    bool operator !=(const t_msg_type& o) const { return v_ != o.v_; }
    const t_msg_type& operator =(const t_msg_type& o) {
        if(&o!=this) { v_=o.v_; } return *this; }
};
```

3.2 Génération des attributs

Chaque variable d'un objet LfP est traduit par un attribut de la classe C++ correspondante. La traduction se fait selon la correspondance suivante :

Variable LfP	Attribut C++
Variable de type simple (integer, boolean)	Attribut du type correspondant

Variable de type port	Attribut de type utilitaire correspondant (cf § 2.2)
Variable de type media	Pointeur sur type correspondant (cf §2.3)
Variable de type classe	Pointeur sur type correspondant

Les attributs sont initialisés dans le constructeur de la classe générée, à la valeur fournie dans le code LfP.

Exemple :

```
class client is
  id : integer;
  i : integer := 0;
  itf : simple_port;
  my_rpc : rpc;
end;
```

```
class client : public LfP::Class
{
public:
   client();
   virtual ~client();

private:
   // variables
   LfP::integer id;
   LfP::integer i;
   client_itf  itf;
   rpc* my_rpc;

public:
   void run(void* arg);
};
```

3.3 Génération des procédures et fonctions

Pour chaque procédure ou fonction, la génération de code produit les éléments suivants :

- un type utilitaire pour la sérialisation des paramètres en entrée (in, inout) de la procédure
- un type utilitaire pour la sérialisation des paramètres en sortie (inout, out, retour) de la procédure
- une méthode dans la classe C++ associée à la classe LfP
- des méthodes utilitaire facilitant l'attente d'activation de méthode à travers un port
- une méthode utilitaire pour faciliter l'appel à la méthode à travers un port.

3.3.1 Types pour la sérialisation des paramètres

Les types sont nommés suivant la nomenclature suivante :

- <nom_de_la_procédure>_call_param_type pour les paramètres en entrée
- <nom_de_la_procédure>_return_value_type pour les paramètres en sortie

Pour chaque paramètre en entrée (ou entrée-sortie) de la procédure, un attribut est crée dans le type call_param_type. Pour chaque paramètre en sortie (ou entrée-sortie) de la procédure, un attribut est crée dans le type return_value_type.

Ces types doivent implémenter les méthodes de sérialisation et de-sérialisation.

```
class server is
  procedure handle_request (num : inout integer) -> ITF
  is
  begin
    num := integer'succ(1);
  end;
end;
```

```
class server : public LfP::Class
{
private:
  struct handle_request_call_param_type
  {
    LfP::integer num_;
    handle_request_call_param_type(const LfP::integer& num=0) :
      num_(num){}
    void serialize(std::ostream& os) const;
    void deserialize(std::istream& is):
  };
  struct handle_request_return_value_type
  {
    LfP::integer num_;
    void serialize(std::ostream& os) const;
    void deserialize(std::istream& is);
  };
};
```

3.3.2 Méthode de la classe C++

Une méthode est générée pour chaque procédure ou fonction de la classe LfP. Cette méthode reprend le nom de la procédure LfP et prend comme arguments les paramètres de la procédure LfP avec les correspondances suivantes :

- argument en entrée (in) : génération d'un paramètre passé par référence constante
- argument en sortie (out) ou entrée-sortie (inout) : génération d'un paramètre passé par référence
- type de retour : utilisé comme type de retour de la méthode

```
class server is
  procedure handle_request (num : inout integer) -> ITF
  is
  begin
    num := integer'succ(1);
  end;
end;
```

```
class server : public LfP::Class
{
public:
   // procedures
   void handle_request(LfP::integer& num);
};
```

3.3.3 Méthodes utilitaire pour l'attente d'activation

Deux méthodes sont générées pour faciliter l'attente d'activation de la méthode exportée. La première méthode permet l'attente sur le port par défaut de la procédure (ou fonction). La deuxième méthode est une méthode générique qui permet l'attente sur un port spécifié dans le code LfP. Le corps de ces méthodes fait appel aux méthodes utilitaires de la classe associée au port.

```
class server is
  procedure handle_request (num : inout integer) -> itf
  is
  begin
    num := integer'succ(1);
  end;
end;
```

```
class server : public LfP::Class
{
private:
   // procedures utiliaires
   void handle_request_wait();
   template <typename P> void handle_request_wait(P& port)
   {
        port.handle_request(*this);
    }
};
```

```
void server::handle_request_wait() {
    itf.handle_request(*this);
}
```

3.3.4 Méthode utilitaire pour les clients

Pour chaque classe LfP exportant des services (i.e. possédant des procédures ou fonctions) la génération de code doit produire une classe utilitaire permettant de faciliter les appels aux services.
Cette classe utilitaire est nommée en préfixant "_proxy" au nom de la classe LfP.
Pour chaque procédure ou fonction de la classe, la classe utilitaire propose une méthode statique générique du même nom que la procédure. Cette méthode prend comme paramètre un port, un discriminant ainsi que les arguments de la procédure LfP.

```
class server is
  procedure handle_request (num : inout integer) -> ITF
  is
  begin
    num := integer'succ(1);
  end;
end;
```

```
class server : public LfP::Class
{
  friend struct server_proxy;
};

struct server_proxy
{
public:
  template <typename D>
  static void handle_request(LfP::Port<D> port, D& discr,  LfP::integer& num)
  {
    server::handle_request_call_param_type param(num);
    server::handle_request_return_value_type retval;
    port.synchronousCall("handle_request",discr,param,retval);
    num=retval.num_;
  }
};
```

Le corps de la méthode statique fait appel aux fonctions définies dans la classe Port du runtime LfP :

- Un appel de procédure/fonction LfP avec des paramètres inout, out ou un type de retour sera implémenté par l'appel à la méthode synchronousCall
- Un appel de procédure/fonction LfP avec uniquement des paramètres in et sans type de retour sera implémenté par l'appel à la méthode asynchronousCall

3.4 Diagramme de transitions

Les diagrammes de transitions LfP sont associés soit à un objet LfP (diagramme principal), soit à une procédure ou fonction d'une classe. Le code généré pour le diagramme se situe :

- dans la méthode run de la classe C++ pour le cas d'un diagramme principal
- dans la méthode associée à la procédure pour les autres cas

3.4.1 Types

Les types énumérés LfP sont traduits selon le chapitre 3.1.
Les types simples LfP sont traduits selon la correspondance suivante :

Type LfP	**Type C++**
integer	LfP::integer
boolean	LfP::boolean
message	LfP::message

3.4.2 Expressions

Les expressions LfP sont traduites naturellement en expression C++.

Dans le cas particulier de l'expression de « ' », le code généré doit faire appel à une méthode statique sur le type concerné.
Exemple :

