



institut de recherche en informatique  
et systèmes aléatoires

# bRTP : BIBLIOTHEQUE RTP

## MANUEL D'UTILISATION

Version: 1.0

Date : 20 Juillet 2006.

Auteur(s):

Adrien Schadle





# Table des matières

<b>1. PRESENTATION DE bRTP</b>	<b>9</b>
<b>2. RTP / RTCP : PRINCIPE</b>	<b>10</b>
2.1. CONSTITUTION D'UN PAQUET RTP	10
2.2. CONSTITUTION D'UN PAQUET RTCP	11
2.2.1. RTCP SR	11
2.2.2. RTCP RR	12
<b>3. EXEMPLES SIMPLES : EMETTEUR – RECEPTEUR</b>	<b>14</b>
3.1. EMETTEUR	14
3.2. RECEPTEUR	16
<b>4. ARCHITECTURE GENERALE</b>	<b>19</b>
<b>5. DESCRIPTION DES OBJETS</b>	<b>20</b>
5.1. CRTPSSESSION	20
5.1.1. Description	20
5.1.2. Méthodes	20
5.2. CRTPTRANSMITTER	24
5.2.1. Description	24
5.2.2. Méthodes	24
5.2.3. Héritage	29
5.3. CRTPPACKET	29
5.3.1. Description	29
5.3.2. Méthodes	29
5.4. CRTCPPACKET	33
5.4.1. Description	33
5.4.2. Méthodes	33
5.5. CRTCPSRPACKET	36
5.5.1. Description	36
5.5.2. Méthodes	36
5.6. CRTCPRRPACKET	40
5.6.1. Description	40
5.6.2. Méthodes	40
5.7. CRTPTIME	44
5.7.1. Description	44
5.7.2. Méthodes	44
<b>6. UTILISER bRTP DANS SON PROJET</b>	<b>46</b>
6.1. FICHIERS NECESSAIRES	46
6.2. CONTRAINTES D'UTILISATION	46

6.3. FONCTIONNALITES NON IMPLEMENTEES ..... 46

## Table des figures

Figure 1 : En-tête d'un paquet RTP .....	10
Figure 2 : En-tête d'un paquet RTCP SR .....	11
Figure 3 : En-tête d'un paquet RTCP RR .....	12
Figure 4 : Diagramme de classe de la librairie bRTP .....	19



## **Glossaire**

DLSR : Delay since Last Sender Report.

LSR : Last Sender Report. Instant du dernier paquet RTCP SR.

LSW : Least Significant Word.

MSW : Most Significant Word.

NTP : Network Time Protocol.

RTCP : RTP Control Protocol.

RTCP SR : RTP Control Protocol Sender Report.

RTCP RR : RTP Control Protocol Receiver Report.

RTP : Real Time Protocol.



# 1. Présentation de bRTP

---

« bRTP » est une librairie Windows qui implémente les mécanismes RTP / RTCP décrits dans le RFC 3550. « bRTP » fonctionne aussi bien pour un émetteur que pour un récepteur sur les couches de transports UDP et UDP-Lite<sup>1</sup> en IPv4 et en IPv6. La bibliothèque fonctionne uniquement en unicast.

Le manuel d'utilisation de la bibliothèque « bRTP » contient 5 chapitres :

- **RTP / RTCP : principe** décrit les mécanismes des échanges RTP et RTCP,
- **Exemple simple : émetteur - récepteur** montre une application simple exploitant les fonctionnalités de la bibliothèque « bRTP »,
- **Architecture Générale** présente la conception de la bibliothèque « bRTP »,
- **Description des objets** qui détaille les objets qui constituent la librairie « bRTP »,
- **Utiliser « bRTP » dans son projet** indique tous les éléments nécessaires au bon fonctionnement de la bibliothèque « bRTP » et les fonctionnalités non implémentées dans « bRTP ».

---

<sup>1</sup> L'utilisation d'UDP-Lite comme protocole de transport requiert les bibliothèques dynamiques Wull (IPv4) et Wull6 (IPv6).

## 2. RTP / RTCP : principe

---

Le protocole RTP est un protocole temps réel non fiable qui permet de transporter des informations multimédia. Le protocole RTP fournit :

- L'identification du type des informations multimédia,
- La numérotation du séquençement,
- L'horodatage des paquets RTP émis.

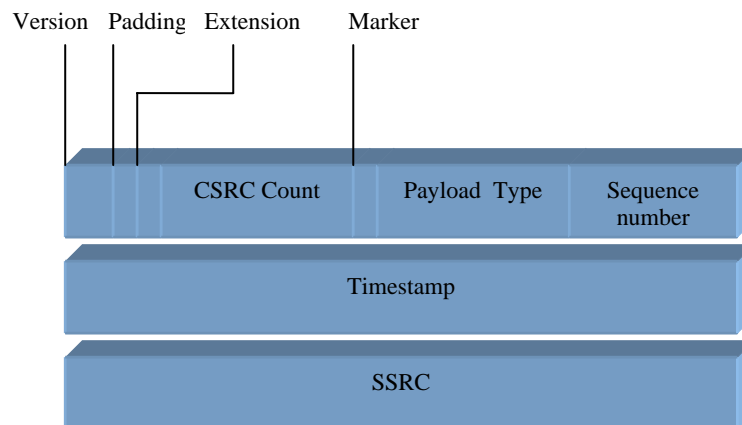
RTCP est le protocole de contrôle associé au protocole RTP. Les paquets RTCP sont émis périodiquement et véhiculent des informations statistiques sur l'état de la transmission entre chaque participant. Ces informations statistiques permettent de calculer des informations de QoS :

- Gigue : temps d'inter arrivée, permet d'estimer l'écart entre l'arrivée des paquets et leur moment de lecture,
- RTT : durée que met un paquet pour faire un aller retour sur le réseau,
- Taux de perte de paquet : estimation du taux de paquet perdu.

Les paragraphes suivant rappellent la constitution de chaque en-tête.

### 2.1. Constitution d'un paquet RTP

Le schéma ci-dessous rappelle la constitution de l'en-tête d'un paquet RTP.



*Figure 1 : En-tête d'un paquet RTP*

Version : Numéro de version du protocole RTP utilisé.

Padding : bit qui indique si des octets sont utilisés pour du padding.

Extension : bit qui indique si le protocole RTP est étendu.

CSRC count : Nombre de contributeurs.

Marker : bit dont l'usage est à définir par l'utilisateur.

Payload Type : Identifiant sur la nature des données multimédia (cf RFC 3551)

Sequence Number : Numéro de séquence du paquet RTP.