```
num := integer'succ(num) ;
```

se traduit par

```
num = integer::succ(num) ;
```

3.4.3 Instructions

Les instructions LfP sont traduites en instructions C++ selon les règles suivantes :

Instruction LfP	Règle de génération	Ex LfP	Ex C++
Label	Label C++ reprenant le nom du label LfP	`:start_loop:`	`start_loop :`
Instruction goto	Instruction goto reprenant le label LfP	`goto start_loop;`	`Goto start_loop;`

Boucle while	Boucle while. La condition de la boucle C++ reprend la condition LfP. Le corps de la boucle reprend les instructions LfP situées entre les mots clés begin et end.	`while i <= 5`
		`begin`
		
		`end;`
			`while(i<=5)`
			`{`
			
			`}`

Boucle for	Boucle for. La variable d'itération reprend le nom de la variable LfP et est de type int. La variable d'itération est initialisée avec la borne inférieure de la plage d'itération. La condition d'arrêt reprend la borne supérieure de la plage d'itération.	`for i in 1..nbr_of_groups`
		`begin`
		
		`end;`
			`for(int i=1;i<nbr_of_groups;++i)`
			`{`
			
			`}`

Instruction If	Instruction if. La condition C++ reprend la condition LfP.
	Un bloc elsif LfP est généré comme une instruction if C++ imbriqué dans le bloc else C++ du if précédent.
	Un bloc else LfP est généré comme un bloc else C++ du if précédent.
		`If msg_kind = join`
		`then`
		`elsif msg_kind = diffuse`
		`then`
		`elsif msg_kind = quit`
		`then`
		`else`
		`end;`
			`if(msg_kind == join)`
			`{`
			`}`
			`else`
			`{`
			`  if(msg_kind == diffuse)`
			`  {`
			`  }`
			`  else`
			`  {`
			`    if(msg_kind == quit)`
			`    {`
			`    }`
			`    else`
			`    {`
			`    }`
			`  }`
			`}`

Affectation	Affectation C++	`num := integer'succ(num);`	`num = LfP::integer::suc(num);`
Appel de procédure/fonction synchrone	Appel de méthode sur la variable de type port. Le discriminant est passé comme premier paramètre à la méthode, suivi des arguments de la procédure LfP. Cet appel utilise la classe proxy du serveur. L'identification du serveur sur lequel faire l'appel de procédure doit être non ambigu dans le code LfP. L'identification peut être faite sur le nom de la méthode si il est unique dans tout le système, ou sur le nom du serveur précisé lors de l'appel de méthode à travers la syntaxe LfP.	`[id] handle_request(i) <-> itf ;`	`server_proxy::handle_request(&itf,id,i);`
Appel de procédure/fonction asynchrone	Appel de méthode sur la variable de type port. Le discriminant est passé comme premier paramètre à la méthode, suivi des arguments de la procédure LfP. Cet appel la classe proxy du serveur. L'identification du serveur sur lequel faire l'appel de procédure doit être non ambiguë dans le code LfP. L'identification peut être faite sur le nom de la méthode si il est unique dans tout le système, ou sur le nom du serveur précisé lors de l'appel de méthode à		

	travers la syntaxe LfP.		
Attente de procédure/fonction	Appel de méthode sur la variable de type port. L'objet devant traiter l'appel est passé en paramètre à cette méthode (en tant que référence C++). Cet appel utilise les méthodes utilitaires associées au type port.	`handle_request <- itf ;`	`itf.handle_request(*this);`
Envoi de message (dans un média)	Appel à la méthode mediaWriteMessage sur la variable de type port.	`(msg) -> target;`	`target.mediaWriteMessage(msg);`

Attente de réception d'un message (dans un média)	Appel à la méthode mediaReadMessage sur la variable de type port.
	Une garde sur la réception est traduite par la génération d'une variable passée à la méthode mediaReadMessage.
		`[id2] (msg) <- target with id1 = id2 ;`	`simple_port::IsEqualGuard guard(id1);`
			`target.mediaReadMessage(msg,&guard);`
			`input.mediaWriteMessage(msg);`

Runtime LfP C++

Table of Contents

4 LfPC++runtime Namespace Index

4.1 LfPC++runtime Namespace List

Here is a list of all namespaces with brief descriptions:

5 LfPC++runtime Hierarchical Index

5.1 LfPC++runtime Class Hierarchy

6 LfPC++runtime Class Index

6.1 LfPC++runtime Class List

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

7 LfPC++runtime File Index

7.1 LfPC++runtime File List

Here is a list of all files with brief descriptions:

8 LfPC++runtime Page Index

8.1 LfPC++runtime Related Pages

9 LfPC++runtime Namespace Documentation

9.1 aonix_posix Namespace Reference

Library for posix utilities wrapped in C++ classes.

9.1.1 Classes

1. class **cond**
   Wrapper class for posix condition variables.

2. class **cond2**
   Wrapper class for posix condition variables This class has its own mutex to synchronize accesses to the condition variable.

3. class **mutex**
   Wrapper for posix mutexes.

4. class **mutex_lock**
   Create a lock on a posix mutex.

5. class **mutex_trylock**
   Try to acquire a posix mutex.

6. class **thread**
   Wrapper class for posix threads.

9.1.2 Detailed Description

Library for posix utilities wrapped in C++ classes.

9.1.2.1.1 Author:

Antony Hubervic

9.2 aonix_util Namespace Reference

Utilitiy classes for exceptions and return value handling.

9.2.1 Classes

7. struct **bad_result_exception**
   exception class

8. struct **positive_domain**
   Check if a value is positive or null.

9. struct **null_domain**
   Check if a value is null.

10. struct **return_value_handler**
    Helper class to deal with return values.

11. class **native_holder**
    Helper class to hold native types.

12. class **non_copyable**
    Helper class to create non copyable classes.

9.2.2 Detailed Description

Utilitiy classes for exceptions and return value handling.

9.2.2.1.1 Author:

Antony Hubervic

9.3 LfP Namespace Reference

Library for executing LfP models.

9.3.1 Classes

13. struct **null_discriminant**
    Default discriminant type.

14. class **Binder**
    *Base class for **LfP** binders.*

15. class **GenericBinder**
    Generic binder implementation.

16. class **Class**
    *Base class for **LfP** classes.*

17. class **Media**
    *Base class for **LfP** medias.*

18. class **Message**
    ***Class** for **LfP** messages.*

19. class **Port**
    *Base class for **LfP** ports.*

20. struct **fifo**
    ***Class** for fifo queuing policy inside binders.*

21. struct **bag**
    ***Class** for bag queuing policy inside binders.*

22. class **integer**
    LfP *integer datatype.*

23. class **boolean**
    LfP *boolean datatype.*

9.3.2 Functions

24. **integer succ** (const **integer** &o)
    Utility free function to compute successor of an integer.

25. **integer pred** (const **integer** &o)
    Utility free function to compute predecessor of an integer.

26. std::ostream & **operator**<< (std::ostream &os, const **integer** &i)
27. std::istream & **operator**>> (std::istream &is, **integer** &i)
28. std::ostream & **operator**<< (std::ostream &os, const **boolean** &i)
29. std::istream & **operator**>> (std::istream &is, **boolean** &i)
30. **boolean succ** (const **boolean** &o)
    Utility free function to compute successor of a boolean.

31. **boolean pred** (const **boolean** &o)
    Utility free function to compute successor of a boolean.

9.3.3 Detailed Description

Library for executing LfP models.

9.3.4 Function Documentation

9.3.4.1 std::ostream& LfP::operator<< (std::ostream & *os*, const boolean & *i*) [inline]

9.3.4.2 std::ostream& LfP::operator<< (std::ostream & *os*, const integer & *i*) [inline]

9.3.4.3 std::istream& LfP::operator>> (std::istream & *is*, boolean & *i*) [inline]

9.3.4.4 std::istream& LfP::operator>> (std::istream & *is*, integer & *i*) [inline]

9.3.4.5 boolean LfP::pred (const boolean & *o*)

Utility free function to compute successor of a boolean.

Calls **boolean::pred**

9.3.4.6 integer LfP::pred (const integer & *o*)

Utility free function to compute predecessor of an integer.

Calls **integer::pred**

9.3.4.7 boolean LfP::succ (const boolean & *o*)

Utility free function to compute successor of a boolean.

Calls **boolean::succ**

9.3.4.8 integer LfP::succ (const integer & *o*)

Utility free function to compute successor of an integer.

Calls **integer::succ**

10 LfPC++runtime Class Documentation

10.1 aonix_util::bad_result_exception Struct Reference

exception class

`#include <posix_util.h>`

10.1.1 Public Member Functions

32. **bad_result_exception** (int value)

10.1.2 Public Attributes

33. int **value_**

10.1.3 Detailed Description

exception class

10.1.4 Constructor & Destructor Documentation

10.1.4.1 aonix_util::bad_result_exception::bad_result_exception (int *value*) `[inline]`

10.1.5 Member Data Documentation

10.1.5.1 int aonix_util::bad_result_exception::value_

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

34. **posix_util.h**

10.1.5.2

10.2 LfP::bag< size > Struct Template Reference

Class for bag queuing policy inside binders.

`#include <queues.h>`

10.2.1 Public Member Functions

35. **Message** & **front** ()
    Return the oldest message without removing it.

36. void **pop_front** ()
    Remove the oldest message.

37. void **push_back** (const **Message** &msg)
    Add a message to the queue.

38. bool **full** ()
    Test if the queue is full.

39. bool **empty** ()
    Test if the queue is empty.

10.2.2 Private Attributes

40. std::set< **Message** > **queue_**

10.2.3 Detailed Description

10.2.3.1 template<int size> struct LfP::bag< size >

Class for bag queuing policy inside binders.

10.2.4 Member Function Documentation

10.2.4.1 template<int size> bool LfP::bag< size >::empty () `[inline]`

Test if the queue is empty.

10.2.4.1.1 Returns:

true if the queue is empty

10.2.4.2 template<int size> Message& LfP::bag< size >::front () [inline]

Return the oldest message without removing it.

10.2.4.2.1 Returns:

oldest message

10.2.4.2.2 Precondition:

queue is not empty

10.2.4.2.3 Postcondition:

queue size is not modified

10.2.4.3 template<int size> bool LfP::bag< size >::full () [inline]

Test if the queue is full.

10.2.4.3.1 Returns:

true if the queue is full

10.2.4.4 template<int size> void LfP::bag< size >::pop_front () [inline]

Remove the oldest message.

10.2.4.4.1 Precondition:

queue is not empty

10.2.4.4.2 Postcondition:

queue size is not modified

10.2.4.5 template<int size> void LfP::bag< size >::push_back (const Message & *msg*) [inline]

Add a message to the queue.

10.2.4.5.1 Parameters:

msg message to be added. message is copied

10.2.4.5.2 Precondition:

queue is not full

10.2.4.5.3 Postcondition:

queue size is increased by one

10.2.5 Member Data Documentation

10.2.5.1 template<int size> std::set<Message> LfP::bag< size >::queue_ [private]

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

41. **queues.h**

10.2.5.2

10.3 LfP::Binder< D > Class Template Reference

Base class for **LfP** binders.

`#include <Binder.h>`

Inheritance diagram for LfP::Binder< D >:

10.3.1 Public Types

42. typedef D **discriminant_type**

10.3.2 Public Member Functions

43. virtual **~Binder** ()
    Destructor.

44. virtual void **classReadMessage** (**Message** &msg, const **Guard** *const guard=0)=0
    Read a message from a class point of view.

45. virtual void **classWriteMessage** (**Message** &msg)=0
    Write a message from a class point of view.

46. virtual void **mediaReadMessage** (**Message** &msg, const **Guard** *const guard=0)=0
    Read a message from a media point of view.

47. virtual void **mediaWriteMessage** (**Message** &msg)=0
    Write a message from a media point of view.

10.3.3 Classes

48. struct **Guard**
    Interface type for the guards to check the discriminant of incoming messages.

10.3.4 Detailed Description

10.3.4.1 template<typename D = null_discriminant> class LfP::Binder< D >

Base class for **LfP** binders.

Type parameter D : discriminant type

10.3.5 Member Typedef Documentation

10.3.5.1 template<typename D = null_discriminant> typedef D LfP::Binder< D >::discriminant_type

Reimplemented in **LfP::GenericBinder< Q, D >** (*p.32*).

10.3.6 Constructor & Destructor Documentation