Timestamp : Indicateur sur l'instant de lecture du paquet RTP.

SSRC : Identifiant de l'émetteur du paquet RTP.

## 2.2. Constitution d'un paquet RTCP

Dans la bibliothèque « bRTP » deux paquets RTCP sont implémentés :

- SR : Sender Report,
- RR : Receiver Report.

Les en-têtes sont rappelés dans les sous paragraphes suivants.

### 2.2.1. RTCP SR

Les paquets RTCP SR sont émis périodiquement par les applications qui émettent les paquets RTP. L'en-tête d'un paquet RTCP SR se présente de la manière suivante.

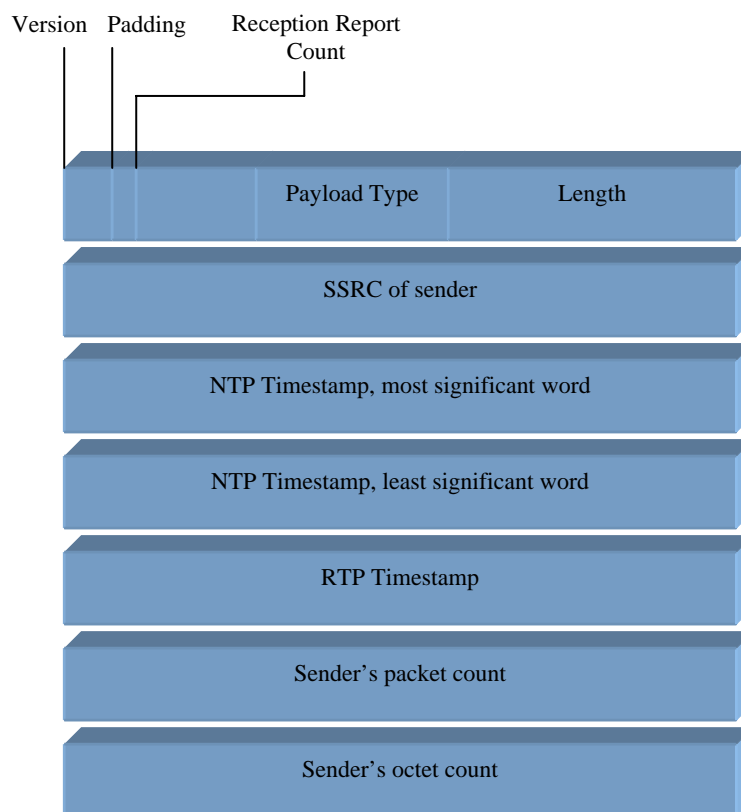


Figure 2 : En-tête d'un paquet RTCP SR

Version : Numéro de version du protocole RTP utilisé.

Padding : bit qui indique si des octets sont utilisés pour du padding.

Reception Report Count : Nombre de « Reception Report » contenu dans l'en-tête du paquet RTCP.

Payload Type : Identifiant sur la nature des données multimédia (200 pour du RTCP SR)

Length : Longueur en octet de l'en-tête RTCP.

SSRC : Identifiant de l'émetteur du paquet RTCP.

NTP timestamp MSW : Partie entière de l'instant d'émission du paquet RTCP SR

NTP timestamp LSW : Partie décimale de l'instant d'émission du paquet RTCP SR

RTP timestamp : Combinaison des 16 bits de poids faible du NTP timestamp MSW et 16 bits de poids fort du NTP timestamp LSW.

Sender's packet count : Nombre de paquets émis par l'émetteur.

Sender's octet count : Nombre d'octets émis par l'émetteur.

### 2.2.2. RTCP RR

Les paquets RTCP RR sont les paquets émis périodiquement par les applications réceptrices de paquets RTP. L'en-tête des paquets RTCP RR se présente de la manière suivante.

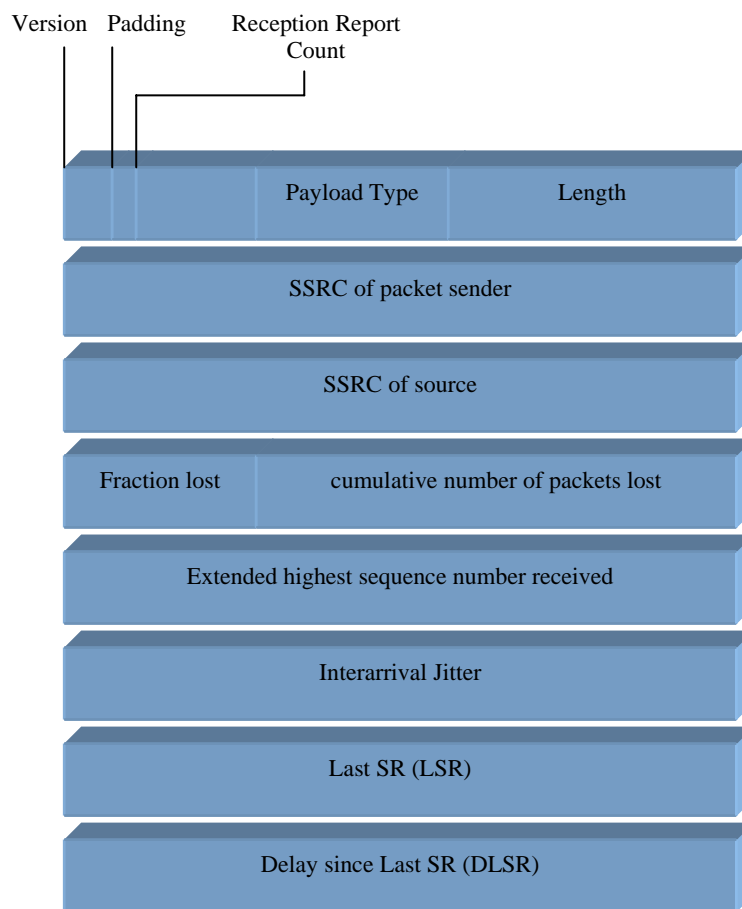


Figure 3 : En-tête d'un paquet RTCP RR

Version : Numéro de version du protocole RTP utilisé.

Padding : bit qui indique si des octets sont utilisés pour du padding.

Reception Report Count : Nombre de « Reception Report » contenu dans l'en-tête du paquet RTCP.

Payload Type : Identifiant sur la nature des données multimédia (200 pour du RTCP SR)

Length : Longueur en octet de l'en-tête RTCP.

SSRC : Identifiant de l'émetteur du paquet RTCP.

SSRC of source : Identifiant de l'émetteur de paquet RTCP SR

Fraction lost : Taux de paquets RTP perdus (sur 255).