10.3.6.1 template<typename D = null_discriminant> virtual LfP::Binder< D >::~Binder () `[inline, virtual]`

Destructor.

Does nothing

10.3.7 Member Function Documentation

10.3.7.1 template<typename D = null_discriminant> virtual void LfP::Binder< D >::classReadMessage (Message & *msg*, const Guard *const *guard* = `0`) `[pure virtual]`

Read a message from a class point of view.

10.3.7.1.1 Parameters:

msg receive the message which was read
guard guard to check if the message has the correct discriminant

blocking if no incoming queue is empty

10.3.7.2 template<typename D = null_discriminant> virtual void LfP::Binder< D >::classWriteMessage (Message & *msg*) `[pure virtual]`

Write a message from a class point of view.

10.3.7.2.1 Parameters:

msg (out) receives the message

blocking if outgoing queue is full

10.3.7.3 Implemented in **LfP::GenericBinder< Q, D >** (*p.32*).template<typename D = null_discriminant> virtual void LfP::Binder< D >::mediaReadMessage (Message & *msg*, const Guard *const *guard* = `0`) `[pure virtual]`

Read a message from a media point of view.

10.3.7.3.1 Parameters:

msg receive the message which was read

guard guard to check if the message has the correct discriminant

blocking if incoming queue is empty

10.3.7.4 template<typename D = null_discriminant> virtual void LfP::Binder< D >::mediaWriteMessage (Message & *msg*) `[pure virtual]`

Write a message from a media point of view.

10.3.7.4.1 Parameters:

msg message to be sent

blocking if outgoing queue is full

Implemented in **LfP::GenericBinder< Q, D >** (*p.33*).

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

49. **Binder.h**

10.3.7.5

10.4 LfP::Binder< D >::Guard Struct Reference

Interface type for the guards to check the discriminant of incoming messages.

`#include <Binder.h>`

Inheritance diagram for LfP::Binder< D >::Guard:

10.4.1 Public Member Functions

50. virtual bool **check** (const **discriminant_type** &discr) const =0
    Check that the discriminant of an incoming message matches some conditions.

10.4.2 Detailed Description

10.4.2.1 template<typename D = null_discriminant> struct LfP::Binder< D >::Guard

Interface type for the guards to check the discriminant of incoming messages.

10.4.3 Member Function Documentation

10.4.3.1 template<typename D = null_discriminant> virtual bool LfP::Binder< D >::Guard::check (const discriminant_type & *discr*) const `[pure virtual]`

Check that the discriminant of an incoming message matches some conditions.

10.4.3.1.1 Parameters:

discr discriminant of the incoming message

10.4.3.1.2 Returns:

true if the discriminant matches the condition, false otherwise

Implemented in **LfP::Port< D >::IsEqualGuard** (*p.69*).

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

51. **Binder.h**

10.4.3.2

10.5 LfP::boolean Class Reference

LfP boolean datatype.

`#include <types.h>`

10.5.1 Public Member Functions

52. **boolean** (bool v=false)
    Default constructor.

53. **operator bool** () const
    Conversion operator to bool.

54. bool **operator==** (const **boolean** &i) const
    Equality comparison operator.

55. bool **operator!=** (const **boolean** &i) const
    Inequality comparison operator.

56. void **serialize** (std::ostream &os) const
    Write the value in a stream.

57. void **deserialize** (std::istream &is)
    Read the value from a stream.

10.5.2 Static Public Member Functions

58. static bool **pred** (bool v)
    Computes the predecessor of a given integer value.

59. static bool **succ** (bool v)
    Computes the successor of a given integer value.

10.5.3 Private Attributes

60. bool **v_**

10.5.4 Friends

61. std::ostream & **operator**<< (std::ostream &os, const **boolean** &i)
62. std::istream & **operator**>> (std::istream &is, **boolean** &i)

10.5.5 Detailed Description

LfP boolean datatype.

10.5.6 Constructor & Destructor Documentation

10.5.6.1 LfP::boolean::boolean (bool *v* = `false`) `[inline]`

Default constructor.

10.5.6.1.1 *Parameters:*

v initial value

Provides implicit conversion from C++ bool

10.5.7 Member Function Documentation

10.5.7.1 void LfP::boolean::deserialize (std::istream & *is*) `[inline]`

Read the value from a stream.

10.5.7.1.1 *Parameters:*

is stream from which the object is read

10.5.7.2 LfP::boolean::operator bool () const `[inline]`

Conversion operator to bool.

10.5.7.2.1 *Returns:*

the object's value as a C++ bool

10.5.7.3 bool LfP::boolean::operator!= (const boolean & *i*) const [inline]

Inequality comparison operator.

10.5.7.3.1 *Parameters:*

i object to be compared to

10.5.7.3.2 *Returns:*

true if the values are different

10.5.7.4 bool LfP::boolean::operator== (const boolean & *i*) const [inline]

Equality comparison operator.

10.5.7.4.1 *Parameters:*

i object to be compared to

10.5.7.4.2 *Returns:*

true if the values are equal

10.5.7.5 static bool LfP::boolean::pred (bool *v*) [inline, static]

Computes the predecessor of a given integer value.

10.5.7.5.1 *Parameters:*

v value to be decreased

10.5.7.5.2 *Returns:*

integer predecessor of v

10.5.7.6 void LfP::boolean::serialize (std::ostream & *os*) const [inline]

Write the value in a stream.

10.5.7.6.1 *Parameters:*

os stream where the object is written

10.5.7.7 static bool LfP::boolean::succ (bool *v*) [inline, static]

Computes the successor of a given integer value.

10.5.7.7.1 *Parameters:*

v value to be invreased

10.5.7.7.2 *Returns:*

integer succecessor of v

10.5.8 Friends And Related Function Documentation

10.5.8.1 std::ostream& operator<< (std::ostream & *os*, const boolean & *i*) `[friend]`

10.5.8.2 std::istream& operator>> (std::istream & *is*, boolean & *i*) `[friend]`

10.5.9 Member Data Documentation

10.5.9.1 bool LfP::boolean::v_ `[private]`

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

63. **types.h**

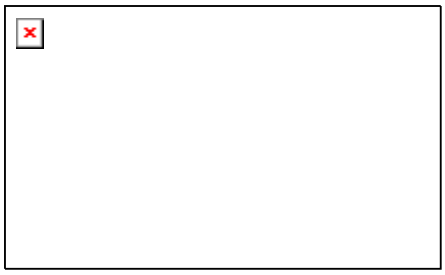
10.5.9.2

10.6 LfP::Class Class Reference

Base class for **LfP** classes.

`#include <Class.h>`

Inheritance diagram for LfP::Class:



10.6.1 Public Member Functions

64. **Class** ()
65. virtual **~Class** ()

10.6.2 Detailed Description

Base class for **LfP** classes.

Executed in its own thread. Children classes must implement the run method. The thread needs to be started using the start method

---

10.6.3 Constructor & Destructor Documentation

10.6.3.1 LfP::Class::Class ()

10.6.3.2 LfP::Class::~Class () `[virtual]`

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

66. **Class.h**
67. **Class.cpp**

10.6.3.3

10.7 aonix_posix::cond Class Reference

Wrapper class for posix condition variables.

`#include <posix_cond.h>`

Inheritance diagram for aonix_posix::cond:

10.7.1 Public Member Functions

68. **cond** (**cond::attribute** &attr)
69. **~cond** ()
70. void **signal** ()
    Signal the condition variable.

71. void **broadcast** ()
    Broadcast the condition variable.

72. void **wait** (**mutex_lock** &lock)
    Wait on the condition variable.

10.7.2 Classes

73. struct **attribute**
    Wrapper for posix condition variable attributes.

10.7.3 Detailed Description

Wrapper class for posix condition variables.

10.7.4 Constructor & Destructor Documentation

10.7.4.1 aonix_posix::cond::cond (cond::attribute & *attr*)

10.7.4.2 aonix_posix::cond::~cond ()

10.7.5 Member Function Documentation

10.7.5.1 void aonix_posix::cond::broadcast ()

Broadcast the condition variable.

10.7.5.2 void aonix_posix::cond::signal ()

Signal the condition variable.

10.7.5.3 void aonix_posix::cond::wait (mutex_lock & *lock*)

Wait on the condition variable.

10.7.5.3.1 Parameters:

lock lock used to synchronize accesses to the condition variable The lock is released while waiting, and reentered when awake

---

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

74. **posix_cond.h**
75. **posix_cond.cpp**

10.7.5.4

10.8 aonix_posix::cond2 Class Reference

Wrapper class for posix condition variables This class has its own mutex to synchronize accesses to the condition variable.

`#include <posix_cond.h>`

Inheritance diagram for aonix_posix::cond2:

10.8.1 Public Member Functions

76. **cond2** (**cond::attribute** &attr)
77. **~cond2** ()
78. void **signal** ()
    Signal the condition variable.

79. void **broadcast** ()
    Broadcast the condition variable.

80. void **wait** ()
    Wait on the condition variable.

81. **mutex** & **get_mutex** ()

10.8.2 Private Attributes

82. **mutex::attribute mutex_attr_**
83. **mutex mutex_**

10.8.3 Detailed Description

Wrapper class for posix condition variables This class has its own mutex to synchronize accesses to the condition variable.

10.8.3.1.1 Author:

Antony Hubervic

10.8.4 Constructor & Destructor Documentation

10.8.4.1 aonix_posix::cond2::cond2 (cond::attribute & *attr*)

10.8.4.2 aonix_posix::cond2::~cond2 ()

10.8.5 Member Function Documentation

10.8.5.1 void aonix_posix::cond2::broadcast ()

Broadcast the condition variable.

10.8.5.2 mutex& aonix_posix::cond2::get_mutex () `[inline]`

10.8.5.3 void aonix_posix::cond2::signal ()

Signal the condition variable.

10.8.5.4 void aonix_posix::cond2::wait ()

Wait on the condition variable.

10.8.6 Member Data Documentation

10.8.6.1 mutex aonix_posix::cond2::mutex_ `[private]`

10.8.6.2 mutex::attribute aonix_posix::cond2::mutex_attr_ `[private]`

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

84. **posix_cond.h**
85. **posix_cond.cpp**

10.8.6.3

10.9 aonix_posix::cond::attribute Struct Reference

Wrapper for posix condition variable attributes.

`#include <posix_cond.h>`

Inheritance diagram for aonix_posix::cond::attribute:

10.9.1 Public Member Functions

86. **attribute** ()
87. **~attribute** ()

10.9.2 Detailed Description

Wrapper for posix condition variable attributes.

10.9.3 Constructor & Destructor Documentation

10.9.3.1 aonix_posix::cond::attribute::attribute () `[inline]`

10.9.3.2 aonix_posix::cond::attribute::~attribute () `[inline]`

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

88. **posix_cond.h**

10.9.3.3

10.10LfP::fifo< size > Struct Template Reference

Class for fifo queuing policy inside binders.

`#include <queues.h>`

10.10.1Public Member Functions

89. **Message** & **front** ()
    Return the oldest message without removing it.

90. void **pop_front** ()
    Remove the oldest message.

91. void **push_back** (const **Message** &msg)
    Add a message to the queue.

92. bool **full** ()
    Test if the queue is full.

93. bool **empty** ()
    Test if the queue is empty.

10.10.2Private Attributes

94. std::list< **Message** > **queue_**

10.10.3Detailed Description

10.10.3.1 template<int size> struct LfP::fifo< size >

Class for fifo queuing policy inside binders.

10.10.4Member Function Documentation

10.10.4.1 template<int size> bool LfP::fifo< size >::empty () `[inline]`

Test if the queue is empty.

10.10.4.1.1 Returns:

true if the queue is empty

10.10.4.2 template<int size> Message& LfP::fifo< size >::front () [inline]

Return the oldest message without removing it.

10.10.4.2.1 Returns:

oldest message

10.10.4.2.2 Precondition:

queue is not empty

10.10.4.2.3 Postcondition:

queue size is not modified

10.10.4.3 template<int size> bool LfP::fifo< size >::full () [inline]

Test if the queue is full.

10.10.4.3.1 Returns:

true if the queue is full

10.10.4.4 template<int size> void LfP::fifo< size >::pop_front () [inline]

Remove the oldest message.

10.10.4.4.1 Precondition:

queue is not empty

10.10.4.4.2 Postcondition:

queue size is decreased by one

10.10.4.5 template<int size> void LfP::fifo< size >::push_back (const Message & *msg*) [inline]

Add a message to the queue.

10.10.4.5.1 Parameters:

msg message to be added. message is copied

10.10.4.5.2 Precondition:

queue is not full

10.10.4.5.3 Postcondition:

queue is increased by one

10.10.5Member Data Documentation

10.10.5.1 template<int size> std::list<Message> LfP::fifo< size >::queue_ [private]