Cumulative number packets lost : nombre de paquets perdus (codé du 24 bits)

Extended highest sequence number received : Nombre de fois où la valeur précédente est revenue à 0 (après avoir utilisé toutes les valeurs des 24 bits).

Interarrival Jitter : Gigue d'inter arrivée.

Last SR : RTP Timestamp du dernier paquet RTCP SR reçu.

DLSR : Délai écoulé avant l'émission du paquet RTCP RR dans l'échelle du RTPTimestamp.

### 3. Exemples simples : émetteur – récepteur

---

Ce chapitre propose un exemple simple de programme qui émet des paquets RTP et un exemple de programme qui reçoit les paquets RTP. Les deux programmes utilisent la librairie « bRTP ». Le code source des exemples est livré avec « bRTP ». Les noms des projets sont respectivement « RTPWullSend » et « RTPWullReceive ».

#### 3.1. Emetteur

Le programme ci-dessous implémente une émission de paquets RTP toute les deux millisecondes en utilisant la librairie « bRTP ». L'émission des paquets RTP est réalisée sur le protocole de transport UDP-Lite. Le programme prend trois paramètres :

- Adresse IPv4 du destinataire,
- Port RTP du destinataire,
- Port RTP local.

```
#include "RTPSession.h"
#include "RTPUDPLitev4Transmitter.h"
#include "RTPPacket.h"
#include "RTCPpacket.h"
#include "Wull.h"

void ShowUsage()
{
    printf("RTPWullSend @IP RemotePort LocalPort\r\n");
    printf("@IP : IPv4 remote address\r\n");
    printf("RemotePort : RTP remote port\r\n");
    printf("LocalPort : RTP local port\r\n");
}

int main(int argc, char* argv[])
{
    printf("RTPWullSend!\n");

    char mesData[1024];
    memset(mesData, 'A', sizeof(mesData));

    if (argc != 4)
    {
        ShowUsage();
        return -1;
    }

    // Initialisation de l'environnement des sockets
    int err;
    err = WULLStartup();

    CRTPSession RTPSession;
    CRTPUDPLitev4Transmitter* pRTPTransmitter = NULL;
    pRTPTransmitter = new CRTPUDPLitev4Transmitter();

    CRTPPacket* pRTPPacket = new CRTPPacket();

    // Création des connexions.
    pRTPTransmitter->SetRemoteIPAddress(argv[1]);
    pRTPTransmitter->SetRemotePort(atoi(argv[2]));
```

```
pRTPTransmitter->SetLocalPort(atoi(argv[3]));
pRTPTransmitter->OpenOutgoingConnection();
pRTPTransmitter->SetChecksumCoverage(8);

// Initialisation du contexte RTP.
RTPSession.SetTransmitter((CRTPTransmitter*)pRTPTransmitter);
RTPSession.SetPayloadType(96);
RTPSession.SetBandwidth(262144);
RTPSession.SetClockRate(90000);
RTPSession.SetIncrementTimeStamp(4500);

// Constitution de la payload du paquet RTP.
pRTPPacket->SetPayload(mesData, sizeof(mesData));

// Emission des paquets toutes les 2ms.
for (int i = 0; i < 50000; i++)
{
    RTPSession.SendRTPPacket(pRTPPacket);
    RTPSession.IncTimeStamp();
    Sleep(2);
}

if (pRTPPacket)
{
    delete pRTPPacket;
}

// Fermeture des connexions
pRTPTransmitter->CloseConnection();

printf("émission terminée");
_getch();
return 0;
}
```

L'environnement des sockets UDP-Lite est initialisé à l'aide de la fonction WULLWULLStartup.

L'émission des paquets RTP est réalisée en quatre étapes :

- Création des connexions,
- Initialisation du contexte RTP,
- Remplissage de la payload d'un paquet RTP,
- Emission des paquets RTP.

Lors de la création des connexions, l'utilisateur indique l'adresse, le port du destinataire ainsi que le port local utilisé dans les connexions. L'utilisateur crée ensuite le type de connexion RTP qu'il souhaite (entrante ou sortante). Dans le cas de UDP-Lite, l'utilisateur peut indiquer la taille de la partie sensible du paquet RTP.

L'initialisation du contexte RTP permet de révéler la nature des informations transportées par les paquets RTP et le pas d'incrémentatation entre les ADU. Dans l'exemple ci-dessus une ADU est contenue dans un paquet RTP.

Avant émission, l'utilisateur fournit la payload à envoyer dans chaque paquet RTP. Dans l'exemple ci-dessus le même paquet est transmis, l'étape de remplissage n'est pas renouvelée avant chaque émission.

Enfin le paquet RTP est émis grâce à la session RTP. La session RTP remplit les champs de l'en-tête RTP du paquet et émet le paquet.

### 3.2. Récepteur

Ce second exemple montre un programme qui réceptionne les paquets RTP. Ce programme est capable de s'adapter au protocole de transport, contrairement à l'exemple du programme émetteur. Le programme de réception prend quatre paramètres :