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

95. **queues.h**

10.10.5.2

10.11LfP::GenericBinder< Q, D > Class Template Reference

Generic binder implementation.

`#include <Binder.h>`

Inheritance diagram for LfP::GenericBinder< Q, D >:

10.11.1Public Types

96. typedef D **discriminant_type**
97. typedef Q **container_type**
98. typedef **Binder**< D >::Guard **guard_type**

10.11.2Public Member Functions

99. **GenericBinder** ()
100. virtual **~GenericBinder** ()
101. void **classReadMessage** (**Message** &msg, const **guard_type** *const guard=0)
102. void **classWriteMessage** (**Message** &msg)

Write a message from a class point of view.

103. void **mediaReadMessage** (**Message** &msg, const **guard_type** *const guard=0)
104. void **mediaWriteMessage** (**Message** &msg)

Write a message from a media point of view.

10.11.3Protected Attributes

105. **container_type to_class_internal_queue**
106. **container_type to_media_internal_queue**

buffer on class side (class -> media)

107. **aonix_posix::mutex::attribute mutex_attr_**

buffer on media side (media -> class)

108. **aonix_posix::cond::attribute cond_attr_**

attribute for mutex creation

109. **aonix_posix::mutex class_mutex_**
attribute for condition variable creation

110. **aonix_posix::cond class_empty_cond_**
mutex to protect the class buffer

111. **aonix_posix::cond class_full_cond_**
condition to wait when class buffer is empty

112. **aonix_posix::cond class_wgard_cond_**
condition to wait when class buffer is full

113. **aonix_posix::mutex media_mutex_**
condition to wait when guard is not valid

114. **aonix_posix::cond media_empty_cond_**
mutex to protect the media buffer

115. **aonix_posix::cond media_full_cond_**
condition to wait when media buffer is empty

116. **aonix_posix::cond media_wgard_cond_**
condition to wait when media buffer is full

10.11.4Detailed Description

10.11.4.1 template<typename Q, typename D = null_discriminant> class LfP::GenericBinder< Q, D >

Generic binder implementation.

Type parameters D : discriminant type (must be serializable through iostream) Q : queuing policy (see fifo and bag)

10.11.5Member Typedef Documentation

10.11.5.1 template<typename Q, typename D = null_discriminant> typedef Q LfP::GenericBinder< Q, D >::container_type

Queuing policy used in this binder

10.11.5.2 template<typename Q, typename D = null_discriminant> typedef D LfP::GenericBinder< Q, D >::discriminant_type

Type for discriminant used in this binder

10.11.5.3 Reimplemented from **LfP::Binder< D >** (*p.13*).template<typename Q, typename D = null_discriminant> typedef Binder<D>::Guard LfP::GenericBinder< Q, D >::guard_type

Interface for guards used on discriminant

10.11.6Constructor & Destructor Documentation

10.11.6.1 template<typename Q, typename D = null_discriminant> LfP::GenericBinder< Q, D >::GenericBinder () `[inline]`

10.11.6.2 template<typename Q, typename D = null_discriminant> virtual LfP::GenericBinder< Q, D >::~GenericBinder () `[inline, virtual]`

10.11.7Member Function Documentation

10.11.7.1 template<typename Q, typename D = null_discriminant> void LfP::GenericBinder< Q, D >::classReadMessage (Message & *msg*, const guard_type *const *guard* = `0`) `[inline]`

10.11.7.2 template<typename Q, typename D = null_discriminant> void LfP::GenericBinder< Q, D >::classWriteMessage (Message & *msg*) `[inline, virtual]`

Write a message from a class point of view.

10.11.7.2.1 Parameters:

msg (out) receives the message

blocking if outgoing queue is full

10.11.7.3 Implements **LfP::Binder< D >** (*p.13*).template<typename Q, typename D = null_discriminant> void LfP::GenericBinder< Q, D >::mediaReadMessage (Message & *msg*, const guard_type *const *guard* = `0`) `[inline]`

10.11.7.4 template<typename Q, typename D = null_discriminant> void LfP::GenericBinder< Q, D >::mediaWriteMessage (Message & *msg*) `[inline, virtual]`

Write a message from a media point of view.

10.11.7.4.1 Parameters:

msg message to be sent

blocking if outgoing queue is full

Implements **LfP::Binder< D >** (*p.14*).

10.11.8Member Data Documentation

10.11.8.1 template<typename Q, typename D = null_discriminant> aonix_posix::cond LfP::GenericBinder< Q, D >::class_empty_cond_ `[protected]`

mutex to protect the class buffer

10.11.8.2 template<typename Q, typename D = null_discriminant> aonix_posix::cond LfP::GenericBinder< Q, D >::class_full_cond_ `[protected]`

condition to wait when class buffer is empty

10.11.8.3 template<typename Q, typename D = null_discriminant> aonix_posix::mutex LfP::GenericBinder< Q, D >::class_mutex_ `[protected]`

attribute for condition variable creation

10.11.8.4 template<typename Q, typename D = null_discriminant> aonix_posix::cond LfP::GenericBinder< Q, D >::class_wgard_cond_ `[protected]`

condition to wait when class buffer is full

10.11.8.5 template<typename Q, typename D = null_discriminant> aonix_posix::cond::attribute LfP::GenericBinder< Q, D >::cond_attr_ `[protected]`

attribute for mutex creation

10.11.8.6 template<typename Q, typename D = null_discriminant> aonix_posix::cond LfP::GenericBinder< Q, D >::media_empty_cond_ [protected]

mutex to protect the media buffer

10.11.8.7 template<typename Q, typename D = null_discriminant> aonix_posix::cond LfP::GenericBinder< Q, D >::media_full_cond_ [protected]

condition to wait when media buffer is empty

10.11.8.8 template<typename Q, typename D = null_discriminant> aonix_posix::mutex LfP::GenericBinder< Q, D >::media_mutex_ [protected]

condition to wait when guard is not valid

10.11.8.9 template<typename Q, typename D = null_discriminant> aonix_posix::cond LfP::GenericBinder< Q, D >::media_wgard_cond_ [protected]

condition to wait when media buffer is full

10.11.8.10 template<typename Q, typename D = null_discriminant> aonix_posix::mutex::attribute LfP::GenericBinder< Q, D >::mutex_attr_ [protected]

buffer on media side (media -> class)

10.11.8.11 template<typename Q, typename D = null_discriminant> container_type LfP::GenericBinder< Q, D >::to_class_internal_queue [protected]

10.11.8.12 template<typename Q, typename D = null_discriminant> container_type LfP::GenericBinder< Q, D >::to_media_internal_queue [protected]

buffer on class side (class -> media)

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

117. **Binder.h**

10.11.8.13

10.12LfP::integer Class Reference

LfP integer datatype.

`#include <types.h>`

10.12.1Public Member Functions

118. **integer** (int v=0)
Default constructor.

119. **integer** (const **integer** &cp)
Copy constructor.

120. **operator int** () const
Conversion operator to int.

121. const **integer** & **operator=** (const **integer** &cp)
Assignment operator.

122. bool **operator==** (const **integer** &i) const
Equality comparison operator.

123. bool **operator!=** (const **integer** &i) const
Inequality comparison operator.

124. void **serialize** (std::ostream &os) const
Write the value in a stream.

125. void **deserialize** (std::istream &is)
Read the value from a stream.

10.12.2Static Public Member Functions

126. static int **pred** (int v)
Computes the predecessor of a given integer value.

127.static int **succ** (int v)

Computes the successor of a given integer value.

10.12.3Private Attributes

128.int **v_**

10.12.4Friends

129.std::ostream & **operator**<< (std::ostream &os, const **integer** &i)
130.std::istream & **operator**>> (std::istream &is, **integer** &i)

10.12.5Detailed Description

LfP integer datatype.

10.12.6Constructor & Destructor Documentation

10.12.6.1 LfP::integer::integer (int *v* = `0`) `[inline]`

Default constructor.

10.12.6.1.1 Parameters:

v initial value

Provides implicit conversion from C++ int

10.12.6.2 LfP::integer::integer (const integer & *cp*) `[inline]`

Copy constructor.

10.12.7Member Function Documentation

10.12.7.1 void LfP::integer::deserialize (std::istream & *is*) `[inline]`

Read the value from a stream.

10.12.7.1.1 Parameters:

is stream from which the object is read

10.12.7.2 LfP::integer::operator int () const [inline]

Conversion operator to int.

10.12.7.2.1 Returns:

the object's value as a C++ int

10.12.7.3 bool LfP::integer::operator!= (const integer & *i*) const [inline]

Inequality comparison operator.

10.12.7.3.1 Parameters:

i object to be compared to

10.12.7.3.2 Returns:

true if the values are different

10.12.7.4 const integer& LfP::integer::operator= (const integer & *cp*) [inline]

Assignment operator.

10.12.7.4.1 Parameters:

cp object to be copied

10.12.7.4.2 Returns:

the current object

10.12.7.5 bool LfP::integer::operator== (const integer & *i*) const [inline]

Equality comparison operator.

10.12.7.5.1 Parameters:

i object to be compared to

10.12.7.5.2 Returns:

true if the values are equal

10.12.7.6 static int LfP::integer::pred (int *v*) [inline, static]

Computes the predecessor of a given integer value.

10.12.7.6.1 Parameters:

v value to be decreased

10.12.7.6.2 Returns:

integer predecessor of v

10.12.7.7 void LfP::integer::serialize (std::ostream & *os*) const [inline]

Write the value in a stream.

10.12.7.7.1 Parameters:

os stream where the object is written

10.12.7.8 static int LfP::integer::succ (int *v*) [inline, static]

Computes the successor of a given integer value.

10.12.7.8.1 Parameters:

v value to be invreased

10.12.7.8.2 Returns:

integer succecessor of v

10.12.8Friends And Related Function Documentation

10.12.8.1 std::ostream& operator<< (std::ostream & *os*, const integer & *i*) [friend]

10.12.8.2 std::istream& operator>> (std::istream & *is*, integer & *i*) [friend]

10.12.9Member Data Documentation

10.12.9.1 int LfP::integer::v_ [private]

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

131.**types.h**

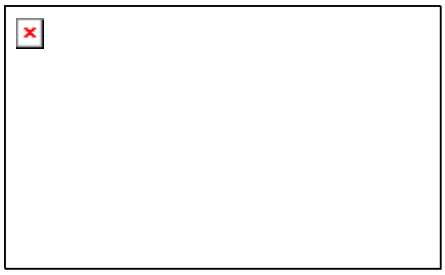
10.12.9.2

10.13LfP::Media Class Reference

Base class for **LfP** medias.

`#include <Media.h>`

Inheritance diagram for LfP::Media:



10.13.1Public Member Functions

132.**Media** ()

Default contructor.

133.virtual **~Media** ()

10.13.2Detailed Description

Base class for **LfP** medias.

Executed in its own thread. Children classes must implement the run method

---

10.13.3Constructor & Destructor Documentation

10.13.3.1 LfP::Media::Media ()

Default contructor.

Automatically start the thread

10.13.3.2 LfP::Media::~Media () `[virtual]`

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

134.**Media.h**
135.**Media.cpp**

10.13.3.3

10.14LfP::Message Class Reference

Class for **LfP** messages.

`#include <Message.h>`

10.14.1Public Member Functions

136. **Message** (const std::string &functionName="")
*Create a **Message** with a functionName.*

137. **Message** (const **Message** &cp)
Copy constructor.

138. const **Message** & **operator=** (const **Message** &cp)
Assignment operator.

139. bool **operator<** (const **Message** &o)
comparison operator

140. void **mediaSetData** (const std::string &data)
Set the data value.

141. void **mediaGetData** (std::string &data) const
Return the data value.

142. void **mediaSetDiscriminant** (const std::string &discr)
Set the discriminant value.

143. void **mediaGetDiscriminant** (std::string &discr) const
Return the discriminant value.

144. template<typename V> void **setData** (const V &data)
Set the data value.

145.template<typename V> void **getData** (V &data) const
Return the data value.

146.template<typename V> void **setDiscriminant** (const V &discr)
Set the discriminant value.

147.template<typename V> void **getDiscriminant** (V &discr) const
Return the data value.

10.14.2Protected Attributes

148.std::string **functionName_**
149.std::string **discr_**
150.std::string **data_**

10.14.3Detailed Description

Class for **LfP** messages.

10.14.4Constructor & Destructor Documentation

10.14.4.1 LfP::Message::Message (const std::string & *functionName* = "")

Create a **Message** with a functionName.