- adresse IPv4 du destinataire,
- port utilisé par le destinataire,
- port local,
- protocole de transport utilisé : « UDP » ou « UDP-Lite ».

```
#include "rtppacket.h"
#include "rtpsession.h"
#include "rtpudplitev4transmitter.h"
#include "rtpudpv4transmitter.h"
#include "Wull.h"

#define RTP_INCOMING 0
#define RTCP_INCOMING 1
#define NB_EVENTS 2

void ShowUsage()
{
    printf("RTPWullReceive @IP RemotePort LocalPort Protocol\r\n");
    printf("@IP : IPv4 remote address\r\n");
    printf("RemotePort : RTP remote port\r\n");
    printf("LocalPort : RTP local port\r\n");
    printf("Protocol : UDP or UDP-Lite\r\n");
}

int main(int argc, char* argv[])
{
    HANDLE hRTPEvent = 0;
    HANDLE hRTCPEvent = 0;
    HANDLE EventArray[NB_EVENTS];

    if (argc != 5)
    {
        ShowUsage();
        return -1;
    }

    // Initialisation de l'environnement des sockets
    int err;
    err = WULLStartup();

    bool bReceivingRTP = TRUE;
    CRTPPacket* pRTPPacket;
    CRTCPPacket* pRTCPEvent;

    printf("RTPWullReceive!\n");

    CRTPSession RTPSession;
```

```

CRTPTransmitter* pRTPTransmitter = NULL;
if (strcmp(argv[4], "UDP-Lite") == 0)
{
    pRTPTransmitter = new CRTPUDPLitev4Transmitter();
}
else
{
    if (strcmp(argv[4], "UDP") == 0)
    {
        pRTPTransmitter = new CRTPUDPv4Transmitter();
    }
    else
    {
        return -1;
    }
}

// Création des connexions entrantes
pRTPTransmitter->SetLocalPort(atoi(argv[3]));
pRTPTransmitter->SetRemoteIPAddress(argv[1]);
pRTPTransmitter->SetRemotePort(atoi(argv[2]));
pRTPTransmitter->OpenIncomingConnection();
RTPSession.SetTransmitter(pRTPTransmitter);

// Initialisation du cotexte RTP
RTPSession.SetBandwidth(262144);
RTPSession.SetClockRate(90000);

// Récupération des événements associé aux arrivées de paquets
hRTPEvent = RTPSession.GetIncomingRTPEvent();
hRTCPEvent = RTPSession.GetIncomingRTCPEvent();
EventArray[RTP_INCOMING] = hRTPEvent;
EventArray[RTCP_INCOMING] = hRTCPEvent;

int nNbOfReceivedPacket = 0;

DWORD dwWaitResult;
while (bReceivingRTP)
{
    dwWaitResult = WSAWaitForMultipleEvents(NB_EVENTS, EventArray,
FALSE, 5000, FALSE);

    if (dwWaitResult == WAIT_TIMEOUT)
    {
        bReceivingRTP = false;
    }
    else
    {
        int nIndex = dwWaitResult - WAIT_OBJECT_0;
        switch(dwWaitResult)
        {
            case RTP_INCOMING :
                // Paquet RTP entrant

                pRTPPacket = RTPSession.ReceiveRTPPacket();
                delete pRTPPacket;
                nNbOfReceivedPacket++;
                break;

            case RTCP_INCOMING :
                // incoming RTCP

```

```

        pRTCPPacket = RTPSession.ReceiveRTCPPacket();
        delete pRTCPPacket;
        break;
    }
}

printf("fin d'emission, %d paquets recus\n\r", nNbOfReceivedPacket);
_getch();
return 0;
}

```

Comme une émission, la réception des paquets RTP se déroule en quatre étapes :

- Création des connexions RTP et RTCP,
- Initialisation du contexte RTP,
- Récupération des évènements de paquets arrivant,
- Réception des paquets.

## 4. Architecture Générale

---

La librairie « bRTP » s'articule autour de quatre classes fondamentales :

- CRTPSession : objet qui gère le contexte RTP,
- CRTPTTransmitter : objet qui gère les connexions réseau pour RTP et RTCP,
- CRTPPacket : objet qui permet d'accéder aux champs d'un paquet RTP,
- CRTCPPacket : objet qui permet d'accéder aux champs d'un paquet RTCP.

Chaque utilisateur de la librairie « bRTP » devra allouer un objet CRTPSession et un objet CRTPTTransmitter.

Le type de RTPTransmitter dépend des protocoles de transport (UDP ou UDP-Lite) et réseau (IPv4 et IPv6) utilisés.

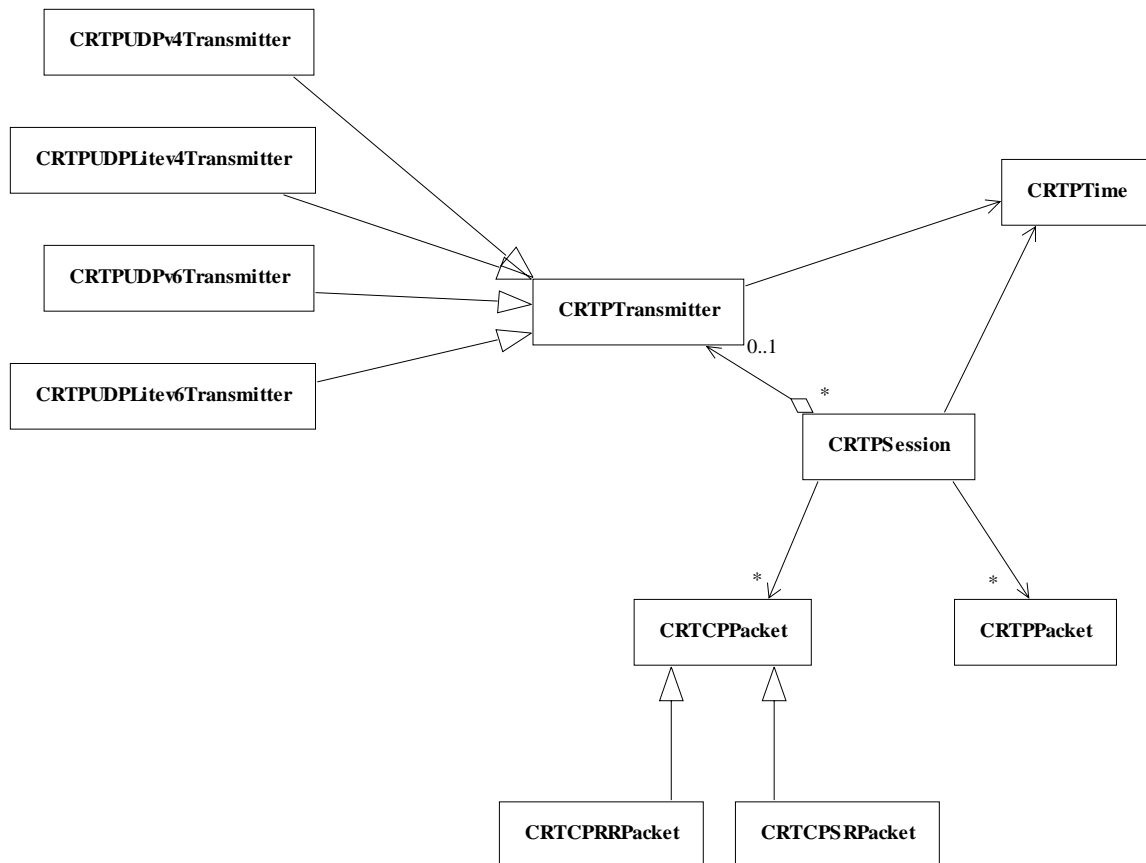


Figure 4 : Diagramme de classe de la librairie « bRTP »

## 5. Description des objets

---

### 5.1. CRTPSession

#### 5.1.1. Description

L'objet CRTPSession est un objet haut niveau qui fait le lien entre une application et le monde RTP. L'objet CRTPSession possède quatre rôles :

- Maintenir le contexte RTP ( SSRC, octets émis, nombre de paquets RTP non reçus, ...):
  - Initialisation du contexte RTP,
  - Mise à jour du contexte RTP,
- Remplir les en-tête des paquets RTP / RTCP,
- Créer et émettre les paquets RTCP lorsque l'intervalle d'émission des paquets RTCP est écoulé,
- Calculer en fonction de la nature des paquets reçus les informations de QoS :
  - Gigue,
  - RTT,
  - Taux de paquets RTP perdus.