10.14.4.1.1 Parameters:

functionName name of the called function

10.14.4.2 LfP::Message::Message (const Message & *cp*)

Copy constructor.

10.14.5Member Function Documentation

10.14.5.1 template<typename V> void LfP::Message::getData (V & *data*) const [inline]

Return the data value.

10.14.5.1.1 Parameters:

data receive the data value

The type V must support operation void deserialize(std::istream&)

10.14.5.2 template<typename V> void LfP::Message::getDiscriminant (V & *discr*) const [inline]

Return the data value.

10.14.5.2.1 Parameters:

discr receive the discriminant value

The type V must support operation void deserialize(std::istream&)

10.14.5.3 void LfP::Message::mediaGetData (std::string & *data*) const [inline]

Return the data value.

10.14.5.3.1 Parameters:

data receive the data value

10.14.5.3.2 Warning:

Must be called only by Medias

10.14.5.4 void LfP::Message::mediaGetDiscriminant (std::string & *discr*) const [inline]

Return the discriminant value.

10.14.5.4.1 Parameters:

discr receive the discriminant value

10.14.5.4.2 Warning:

Must be called only by Medias

10.14.5.5 void LfP::Message::mediaSetData (const std::string & *data*) [inline]

Set the data value.

10.14.5.5.1 Parameters:

data new value for data

10.14.5.5.2 Warning:

Must be called only by Medias

10.14.5.6 void LfP::Message::mediaSetDiscriminant (const std::string & *discr*) [inline]

Set the discriminant value.

10.14.5.6.1 Parameters:

discr data new value for discriminant

10.14.5.6.2 Warning:

Must be called only by Medias

10.14.5.7 bool LfP::Message::operator< (const Message & *o*) [inline]

comparison operator

10.14.5.8 const Message & LfP::Message::operator= (const Message & *cp*)

Assignment operator.

10.14.5.9 template<typename V> void LfP::Message::setData (const V & *data*) [inline]

Set the data value.

10.14.5.9.1 Parameters:

data data new value for data

The type V must support operation void serialize(std::ostream&)

10.14.5.10 template<typename V> void LfP::Message::setDiscriminant (const V & *discr*) [inline]

Set the discriminant value.

10.14.5.10.1 Parameters:

discr data new value for discriminant

The type V must support operation void serialize(std::ostream&)

10.14.6Member Data Documentation

10.14.6.1 std::string LfP::Message::data_ [protected]

10.14.6.2 std::string LfP::Message::discr_ [protected]

10.14.6.3 std::string LfP::Message::functionName_ [protected]

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

151.**Message.h**
152.**Message.cpp**

10.14.6.4

10.15aonix_posix::mutex Class Reference

Wrapper for posix mutexes.

`#include <posix_mutex.h>`

Inheritance diagram for aonix_posix::mutex:

10.15.1Public Member Functions

153.**mutex** (const **attribute** &attr)

Create a mutex with the specified attribute.

154.**~mutex** ()

10.15.2Classes

155.class **attribute**

Wrapper for posix mutex attributes.

10.15.3Detailed Description

Wrapper for posix mutexes.

10.15.3.1.1 Author:

Antony Hubervic

10.15.4Constructor & Destructor Documentation

10.15.4.1 aonix_posix::mutex::mutex (const attribute & *attr*) `[inline]`

Create a mutex with the specified attribute.

10.15.4.1.1 Parameters:

attr attribute for the mutex

10.15.4.2 aonix_posix::mutex::~mutex () [inline]

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

156.**posix_mutex.h**

10.15.4.3

10.16aonix_posix::mutex::attribute Class Reference

Wrapper for posix mutex attributes.

`#include <posix_mutex.h>`

Inheritance diagram for aonix_posix::mutex::attribute:

10.16.1Public Member Functions

157.**attribute** ()
158.**attribute** (int type)
159.**~attribute** ()
160.void **set_type** (int type)
Set posix mutex type.

161.int **get_type** () const
Get posix mutex type.

10.16.2Detailed Description

Wrapper for posix mutex attributes.

10.16.3Constructor & Destructor Documentation

10.16.3.1 aonix_posix::mutex::attribute::attribute () `[inline]`

10.16.3.2 aonix_posix::mutex::attribute::attribute (int *type*) `[inline]`

10.16.3.3 aonix_posix::mutex::attribute::~attribute () `[inline]`

10.16.4Member Function Documentation

10.16.4.1 int aonix_posix::mutex::attribute::get_type () const [inline]

Get posix mutex type.

10.16.4.1.1 Returns:

posix mutex type

10.16.4.2 void aonix_posix::mutex::attribute::set_type (int *type*) [inline]

Set posix mutex type.

10.16.4.2.1 Parameters:

type posix mutex type (see posix doc)

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

162.**posix_mutex.h**

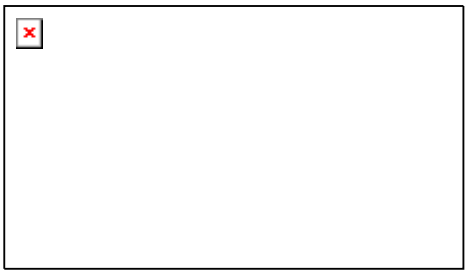
10.16.4.3

10.17aonix_posix::mutex_lock Class Reference

Create a lock on a posix mutex.

`#include <posix_mutex.h>`

Inheritance diagram for aonix_posix::mutex_lock:



10.17.1Public Member Functions

163.**mutex_lock** (**mutex** &m)
164.virtual **~mutex_lock** ()
165.**mutex** & **get_mutex** ()

Return the mutex acquired by this object.

10.17.2Protected Member Functions

166.**mutex_lock** (**mutex** &m, int dummy)

10.17.3Protected Attributes

167.**mutex** & **mutex_**

10.17.4Private Member Functions

168.**mutex_lock** (const **mutex_lock** &cp)
169.const **mutex_lock** & **operator=** (const **mutex_lock** &cp)

10.17.5Detailed Description

Create a lock on a posix mutex.

The lock is automatically released when the object is destroyed

---

10.17.6Constructor & Destructor Documentation

10.17.6.1 aonix_posix::mutex_lock::mutex_lock (mutex & *m*) [inline]

10.17.6.2 virtual aonix_posix::mutex_lock::~mutex_lock () [inline, virtual]

10.17.6.3 aonix_posix::mutex_lock::mutex_lock (mutex & *m*, int *dummy*) [inline, protected]

10.17.6.4 aonix_posix::mutex_lock::mutex_lock (const mutex_lock & *cp*) [private]

10.17.7Member Function Documentation

10.17.7.1 mutex& aonix_posix::mutex_lock::get_mutex () [inline]

Return the mutex acquired by this object.

10.17.7.1.1 Warning:

for internal use only

10.17.7.2 const mutex_lock& aonix_posix::mutex_lock::operator= (const mutex_lock & *cp*) [private]

10.17.8Member Data Documentation

10.17.8.1 mutex& aonix_posix::mutex_lock::mutex_ [protected]

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

170.**posix_mutex.h**

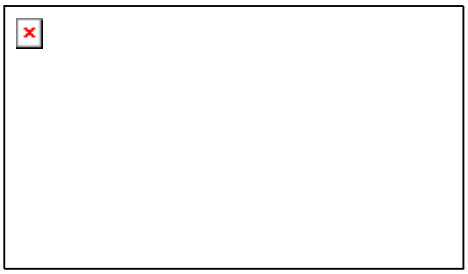
10.17.8.2

10.18aonix_posix::mutex_trylock Class Reference

Try to acquire a posix mutex.

`#include <posix_mutex.h>`

Inheritance diagram for aonix_posix::mutex_trylock:



10.18.1Public Member Functions

171.**mutex_trylock** (**mutex** &m)
172.bool **is_locked** () const

10.18.2Private Attributes

173.bool **is_locked_**

10.18.3Classes

174.struct **busy_domain**

10.18.4Detailed Description

Try to acquire a posix mutex.

10.18.4.1.1 Warning:

broken

10.18.5Constructor & Destructor Documentation

10.18.5.1 aonix_posix::mutex_trylock::mutex_trylock (mutex & *m*) `[inline]`

10.18.6Member Function Documentation

10.18.6.1 bool aonix_posix::mutex_trylock::is_locked () const [inline]

Specifies whether the mutex was acquired

10.18.6.1.1 Returns:

true if the mutex was acquired, false otherwise

10.18.7Member Data Documentation

10.18.7.1 bool aonix_posix::mutex_trylock::is_locked_ [private]

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

175.**posix_mutex.h**

10.18.7.2

10.19aonix_posix::mutex_trylock::busy_domain Struct Reference

10.19.1Static Public Member Functions

176.static bool **check** (int val)

10.19.2Member Function Documentation

10.19.2.1 static bool aonix_posix::mutex_trylock::busy_domain::check (int *val*) `[inline, static]`

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

177.**posix_mutex.h**

10.19.2.2

10.20aonix_util::native_holder< T > Class Template Reference

Helper class to hold native types.

`#include <posix_util.h>`

10.20.1Public Types

178.typedef T **native_type**

Native type.

10.20.2Public Member Functions

179.const **native_type** & **native** () const

Return a referenc to the native value.

180.**native_type** & **native** ()

Return a referenc to the native value.

10.20.3Protected Member Functions

181.**native_holder** (const **native_type** &n)

182.**native_holder** ()

10.20.4Protected Attributes

183.**native_type native_**

10.20.5Detailed Description

10.20.5.1 template<typename T> class aonix_util::native_holder< T >

Helper class to hold native types.

T : native type

10.20.6Member Typedef Documentation

10.20.6.1 template<typename T> typedef T aonix_util::native_holder< T >::native_type

Native type.

10.20.7Constructor & Destructor Documentation

10.20.7.1 template<typename T> aonix_util::native_holder< T >::native_holder (const native_type & *n*) [inline, protected]

10.20.7.2 template<typename T> aonix_util::native_holder< T >::native_holder () [inline, protected]

10.20.8Member Function Documentation

10.20.8.1 template<typename T> native_type& aonix_util::native_holder< T >::native () [inline]

Return a referenc to the native value.

10.20.8.1.1 Returns:

reference to the native value

10.20.8.2 template<typename T> const native_type& aonix_util::native_holder< T >::native () const [inline]

Return a referenc to the native value.

10.20.8.2.1 Returns:

reference to the native value

10.20.9Member Data Documentation

10.20.9.1 template<typename T> native_type aonix_util::native_holder< T >::native_ [protected]

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

184.**posix_util.h**

10.20.9.2

10.21aonix_util::non_copyable< T > Class Template Reference

Helper class to create non copyable classes.

`#include <posix_util.h>`

10.21.1Protected Member Functions

185.**non_copyable** ()

10.21.2Private Member Functions

186.**non_copyable** (const **non_copyable**< T > &cp)
187.const **non_copyable**< T > & **operator=** (const **non_copyable**< T > &cp)

10.21.3Detailed Description

10.21.3.1 template<typename T> class aonix_util::non_copyable< T >

Helper class to create non copyable classes.

Must be inherited by classes

10.21.4Constructor & Destructor Documentation

10.21.4.1 template<typename T> aonix_util::non_copyable< T >::non_copyable () `[inline, protected]`

10.21.4.2 template<typename T> aonix_util::non_copyable< T >::non_copyable (const non_copyable< T > & *cp*) `[private]`

10.21.5Member Function Documentation

10.21.5.1 template<typename T> const non_copyable<T>& aonix_util::non_copyable< T >::operator= (const non_copyable< T > & *cp*) `[private]`

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

188.**posix_util.h**

10.21.5.2

10.22LfP::null_discriminant Struct Reference

Default discriminant type.

`#include <Binder.h>`

10.22.1Public Member Functions

189. void **serialize** (std::ostream &os)
Serailization function.

190. void **deserialize** (std::istream &is)
Deserialization function.

191. bool **operator==** (const **null_discriminant** &o) const
Equality comparison operator.

192. template<class V> bool **operator==** (const V &o) const
Templated equality comparison operator.

10.22.2Detailed Description

Default discriminant type.

This class will successfully compare to any other class

10.22.3Member Function Documentation

10.22.3.1 void LfP::null_discriminant::deserialize (std::istream & *is*) `[inline]`

Deserialization function.

Does nothing

10.22.3.2 template<class V> bool LfP::null_discriminant::operator== (const V & *o*) const [inline]

Templated equality comparison operator.

10.22.3.2.1 Returns:

true

Always return true. The operator is templated so this class can be succesfully compared to any other class. This allows to pass any kind of guard based on equality comparison

10.22.3.3 bool LfP::null_discriminant::operator== (const null_discriminant & *o*) const [inline]

Equality comparison operator.

10.22.3.3.1 Returns:

true

Always return true

10.22.3.4 void LfP::null_discriminant::serialize (std::ostream & *os*) [inline]

Serailization function.

Does nothing

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

193.**Binder.h**

10.22.3.5

10.23aonix_util::null_domain Struct Reference

Check if a value is null.