#### 5.1.2. Méthodes

```
CRTPSession();
```

##### Description

Initialisation du contexte RTP.

```
~CRTPSession();
```

##### Description

Destruction du contexte RTP. L'objet CRTPTransmitter associé est détruit dans cette méthode.

```
void SetTransmitter(  
    CRTPTransmitter* pRTPTransmitter  
);
```

##### Description

Méthode qui associe un objet CRTPTransmitter au contexte RTP.

##### Paramètre

*pRTPTransmitter*

[in] Objet permettant d'émettre ou de recevoir des paquets RTP / RTCP.

```
void SetPayloadType(  
    char cPayloadType  
);
```

### Description

Initialisation de la nature du flux transporté par les paquets RTP (cf. [RFC3551] ).

### Paramètre

*cPayloadType*

[in] type du flux émis

```
void SetBandwidth(  
    unsigned long ulBandwidth  
);
```

### Description

Méthode qui fournit la bande passante (o/s) dédiée aux échanges RTP / RTCP. L'intervalle d'émission des paquets RTCP est calculé à partir de cette information. Tout appel à la méthode SetBandwidth en cours d'exécution réévalue l'intervalle d'émission des paquets RTCP.

### Paramètre

*ulBandwidth*

[in] Bande passante dédiée aux échanges RTP / RTCP en o/s

```
void SetClockRate(  
    unsigned long ulClockRate  
);
```

### Description

Initialisation la fréquence d'horodatage du flux multimedia transporté par les paquets RTP. Cette information est nécessaire pour remplir les champs des paquets RTCP.

### Paramètre

*ulClockRate*

[in] Fréquence d'horodatage.

```
void SetIncrementTimeStamp(  

```

```
unsigned long ulIncrementTimeStamp  
);
```

### Description

Initialisation du pas d'incrément du timestamp.

### Paramètre

*ulIncrementTimeStamp*

[in] Pas d'incrément du timestamp.

```
unsigned long IncTimeStamp();
```

### Description

Incrémente le timestamp du contexte RTP.

### Valeur retournée

timestamp après incrémentation.

```
int SendRTPPacket(  
    CRTPPacket* pRTPPacket  
);
```

### Description

Méthode typiquement utilisée par une application serveur. Elle remplit les champs de l'en-tête RTP du paquet et l'émet sur le réseau par l'intermédiaire de l'objet CRTPPTransmitter. Si l'intervalle d'émission des paquets RTCP est écoulé un paquet RTCP SR est créé et émis.

### Paramètre

*pRTPPacket*

[in] Pointeur sur un objet CRTPPacket à émettre.

### Valeur retournée

La méthode retourne le nombre d'octets émis (en-tête RTP + payload RTP).

```
CRTPPacket* ReceiveRTPPacket();
```

### Description

Méthode typiquement utilisée par une application cliente, elle permet de récupérer le paquet RTP reçu. Cette méthode met à jour le contexte RTP et si l'intervalle d'émission des paquets RTCP est écoulé, elle crée un paquet RTCP RR et l'émet.

### Valeur retournée

Pointeur vers l'objet CRTPPacket reçu.

```
CRTCPSocket* ReceiveRTPPacket();
```

### Description

Méthode qui permet de récupérer le paquet RTP reçu. Cette méthode met à jour le contexte RTP.

### Valeur retournée

Pointeur vers l'objet CRTCPSocket reçu.

```
HANDLE GetIncomingRTPEvent();
```

### Description

La méthode GetIncomingRTPEvent permet de récupérer l'évènement déclenché lors de l'arrivée de paquets RTP.

### Valeur retournée

HANDLE de l'évènement.

```
HANDLE GetIncomingRTCPEvent();
```

### Description

La méthode GetIncomingRTCPEvent permet de récupérer l'évènement déclenché lors de l'arrivée de paquets RTCP.

### Valeur retournée

HANDLE de l'évènement.

```
HANDLE GetNextRTPTime();
```

### Description

Cette Méthode retourne l'instant d'émission en microseconde du prochain paquet RTCP.

### Valeur retournée

Instant en microseconde.

```
unsigned long GetJitter();
```

### Description

Méthode qui retourne la gigue d'inter arrivée. La gigue est recalculée à chaque réception de paquets RTP. Si l'application qui utilise « bRTP » ne reçoit pas de paquets RTP, la gigue est à 0.

### Valeur retournée

Gigue d'inter arrivée.

```
unsigned long GetRTT();
```

### Description

Méthode qui retourne le RTT (Round Trip Time) en milliseconde d'un paquet RTCP.

### Valeur retournée

Durée du RTT en millisecondes.

```
unsigned short GetLocalPort();
```

### Description

Cette Méthode retourne le numéro du port local de la connexion RTP associé à la session RTP.

### Valeur retournée

Numéro du port RTP.

## 5.2. CRTPTransmitter

### 5.2.1. Description

CRTPTransmitter est l'objet en charge d'émettre et de recevoir les paquets RTP / RTCP sur les sockets. L'objet CRTPTransmitter doit s'adapter au protocole de transport utilisé dans les connexions RTP / RTCP. Pour s'adapter l'objet CRTPTransmitter est un objet abstrait qui formalise les fonctionnalités réseaux nécessaires à une session RTP. Les objets qui héritent de CRTPTransmitter s'adaptent aux particularités du protocole de transport.

### 5.2.2. Méthodes

```
CRTPTransmitter();
```

### Description

Initialisation.

```
~CRTPTransmitter();
```

### Description

Destruction.

```
char OpenOutgoingConnection();
```

### Description

Méthode qui ouvre une connexion RTP sortante (pour une application serveur par exemple). La connexion RTCP est entrante et sortante.

Cette méthode est spécifique au protocole de transport et est redéfinie dans chaque objet héritant de CRTPTTransmitter.

### Valeur retournée

Constante OK si Toutes les connexions sont bien ouvertes, constante NOT\_OK dans le cas contraire. Les constantes OK et NOT\_OK sont définies dans le fichier RTPCommon.h.

```
char OpenIncomingConnection();
```

### Description

Méthode qui ouvre une connexion RTP entrante (pour une application Player par exemple). La connexion RTCP est entrante et sortante.