```
#include <posix_util.h>
```

10.23.1Static Public Member Functions

194.static bool **check** (int value)

10.23.2Detailed Description

Check if a value is null.

10.23.3Member Function Documentation

10.23.3.1 static bool aonix_util::null_domain::check (int *value*) `[inline, static]`

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

195.**posix_util.h**

10.23.3.2

10.24LfP::Port< D > Class Template Reference

Base class for **LfP** ports.

`#include <Port.h>`

10.24.1Public Types

196. typedef D **discriminant_type**
Discriminant type used in this port.

197. typedef **Binder**< D >::Guard **guard_type**
Guard type for discriminants used in this port.

10.24.2Public Member Functions

198. **Port** ()
Create a port with no attached binder.

199. **Port** (**Binder**< D > &binder)
Create a port attached to a specific binder.

200. **Port** (const **Port**< D > &cp)
Copy constructor.

201. virtual **~Port** ()
Destructor.

202. void **mediaReadMessage** (**Message** &msg, const **guard_type** *const guard=0)
Read a message from a media point of view.

203. void **mediaWriteMessage** (**Message** &msg)
Write a message from a media point of view.

204. template<typename P> void **waitMethod** (const std::string &methodName, **discriminant_type** &discr, P &params, const **guard_type** *const guard=0)

Wait for an incoming method call.

205. template<typename V> void **simpleSend** (**discriminant_type** &discr, V &value)
Send a simple message.

206. template<typename P> void **asynchronousCall** (const std::string &methodName, **discriminant_type** &discr, const P &param)
Asynchronous method call.

207. template<typename P, typename V> void **synchronousCall** (const std::string &methodName, **discriminant_type** &discr, const P &param, V &retval)
Synchronous method call.

10.24.3Protected Member Functions

208. void **setBinder** (**Binder**< D > &binder)
Helper function to attach this port to a binder.

10.24.4Private Attributes

209. **Binder**< D > * **binder_**

10.24.5Classes

210. struct **IsEqualGuard**
Simple guard which tests equality between 2 discriminants.

10.24.6Detailed Description

10.24.6.1 template<typename D> class LfP::Port< D >

Base class for **LfP** ports.

Template parameter D is the discriminant type. Must support serialization

10.24.7Member Typedef Documentation

10.24.7.1 template<typename D> typedef D LfP::Port< D >::discriminant_type

Discriminant type used in this port.

10.24.7.2 template<typename D> typedef Binder<D>::Guard LfP::Port< D >::guard_type

Guard type for discriminants used in this port.

10.24.8Constructor & Destructor Documentation

10.24.8.1 template<typename D> LfP::Port< D >::Port () `[inline]`

Create a port with no attached binder.

10.24.8.1.1 Warning:

for internal use only

10.24.8.2 template<typename D> LfP::Port< D >::Port (Binder< D > & *binder*) `[inline]`

Create a port attached to a specific binder.

10.24.8.2.1 Parameters:

binder binder to attach to

10.24.8.2.2 Warning:

the binder passed in the parameter must live as long as the port exists. However, the binder lifecycle is not managed by the port
this constructor must not be used by a child class if the binder is an attribute of this child class (init order pb)

10.24.8.3 template<typename D> LfP::Port< D >::Port (const Port< D > & *cp*) `[inline]`

Copy constructor.

10.24.8.3.1 Warning:

the two ports will use the same binder

10.24.8.4 template<typename D> virtual LfP::Port< D >::~Port () [inline, virtual]

Destructor.

10.24.9Member Function Documentation

10.24.9.1 template<typename D> template<typename P> void LfP::Port< D >::asynchronousCall (const std::string & *methodName*, discriminant_type & *discr*, const P & *param*) [inline]

Asynchronous method call.

10.24.9.1.1 Parameters:

methodName name of the method
discr discriminant of the outgoing message
param in parameters for the called function D : discriminant type. Must support void serialize(std::ostream&) P : type holding method in parameters. Must support void serialize(std::ostream&)

10.24.9.2 template<typename D> void LfP::Port< D >::mediaReadMessage (Message & *msg*, const guard_type *const *guard* = 0) [inline]

Read a message from a media point of view.

10.24.9.2.1 Parameters:

msg receive the message which was read
guard guard to check if the message has the correct discriminant

10.24.9.2.2 Warning:

must be called only in medias

Blocking if queue is empty

10.24.9.3 template<typename D> void LfP::Port< D >::mediaWriteMessage (Message & *msg*) [inline]

Write a message from a media point of view.

10.24.9.3.1 Parameters:

msg message to be sent

10.24.9.3.2 Warning:

must be called only in medias

blocking if queue is full

10.24.9.4 template<typename D> void LfP::Port< D >::setBinder (Binder< D > & *binder*) [inline, protected]

Helper function to attach this port to a binder.

Must be called by children classes, preferably in their constructors

10.24.9.5 template<typename D> template<typename V> void LfP::Port< D >::simpleSend (discriminant_type & *discr*, V & *value*) [inline]

Send a simple message.

10.24.9.5.1 Parameters:

value message data
discr discriminant of the outgoing message

Can be used to send a return value Type parameters D : discriminant type. Must support void serialize(std::ostream&) V : message data type. Must support void serialize(std::ostream&)

10.24.9.6 template<typename D> template<typename P, typename V> void LfP::Port< D >::synchronousCall (const std::string & *methodName*, discriminant_type & *discr*, const P & *param*, V & *retval*) [inline]

Synchronous method call.

10.24.9.6.1 Parameters:

methodName name of the method
discr discriminant of the outgoing message
param in parameters for the called function
retval inout/out/return parameters for the called function D : discriminant type. Must support void serialize(std::ostream&) P : type holding method in parameters. Must support void serialize(std::ostream&) V : type holding method inout/out/return parameters. Must support void deserialize(std::istream&)

10.24.9.7 template<typename D> template<typename P> void LfP::Port< D >::waitMethod (const std::string & *methodName*, discriminant_type & *discr*, P & *params*, const guard_type *const *guard* = 0) [inline]

Wait for an incoming method call.

10.24.9.7.1 Parameters:

methodName name of the method (not used currently)
discr receive the discriminant of the incoming message
params receive the paramaters for the method call
guard guard on the incoming discriminant

10.24.9.7.2 Todo:

check method name

Type parameters P : type holding method in parameters. Must support void deserialize(std::istream&)

10.24.10 Member Data Documentation

10.24.10.1 template<typename D> Binder<D>* LfP::Port< D >::binder_ [private]

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

211. **Port.h**

10.24.10.2

10.25LfP::Port< D >::IsEqualGuard Struct Reference

Simple guard which tests equality between 2 discriminants.

`#include <Port.h>`

Inheritance diagram for LfP::Port< D >::IsEqualGuard:

10.25.1Public Member Functions

212. **IsEqualGuard** (const **discriminant_type** &discr)
213. bool **check** (const **discriminant_type** &discr) const

Check that the discriminant of an incoming message matches some conditions.

10.25.2Private Attributes

214. **discriminant_type discr_**

10.25.3Detailed Description

10.25.3.1 template<typename D> struct LfP::Port< D >::IsEqualGuard

Simple guard which tests equality between 2 discriminants.

10.25.4Constructor & Destructor Documentation

10.25.4.1 template<typename D> LfP::Port< D >::IsEqualGuard::IsEqualGuard (const discriminant_type & *discr*) `[inline]`

10.25.5Member Function Documentation

10.25.5.1 template<typename D> bool LfP::Port< D >::IsEqualGuard::check (const discriminant_type & *discr*) const `[inline, virtual]`

Check that the discriminant of an incoming message matches some conditions.

10.25.5.1.1 Parameters:

discr discriminant of the incoming message

10.25.5.1.2 Returns:

true if the discriminant matches the condition, false otherwise

Implements **LfP::Binder< D >::Guard** (*p.15*).

10.25.6Member Data Documentation

10.25.6.1 template<typename D> discriminant_type LfP::Port< D >::IsEqualGuard::discr_ [private]

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

215.**Port.h**

10.25.6.2

10.26aonix_util::positive_domain Struct Reference

Check if a value is positive or null.

`#include <posix_util.h>`

10.26.1Static Public Member Functions

216.static bool **check** (int value)

10.26.2Detailed Description

Check if a value is positive or null.

10.26.3Member Function Documentation

10.26.3.1 static bool aonix_util::positive_domain::check (int *value*) `[inline, static]`

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

217.**posix_util.h**

10.26.3.2

10.27aonix_util::return_value_handler< K, T > Struct Template Reference

Helper class to deal with return values.

```
#include <posix_util.h>
```

10.27.1Public Types

218.typedef K **domain_checker**

Type for checking whether the input value is valid.

10.27.2Public Member Functions

219.**return_value_handler** (const T &value)

Create a cheker with the specified input value.

220.**operator int** ()

10.27.3Private Attributes

221.T **value_**

10.27.4Detailed Description

10.27.4.1 template<typename K, typename T = int> struct aonix_util::return_value_handler< K, T >

Helper class to deal with return values.

When converted to int, throw an exception if the value is not inside the specified domain sample use : int val=return_value_handler<some_domain>(somefunction()); Type K must support : static bool check(const T&)

10.27.5Member Typedef Documentation

10.27.5.1 template<typename K, typename T = int> typedef K aonix_util::return_value_handler< K, T >::domain_checker

Type for checking whether the input value is valid.

10.27.6Constructor & Destructor Documentation

10.27.6.1 template<typename K, typename T = int> aonix_util::return_value_handler< K, T >::return_value_handler (const T & *value*) [inline]

Create a cheker with the specified input value.

10.27.6.1.1 Parameters:

value value to be checked

10.27.7Member Function Documentation

10.27.7.1 template<typename K, typename T = int> aonix_util::return_value_handler< K, T >::operator int () [inline]

Check the input value w.r.t. the domain

10.27.7.1.1 Returns:

0 if the value is valid

10.27.7.1.2 Exceptions:

bad_result_exception if the value is not in the specified domain

---

10.27.8Member Data Documentation

10.27.8.1 template<typename K, typename T = int> T aonix_util::return_value_handler< K, T >::value_ [private]

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

222.**posix_util.h**

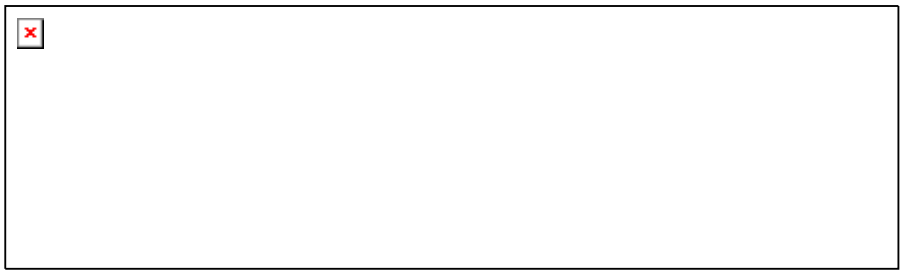
10.27.8.2

10.28aonix_posix::thread Class Reference

Wrapper class for posix threads.

`#include <posix_thread.h>`

Inheritance diagram for aonix_posix::thread:



10.28.1Public Member Functions

223. **thread** (const **attribute** &attr)
   Create a thread with the specified attribute.

224. **thread** ()
   Create a thread with default attribute.

225. virtual **~thread** ()
   Destructor.

226. bool **operator==** (const **thread** &other)
   Test equality of two threads.

227. void **get_schedparam** (int &policy, sched_param &param) const
   Return the posix scheduling parameters.

228. void **cancel** ()
   Cancel the thread.

229. void * **join** ()
   Wait until the thread is terminated.

230. void **start** (void *arg=0)
Start the thread.

231. virtual void **run** (void *arg)=0
Main function for the thread.

10.28.2Static Public Member Functions

232. static int **setcancelstate** (int val)
Return current thread Set scheduling policy.

233. static int **setcanceltype** (int val)
Set scheduling policy.

234. static void **testcancel** ()

10.28.3Private Member Functions

235. **thread** (const pthread_t &native)

10.28.4Private Attributes

236. **attribute attr_**
237. **thread_data_type thread_data**

10.28.5Classes

238. struct **attribute**
Performs a function only once Wrapper for posix thread attributes.

239. struct **thread_data_type**

10.28.6Detailed Description

Wrapper class for posix threads.

10.28.6.1.1 Author:

Antony Hubervic

10.28.7Constructor & Destructor Documentation

10.28.7.1 aonix_posix::thread::thread (const attribute & *attr*)

Create a thread with the specified attribute.

10.28.7.1.1 Parameters:

attr thread attribute

10.28.7.2 aonix_posix::thread::thread ()

Create a thread with default attribute.

10.28.7.3 aonix_posix::thread::~thread () `[virtual]`

Destructor.

The underlying (native) thread object is not destroyed

10.28.7.4 aonix_posix::thread::thread (const pthread_t & *native*) `[inline, private]`

10.28.8Member Function Documentation

10.28.8.1 void aonix_posix::thread::cancel ()

Cancel the thread.

10.28.8.2 void aonix_posix::thread::get_schedparam (int & *policy*, sched_param & *param*) const

Return the posix scheduling parameters.

10.28.8.2.1 Parameters:

policy receive the posix scheduling policy
param receive the posix scheduling parameters

10.28.8.3 void * aonix_posix::thread::join ()

Wait until the thread is terminated.

10.28.8.3.1 Returns:

thread return value

10.28.8.4 bool aonix_posix::thread::operator== (const thread & *other*) [inline]

Test equality of two threads.

10.28.8.4.1 Parameters:

other thread to compare tp

10.28.8.4.2 Returns:

true if both objects identify the same thread

10.28.8.5 virtual void aonix_posix::thread::run (void * *arg*) [pure virtual]

Main function for the thread.

10.28.8.5.1 Parameters:

arg parameters for the thread execution

Must be implemented by subclasses

10.28.8.6 static int aonix_posix::thread::setcancelstate (int *val*) [inline, static]

Return current thread Set scheduling policy.

10.28.8.6.1 Parameters:

val posix scheduling policy

10.28.8.6.2 Exceptions:

bad_result_exception if underlying posix function fails

10.28.8.7 static int aonix_posix::thread::setcanceltype (int *val*) [inline, static]

Set scheduling policy.

10.28.8.7.1 Parameters:

val posix scheduling policy

10.28.8.7.2 Exceptions:

bad_result_exception if underlying posix function fails

10.28.8.8 void aonix_posix::thread::start (void * *arg* = `0`)

Start the thread.

10.28.8.8.1 Parameters:

arg parameter for the main function of the thread

10.28.8.9 static void aonix_posix::thread::testcancel () `[inline, static]`

10.28.9Member Data Documentation

10.28.9.1 attribute aonix_posix::thread::attr_ `[private]`

10.28.9.2 thread_data_type aonix_posix::thread::thread_data `[private]`

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

240.**posix_thread.h**
241.**posix_thread.cpp**

10.28.9.3

10.29aonix_posix::thread::attribute Struct Reference

Performs a function only once Wrapper for posix thread attributes.

`#include <posix_thread.h>`

Inheritance diagram for aonix_posix::thread::attribute:

10.29.1Public Member Functions

242. **attribute** ()
243. **~attribute** ()
244. void **set_schedparam** (int prio, int policy)
    Set scheduling parameters.

245. void **set_schedpolicy** (int arg)
    Set scheduling policy.

246. int **get_schedpolicy** () const
    Return scheduling policy.

247. void **set_scope** (int arg)
    Set thread scope.

248. int **get_scope** () const
    Return thread scope.

249. void **set_inheritsched** (int arg)
    Set scheduling inheritance policy.

250. int **get_inheritsched** () const
    Return scheduling inheritance policy.

251. void **set_detachstate** (int arg)
Set detach state for the thread.

252. int **get_detachstate** () const
Return detach state for the thread.

253. void **set_schedparam** (sched_param &arg)
Set scheduling parameters.

254. void **get_shedparam** (sched_param &retval) const
Return scheduling parameters.

10.29.2 Detailed Description

Performs a function only once Wrapper for posix thread attributes.

10.29.3 Constructor & Destructor Documentation

10.29.3.1 aonix_posix::thread::attribute::attribute () `[inline]`

10.29.3.2 aonix_posix::thread::attribute::~attribute () `[inline]`

10.29.4 Member Function Documentation

10.29.4.1 int aonix_posix::thread::attribute::get_detachstate () const `[inline]`

Return detach state for the thread.

10.29.4.1.1 Returns:

posix detach state for the thread

10.29.4.1.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.2 int aonix_posix::thread::attribute::get_inheritsched () const [inline]

Return scheduling inheritance policy.

10.29.4.2.1 Returns:

posix scheduling inheritance policy

10.29.4.2.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.3 int aonix_posix::thread::attribute::get_schedpolicy () const [inline]

Return scheduling policy.

10.29.4.3.1 Returns:

posix scheduling policy

10.29.4.3.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.4 int aonix_posix::thread::attribute::get_scope () const [inline]

Return thread scope.

10.29.4.4.1 Returns:

posix thread scope

10.29.4.4.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.5 void aonix_posix::thread::attribute::get_shedparam (sched_param & *retval*) const [inline]

Return scheduling parameters.

10.29.4.5.1 Parameters:

retval posix scheduling parameters

10.29.4.5.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.6 void aonix_posix::thread::attribute::set_detachstate (int *arg*) [inline]

Set detach state for the thread.

10.29.4.6.1 Parameters:

arg posix detach state for the thread

10.29.4.6.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.7 void aonix_posix::thread::attribute::set_inheritsched (int *arg*) [inline]

Set scheduling inheritance policy.

10.29.4.7.1 Parameters:

arg posix scheduling inheritance policy

10.29.4.7.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.8 void aonix_posix::thread::attribute::set_schedparam (sched_param & *arg*) [inline]

Set scheduling parameters.

10.29.4.8.1 Parameters:

arg posix scheduling parameters

10.29.4.8.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.9 void aonix_posix::thread::attribute::set_schedparam (int *prio*, int *policy*) [inline]

Set scheduling parameters.

10.29.4.9.1 Parameters:

prio posix priority
policy posix scheduling policy

10.29.4.10 void aonix_posix::thread::attribute::set_schedpolicy (int *arg*) [inline]

Set scheduling policy.

10.29.4.10.1 Parameters:

arg posix scheduling policy

10.29.4.10.2 Exceptions:

bad_result_exception if underlying posix function fails

10.29.4.11 void aonix_posix::thread::attribute::set_scope (int *arg*) [inline]

Set thread scope.

10.29.4.11.1 Parameters:

arg posix thread scope

10.29.4.11.2 Exceptions:

bad_result_exception if underlying posix function fails

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

255. **posix_thread.h**

10.29.4.12

10.30aonix_posix::thread::thread_data_type Struct Reference

`#include <posix_thread.h>`

10.30.1Public Attributes

256.**aonix_posix::thread** * **thread_object**
257.void * **arg**

10.30.2Member Data Documentation

10.30.2.1 void* aonix_posix::thread::thread_data_type::arg

10.30.2.2 aonix_posix::thread* aonix_posix::thread::thread_data_type::thread_object

---

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

258.**posix_thread.h**

11 LfPC++runtime File Documentation

11.1 Binder.h File Reference