Cette méthode est spécifique au protocole de transport et est redéfinie dans chaque objet héritant de CRTPTTransmitter.

### Valeur retournée

Constante OK si Toutes les connexions sont bien ouvertes, constante NOT\_OK dans le cas contraire. Les constantes OK et NOT\_OK sont définies dans le fichier RTPCommon.h.

```
int sendRTP(  
    char* pBuffer,  
    int nBufferSize,  
    LONGLONG* pSendTime  
);
```

### Description

Méthode qui envoie un paquet RTP.

Cette méthode est spécifique au protocole de transport et est redéfinie dans chaque objet héritant de CRTPTTransmitter.

### Paramètres

*pBuffer*

[in] Buffer contenant tout le paquet RTP (en-tête + payload).

*nBufferSize*

[in] Taille du buffer à envoyer.

*pSendTime*

[out] Instant d'émission en microseconde

### Valeur retournée

Nombre d'octets envoyés.

```
int sendRTCP(  
    char* pBuffer,  
    int nBufferSize,  
);
```

### Description

Méthode qui envoie un paquet RTCP.

Cette méthode est spécifique au protocole de transport et est redéfinie dans chaque objet héritant de CRTPTTransmitter.

### Paramètres

*pBuffer*

[in] Buffer contenant tout le paquet RTCP.

*nBufferSize*

[in] Taille du buffer à envoyer.

### Valeur retournée

Nombre d'octets envoyés.

```
int ReceiveRTP(  
    char* pBuffer,  
    int nBufferSize,  
    LONGLONG* pSendTime  
);
```

### Description

Méthode qui récupère le paquet RTP.

Cette méthode est spécifique au protocole de transport et est redéfinie dans chaque objet héritant de CRTPTTransmitter.

### Paramètres

*pBuffer*

[in] Buffer contenant le paquet RTP (en-tête + payload).

*nBufferSize*

[in] Taille du buffer de réception.

*pSendTime*

[out] Instant de récupération du paquet RTP.

### **Valeur retournée**

Nombre d'octets reçus.

```
int ReceiveRTCP(  
    char* pBuffer,  
    int nBufferSize,  
    LONGLONG* pSendTime  
);
```

### **Description**

Méthode qui récupère le paquet RTP.

Cette méthode est spécifique au protocole de transport et est redéfinie dans chaque objet héritant de CRTPTTransmitter.

### **Paramètres**

*pBuffer*

[in] Buffer contenant le paquet RTP (en-tête + payload).

*nBufferSize*

[in] Taille du buffer de réception.

*pSendTime*

[out] Instant de récupération du paquet RTP.

### **Valeur retournée**

Nombre d'octets reçus.

```
void CloseConnection();
```

### **Description**

Méthode qui ferme les connexions RTP et RTCP.

```
HANDLE GetIncomingRTPEvent();
```

### **Description**

La méthode GetIncomingRTPEvent permet de récupérer l'évènement déclenché lors de l'arrivée de paquets RTP.

### **Valeur retournée**

HANDLE de l'évènement.

```
void SetLocalPort(  
    unsigned short usLocalPort  
);
```

### Description

Initialise le port RTP local.

### Paramètre

*usLocalPort*

[in] Numéro du port RTP local.

```
unsigned short GetLocalPort();
```

### Description

Retourne le numéro du port RTP local.

### Valeur retournée

Numéro du port RTP local.

```
void SetRemotePort(  
    unsigned short usRemotePort  
);
```

### Description

Initialise le port RTP du correspondant (player ou serveur).

### Paramètre

*usRemotePort*

[in] Numéro du port RTP distant.

```
void SetRemoteIPAddress(  
    char* szIPAddress  
);
```

### Description

Initialise l'adresse IP du correspondant (player ou serveur).

### Paramètre

*szRemoteIPAddress*

[in] Adresse du correspondant (0.0.0.0 pour IPv4 ou 0:0:0:0:0:0:0:0 pour IPv6)

```
bool IsSender();
```

### Description

Détermine si l'utilisateur de « bRTP » est un émetteur « RTP ».

### Valeur retournée

Return true si l'utilisateur peut émettre des paquets RTP, false sinon.

### 5.2.3. Héritage

Quatre objets héritent de CRTPTransmitter :

- CRTPUDPv4Transmitter : pour utiliser le protocole UDP dans IPv4,
- CRTPUDPLitev4Transmitter : pour utiliser le protocole UDP-Lite dans IPv4,
- CRTPUDPv6Transmitter : pour utiliser le protocole UDP dans IPv6,
- CRTPUDPLitev6Transmitter : pour utiliser le protocole UDP-Lite dans IPv6.

Toute implémentation d'un nouveau protocole de transport doit hériter de l'objet CRTPTransmitter.

## 5.3. CRTPPacket

### 5.3.1. Description

CRTPPacket est un objet qui permet d'accéder facilement aux champs d'un en-tête RTP.

### 5.3.2. Méthodes

```
CRTPPacket();
```

### Description

Initialisation.

```
~CRTPPacket();
```

### Description

Destruction.

```
void SetPayloadType(  
char PT  
);
```

### Description

Renseigne le champ PT de l'en-tête RTP.

### Paramètre

*PT*

[in] Type de la payload transporté dans le paquet RTP (cf. [RFC3551]).

```
void SetTimeStamp(  
unsigned long ulTS  
);
```

### Description

Renseigne le champ Timestamp de l'en-tête RTP.

### Paramètre

*ulTS*

[in] Timestamp du paquet RTP.

```
void SetSequenceNumber(  
unsigned short usSequenceNumber  
);
```

### Description

Renseigne le champ SequenceNumber de l'en-tête RTP.

### Paramètre

*usSequenceNumber*

[in] Numéro de séquence du paquet RTP.

```
void SetSSRC(  
    unsigned long ulSSRC  
);
```

### Description

Renseigne le champ SequenceNumber de l'en-tête RTP.

### Paramètre

*ulSSRC*

[in] Identifiant SSRC de l'émetteur du paquet RTP.

```
void SetPacketSize(  
    int nPacketSize  
);
```

### Description

Méthode qui renseigne la taille du paquet RTP (en-tête RTP + payload RTP).

### Paramètre

*nPacketSize*

[in] Taille du paquet RTP.

```
int GetPacketSize();
```

### Description

Méthode qui retourne la taille du paquet RTP (en-tête + payload).

### Valeur retournée

Taille du paquet RTP.

```
int GetMaxPacketSize();
```

### Description

Méthode qui retourne la taille maximale que peut avoir un paquet RTP.