```
#include <iostream>
#include "posix_cond.h"
#include "Message.h"
```

11.1.1 Namespaces

259.namespace **LfP**

11.1.2 Classes

260.struct **LfP::null_discriminant**

Default discriminant type.

261.class **LfP::Binder< D >**

*Base class for **LfP** binders.*

262.struct **LfP::Binder< D >::Guard**

Interface type for the guards to check the discriminant of incoming messages.

263.class **LfP::GenericBinder< Q, D >**

Generic binder implementation.

11.2 Class.cpp File Reference

```
#include "Class.h"
```

11.2.1 Namespaces

264.namespace **LfP**

11.3 Class.h File Reference

```
#include "posix_thread.h"
```

11.3.1 Namespaces

265.namespace **LfP**

11.3.2 Classes

266.class **LfP::Class**

*Base class for **LfP** classes.*

11.4 Media.cpp File Reference

```
#include "Media.h"
#include "Port.h"
```

11.4.1 Namespaces

267.namespace **LfP**

11.5 Media.h File Reference

`#include "posix_thread.h"`

11.5.1 Namespaces

268.namespace **LfP**

11.5.2 Classes

269.class **LfP::Media**

*Base class for **LfP** medias.*

11.6 Message.cpp File Reference

```
#include "Message.h"
```

11.6.1 Namespaces

270.namespace **LfP**

11.7 Message.h File Reference

```
#include <string>
#include <sstream>
#include "types.h"
```

11.7.1 Namespaces

271.namespace **LfP**

11.7.2 Classes

272.class **LfP::Message**

***Class** for **LfP** messages.*

11.8 Port.h File Reference

```
#include <sstream>
#include <assert.h>
#include "posix_mutex.h"
#include "posix_cond.h"
#include "types.h"
#include "Message.h"
#include "Binder.h"
```

11.8.1 Namespaces

273.namespace **LfP**

11.8.2 Classes

274.class **LfP::Port< D >**

*Base class for **LfP** ports.*

275.struct **LfP::Port< D >::IsEqualGuard**

Simple guard which tests equality between 2 discriminants.

11.9 posix_cond.cpp File Reference

```
#include "posix_cond.h"
#include "posix_util.h"
```

11.9.1 Namespaces

276.namespace **aonix_posix**

11.10posix_cond.h File Reference

```
#include <pthread.h>
#include "posix_mutex.h"
```

11.10.1Namespaces

277.namespace **aonix_posix**

11.10.2Classes

278.class **aonix_posix::cond**

Wrapper class for posix condition variables.

279.struct **aonix_posix::cond::attribute**

Wrapper for posix condition variable attributes.

280.class **aonix_posix::cond2**

Wrapper class for posix condition variables This class has its own mutex to synchronize accesses to the condition variable.

11.11posix_mutex.cpp File Reference

```
#include "posix_mutex.h"
#include "posix_util.h"
```

11.12posix_mutex.h File Reference

```
#include <pthread.h>
#include <errno.h>
#include "posix_util.h"
```

11.12.1Namespaces

281.namespace **aonix_posix**

11.12.2Classes

282.class **aonix_posix::mutex**
Wrapper for posix mutexes.

283.class **aonix_posix::mutex::attribute**
Wrapper for posix mutex attributes.

284.class **aonix_posix::mutex_lock**
Create a lock on a posix mutex.

285.class **aonix_posix::mutex_trylock**
Try to acquire a posix mutex.

286.struct **aonix_posix::mutex_trylock::busy_domain**

11.13posix_thread.cpp File Reference

```
#include "posix_thread.h"
```

11.13.1Namespaces

287.namespace **aonix_posix**

11.13.2Functions

288.void * **thread_main** (void *arg)

11.13.3Function Documentation

11.13.3.1 void* thread_main (void * *arg*)

11.13.3.2

11.14posix_thread.h File Reference

```
#include <pthread.h>
#include "posix_util.h"
```

11.14.1Namespaces

289.namespace **aonix_posix**

11.14.2Classes

290.class **aonix_posix::thread**

Wrapper class for posix threads.

291.struct **aonix_posix::thread::attribute**

Performs a function only once Wrapper for posix thread attributes.

292.struct **aonix_posix::thread::thread_data_type**

11.15posix_util.h File Reference

11.15.1Namespaces

293. namespace **aonix_util**

11.15.2Classes

294. struct **aonix_util::bad_result_exception**
exception class

295. struct **aonix_util::positive_domain**
Check if a value is positive or null.

296. struct **aonix_util::null_domain**
Check if a value is null.

297. struct **aonix_util::return_value_handler< K, T >**
Helper class to deal with return values.

298. class **aonix_util::native_holder< T >**
Helper class to hold native types.

299. class **aonix_util::non_copyable< T >**
Helper class to create non copyable classes.

11.16queues.h File Reference

```
#include <set>
#include <list>
#include <cassert>
#include "Message.h"
```

11.16.1Namespaces

300.namespace **LfP**

11.16.2Classes

301.struct **LfP::fifo< size >**

Class *for fifo queuing policy inside binders.*

302.struct **LfP::bag< size >**

Class *for bag queuing policy inside binders.*

11.17types.h File Reference

`#include <iostream>`

11.17.1Namespaces

303.namespace **LfP**

11.17.2Classes

304.class **LfP::integer**

LfP *integer datatype.*

305.class **LfP::boolean**

LfP *boolean datatype.*

11.17.3Functions

306.integer **LfP::succ** (const integer &o)

Utility free function to compute successor of an integer.

307.integer **LfP::pred** (const integer &o)

Utility free function to compute predecessor of an integer.

308.std::ostream & **LfP::operator**<< (std::ostream &os, const integer &i)
309.std::istream & **LfP::operator**>> (std::istream &is, integer &i)

12 LfPC++runtime Page Documentation

12.1 Todo List

*12.1.1.1.1 Member LfP::Port::waitMethod (const std::string &methodName, discriminant_type &discr, P &params, const guard_type *const guard=0)*

check method name

13 Index