### Valeur retournée

Taille maximale d'un paquet RTP.

```
int SetPayload(  
    const char*    pData,
```

```
unsigned short usDataLength  
);
```

### Description

Méthode qui copie les données dans la payload du paquet RTP.

### Paramètres

*pData*

[in] buffer des données à copier.

*usDataLength*

[in] taille des données à copier.

### Valeur retournée

Taille des données copiées.

```
Char* GetPayload();
```

### Description

Méthode qui retourne un pointeur sur le début de la payload du paquet RTP.

### Valeur retournée

Pointeur sur le début de la payload RTP.

```
char* GetRTPBuffer();
```

### Description

Méthode qui retourne un pointeur sur le début du paquet RTP (en-tête RTP).

### Valeur retournée

Pointeur sur le début du paquet RTP.

```
char GetPayloadType();
```

### Description

Retourne le type de la payload contenu dans le paquet RTP.

### Valeur retournée

Numéro du type de la payload RTP (cf. [RFC3551]).

```
Unsigned long GetTimeStamp();
```

### Description

Méthode qui retourne le timestamp associé au paquet RTP.

### **Valeur retournée**

Timestamp contenu dans le paquet RTP.

```
unsigned long GetSSRC();
```

### **Description**

Méthode qui retourne le SSRC du paquet RTP.

### **Valeur retournée**

Valeur du SSRC.

```
unsigned short GetSequenceNumber();
```

### **Description**

Méthode qui retourne le numéro de séquence du paquet RTP.

### **Valeur retournée**

Valeur du Sequence number.

## 5.4. CRTCPPacket

### 5.4.1. Description

CRTCPPacket est un objet qui permet d'accéder aux champs communs des en-têtes RTCP RR et RTCP SR.

### 5.4.2. Méthodes

```
CRTCPPacket();
```

### **Description**

Initialisation.

```
~CRTCPPacket();
```

### **Description**

Destruction.

```
void SetPayloadType(  
char nPayloadType  
);
```

### Description

Renseigne le champ PT de l'en-tête RTCP.

### Paramètre

*nPayloadType*

[in] Type de la payload transporté dans le paquet RTCP (cf. [RFC3550]).

```
void SetArrivalTime(  
__int64 llArrivalTime  
);
```

### Description

Méthode qui permet d'assigner le temps d'arriver d'un paquet RTCP.

### Paramètre

*llArrivalTime*

[in] Instant d'arrivée du paquet RTCP en micro seconde.

```
void SetLength(  
unsigned short usLength  
);
```

### Description

Renseigne le champ Length de l'en-tête RTCP.

### Paramètre

*usLength*

[in] Taille de l'en-tête RTCP, cette information permet de valider un paquet RTCP à sa réception.

```
void SetPacketSize(  
int nPacketSize  
);
```

### Description

Méthode qui renseigne la taille du paquet RTCP.

**Paramètre**

*nPacketSize*

[in] Taille du paquet RTCP.

```
int GetPacketSize();
```

**Description**

Méthode qui retourne la taille du paquet RTP (en-tête + payload).

**Valeur retournée**

Taille du paquet RTP.

```
int GetMaxPacketSize();
```

**Description**

Méthode qui retourne la taille maximale que peut avoir un paquet RTP.

**Valeur retournée**

Taille maximale d'un paquet RTP.

```
char* GetRTCPBuffer();
```

**Description**

Méthode qui retourne un pointeur sur le début du paquet RTCP.

**Valeur retournée**

Pointeur sur le début du paquet RTCP.

```
char GetPayloadType();
```

**Description**

Retourne le type de la payload contenu dans le paquet RTP.

**Valeur retournée**

Numéro du type de la payload RTP (cf. [RFC3551]).

```
char GetVersion();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « Version » du paquet RTCP.

### Valeur retournée

Valeur du champ « Version » du paquet RTCP.

```
char GetPadding();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « P » du paquet RTCP.

### Valeur retournée

Valeur du champ « P » du paquet RTCP.

```
char GetRC();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « RC » du paquet RTCP.

### Valeur retournée

Valeur du champ « RC » du paquet RTCP.

```
Unsigned short GetLength();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « Length » du paquet RTCP.

### Valeur retournée

Valeur du champ « Length » du paquet RTCP.

## 5.5. CRTCP SRPacket

### 5.5.1. Description

CRTCP SRPacket est un objet qui permet d'accéder facilement aux champs spécifiques des paquets RTCP SR.

### 5.5.2. Méthodes

```
CRTCP SRPacket();
```

### Description

Initialisation.

```
~CRTCP SRPacket();
```

### Description

Destruction.

```
void SetSSRC(  
unsigned long ulSSRC  
);
```

### Description

Méthode qui renseigne le champ « SSRC » de l'en-tête RTCP SR.

### Paramètre

*ulSSRC*

[in] Valeur du SSRC.

```
void SetNTPTimestampMSW(  
unsigned long nNTPTimestampMSW  
);
```

### Description

Méthode qui renseigne le champ « MSW » de l'en-tête RTCP SR.

### Paramètre

*nNTPTimestampMSW*

[in] Valeur du MSW.

```
void SetNTPTimestampLSW(  
unsigned long nNTPTimestampLSW  
);
```

### Description

Méthode qui renseigne le champ « LSW » de l'en-tête RTCP SR.

### Paramètre

*nNTPTimestampLSW*

[in] Valeur du LSW.

```
void SetRTPTimestamp(  
    unsigned long ulRTPTimestamp  
);
```

### Description

Méthode qui renseigne le champ « RTPTimestamp » de l'en-tête RTCP SR.

### Paramètre

*ulRTPTimestamp*

[in] Valeur du RTPTimestamp.

```
void SetRTPOctetCount(  
    unsigned long nByteCount  
);
```

### Description

Méthode qui renseigne le champ « sender's octet count » de l'en-tête RTCP SR.

### Paramètre

*nByteCount*

[in] Valeur du « sender's octet count ».

```
void SetRTPPacketsCount(  
    unsigned long nPacketsCount  
);
```

### Description

Méthode qui renseigne le champ « sender's packet count » de l'en-tête RTCP SR.

### Paramètre

*nPacketsCount*

[in] Valeur du « sender's octet count ».

```
unsigned long GetSSRC();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « SSRC » du paquet RTCP SR.

### Valeur retournée

Valeur du « SSRC ».

```
unsigned long GetNTPTimeStampMsw();
```

### **Description**

Méthode qui retourne la valeur contenue dans le champ « NTP timestamp, MSW » du paquet RTCP.

### **Valeur retournée**

Valeur du « NTP timestamp, MSW ».

```
unsigned long GetNTPTimeStampLsw();
```

### **Description**

Méthode qui retourne la valeur contenue dans le champ «NTP timestamp, LSW » du paquet RTCP.

### **Valeur retournée**

Valeur du « NTP timestamp, LSW ».

```
unsigned long GetRTPTimeStamp();
```

### **Description**

Méthode qui retourne la valeur contenue dans le champ «RTPTimestamp » du paquet RTCP.

### **Valeur retournée**

Valeur du « RTPTimestamp ».

```
unsigned long GetRTPOctetCount();
```

### **Description**

Méthode qui retourne la valeur contenue dans le champ « sender's octet count » du paquet RTCP.

### **Valeur retournée**

Valeur du « sender's octet count ».

```
unsigned long GetRTPPacketCount();
```

### **Description**

Méthode qui retourne la valeur contenue dans le champ « sender's packet count » du paquet RTCP.

### **Valeur retournée**

Valeur du « sender's packet count ».

## 5.6. CRTCPRRPacket

### 5.6.1. Description

CRTCPRRPacket est un objet qui permet d'accéder facilement aux champs spécifiques des paquets RTCP RR.

### 5.6.2. Méthodes

```
CRTCPRRPacket();
```

#### Description

Initialisation.

```
~CRTCPRRPacket();
```

#### Description

Destruction.

```
void SetReportCount(  
char nReportCount  
);
```

#### Description

Méthode qui renseigne le champ « RC » de l'en-tête RTCP RR.

#### Paramètre

*nReportCount*

[in] Valeur du RC.

```
void SetSSRC (  
unsigned long ulSSRC  
);
```

#### Description

Méthode qui renseigne le champ « SSRC » de l'en-tête RTCP RR.

#### Paramètre

*nSSRC*

[in] Valeur du SSRC.

```
void SetRemoteSSRC (  
    unsigned long ulRemoteSSRC  
);
```

### Description

Méthode qui renseigne le champ « SSRC destinataire » de l'en-tête RTCP RR.

### Paramètre

*ulRemoteSSRC*

[in] Valeur du SSRC destinataire.

```
void SetFractionLost (  
    char nFractionLost  
);
```

### Description

Méthode qui renseigne le champ « fraction lost » de l'en-tête RTCP RR.

### Paramètre

*nFractionLost*

[in] Valeur du « Fraction lost ».

```
void SetPacketsLost (  
    unsigned long nPacketsLost  
);
```

### Description

Méthode qui renseigne le champ « cumulative packet lost » de l'en-tête RTCP RR.

### Paramètre

*nPacketsLost*

[in] Valeur du « cumulative packet lost ».

```
void SetExtendedSequenceNumber (  
    unsigned long nExtSeqNb  
);
```

### Description

Méthode qui renseigne le champ « extended highest sequence number received » de l'en-tête RTCP RR.

### Paramètre

*nExtSeqNb*

[in] Valeur du « extended highest sequence number received ».

```
void SetJitter (  
    unsigned long fJitter  
);
```

### Description

Méthode qui retourne la valeur contenue dans le champ « interarrival jitter » du paquet RTCP RR.

### Paramètre

*fJitter*

[in] Valeur du « interarrival jitter ».

```
void SetLSR (  
    unsigned long fLSR  
);
```

### Description

Méthode qui retourne la valeur contenue dans le champ « LSR » du paquet RTCP RR.

### Paramètre

*fLSR*

[in] Valeur du « LSR ».

```
void SetDLSR (  
    unsigned long fDLSR  
);
```

### Description

Méthode qui retourne la valeur contenue dans le champ « DLSR » du paquet RTCP RR.

### Paramètre

*fDLSR*

[in] Valeur du « DLSR ».

```
char GetSSRC ();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « SSRC » du paquet RTCP RR.

### Valeur retournée

Valeur du « SSRC ».

```
char GetRemoteSSRC ();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « SSRC\_1 » du paquet RTCP RR.

### Valeur retournée

Valeur du « SSRC\_1 ».

```
char GetFractionLost ();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « fraction lost » du paquet RTCP RR.

### Valeur retournée

Valeur du « fraction lost ».

```
char GetPacketLost ();
```

### Description

Méthode qui retourne la valeur contenue dans le champ « cumulative number of packets lost » du paquet RTCP RR.

### Valeur retournée

Valeur du « cumulative number of packets lost ».

```
char GetExtendedSequenceNumber ();
```

## Description

Méthode qui retourne la valeur contenue dans le champ « extended highest sequence number received » du paquet RTCP RR.

## Valeur retournée

Valeur du « extended highest sequence number received ».

```
unsigned long GetJitter ();
```

## Description

Méthode qui retourne la valeur contenue dans le champ « interarrival jitter » du paquet RTCP.

## Valeur retournée

Valeur du « interarrival jitter ».

```
unsigned long GetLSR ();
```

## Description

Méthode qui retourne la valeur contenue dans le champ « LSR » du paquet RTCP.

## Valeur retournée

Valeur du « LSR ».

```
unsigned long GetDLSR ();
```

## Description

Méthode qui retourne la valeur contenue dans le champ « DLSR » du paquet RTCP.

## Valeur retournée

Valeur du « DLSR ».

## 5.7. CRTPTime

### 5.7.1. Description

Objet qui permet de récupérer l'instant en microsecondes.

### 5.7.2. Méthodes

```
static LONGLONG GetDLSR ();
```

## Description

Méthode qui retourne l'instant en microsecondes.

**Valeur retournée**

Valeur du « DLSR ».

## 6. Utiliser bRTP dans son projet

---

### 6.1. Fichiers nécessaires

L'utilisation de « bRTP » nécessite :

- d'accéder aux « .h » qui décrivent les objets manipulés,
- de lier au projet la librairie statique « bRTP.lib ».

### 6.2. Contraintes d'utilisation

Le protocole IPv6 doit être installé sur les postes utilisant « bRTP ».

Pour utiliser « bRTP », plusieurs logiciels sont nécessaires :

- Microsoft SDK qui peut être téléchargé à l'adresse : <http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en#RelatedDownloads>
- Wull pour gérer les sockets UDP-Lite sur IPv4,
- Wull6 pour gérer les sockets UDP-Lite sur IPv6.

### 6.3. Fonctionnalités non implémentées

Tous les mécanismes du [RFC3550] ne sont pas implémentés. Les fonctionnalités non implémentées sont :

- le multicast,
- la gestion dynamique des membres,
- certains paquets RTCP :
  - SDES,
  - BY,
  - APP,
- les mixer et les translater,
- la gestion des collisions de paquets,
- le cryptage.