Project	Publising Game Management System
Team	기술본부/게임시스템팀
Document Title	PMSConn 사용 방법
Document Class	Publising Game Management System

Revision	0.5	
Author	계동원 (truewise@neowiz.com) 서한호 (answer@neowiz.com)	
Date	2009/01/02	
Revision History	0.1	First Release
	0.2	연동에 대한 설명 추가
	0.3	PC 방 통계 추가 (section 3)
	0.4	채널링 정보 추가 (section 4)
	0.5	유료 PC방 통계 추가 (section 5)

1 Introduction

1.1 PMS는 게임 서버로부터 통계 정보를 받고, 게임 서버를 제어하기 쉽게 종합적으로 관리하고 모니터링 하기 위한 네오위즈의 시스템 플렛폼이다.

1.2 PMS 시스템은 중앙에서 데이터를 관리하고 각 호스트를 제어하는 MA(메인 서버)와 게임 서버와 MA 의 연동을 돕기 위해 각 호스트에 설치되는 HA, 그리고 운영자나 시스템 엔지니어가 게임 서버 상황을 모니터링 하고 제어하거나 공지 메시지를 보내기 위해 사용하는 MC, 이렇게 3가지로 구성된다.

1.3 개발사에서 제작한 게임 서버는 시스템에 설치되면서 MA 에 각종 데이터를 보내줘야 하는데, 이 때 게임 서버와 직접 통신을 하는 서비스 데몬이 HA 이다. HA 는 ADL 이라는 네오위즈 자체 프로토콜과 TEXT 기반의 라인 프로토콜 두가지 방식으로 연동이 가능한데, ADL 기반의 프로토콜이 더 안정적이고 많은 기능을 제공한다. ADL 방식이 연동하기 난해한 부분이 있어서 본사에서는 PMSConn 이라는 미들웨어를 제공한다. PMSConn 은 ADL 통신 프로토콜의 Wrapper 라이브러리이다.

1.4 PMSConn 을 이용해 처음 시작 시 초기화를 하면, 하트 비트 요청에 대한 응답이 자동으로 처리되고, 퍼포먼스 정보 요청에 대한 응답, 통계 정보 요청에 대한 응답 등에 대해서 응답하면 된다. 경우에 따라서는 게임 서버에서 자체 진단한 오류 메시지를 전달 할 수도 있다.

1.5 PMSConn 의 사용의 편의를 돕기 위해 VirtualPMS 와 PMSConnSample 이라는 테스트 프로그램이 제공된다.

1.6 문의 사항은 truewise@neowiz.com 또는 answer@neowiz.com 으로…

2 기본 설명

2.1 수신 메시지 (IPMSObject.h)

```
switch (msg.mTagID)
{
case PayloadGS::msgPMSAnnounceReq_Tag:
        // 공지 메시지가 요청된 경우 (MC->MA->HA->GS)
        // 수신된 공지 메시지를 처리한다.
        // 리턴 값에 의해 HA 에 수신 했다는 응답 성공/실패 결정
        OnAnnounceReq(msg.un.m_msgPMSAnnounceReq);
        break;
case PayloadGS::msgPMSHeartBeatReq_Tag:
        // 하트 비트 메시지 요청 (MA->HA->GS)
        // 리턴값에 의해 하트 비트 에러 처리
        // 5회 이상 연속으로 응답 하지 않으면 장애로 판단한다.
        OnHeartBeatReq(msg.un.m_msgPMSHeartBeatReq);
        break;
case PayloadGS::msgPMSOrderReq_Tag:
        // 특별히 정의된 메시지를 전달할 때 사용하는 메시지.
        // 별도로 필요한 협의가 있을 경우 이 메시지를 응용해서 사용할 수 있다.
        // 일반적으로는 사용하지 않는다.
        OnOrderReq(msg.un.m_msgPMSOrderReq);
        break;
case PayloadGS::msgPMSPerformReq_Tag:
        // 게임서버의 퍼포먼스 정보 (혹은 기타 약속된 정보)에 대한 요청
        OnPerformReq(msg.un.m_msgPMSPerformReq);
        break;
case PayloadGS::msgPMSRegionInfoReq_Tag:
        // 지역별 인원 통계에 대한 요청
        OnRegionInfoReq(msg.un.m_msgPMSRegionInfoReq);
        break;
case PayloadGS::msgPMSStatInfoReq_Tag:
        // CU(접속자 통계), 룸, 채널 정보 등 상태 정보에 대한 요청
        OnStatInfoReq(msg.un.m_msgPMSStatInfoReq);
        break;
}
```

2.2 **virtual BOOL OnAnnounceReq(DWORD dwSSN, DWORD dwCategoryID, LPCSTR lpszMsg)**

공지 메시지가 lpszMSG 에 담겨서 전달된다.

이 이벤트가 발생하면 게임 서버에서 공지 메시지를 처리하면 된다.

경우에 따라, CategoryID 에 해당되는 정보가 있으면 해당 타겟에만 처리한다.

리턴값에 의해 공지 메시지 전달(혹은 처리)가 성공했는지 여부를 HA/MA로 응답한다.

2.3 **virtual BOOL OnHeartbeatReq(LONG lIndex)**

특별히 처리해야 할 사항이 없다면 TRUE 를 리턴하면 된다.

lIndex 는 하트비트 인덱스 값을 의미한다.

5번 연속 요청에 대해 응답하지 않으면 장애로 간주한다.

2.4 **virtual BOOL OnRegionInfoReq(IPMSReionInfoList* plstRegionInfo)**

지역(+연령/성)별 통계 데이터 요청 이벤트이다.

```
VLONG vLong;
plstRegionInfo->AddRegionInfo(SSN, category, &vLong)  //vLong.size() = 238
```

vLong 백터에 [지역:17][연령:7][성별:2] 에 해당되는 데이터를 넣어서 SSN (게임 번호), category 등과 함께 HA 로 응답을 보낸다.

2.5 **virtual BOOL OnStatInfoReq(IPMSStatInfoList* plstStatInfo)**

상태 정보 요청 이벤트이다.

```
plstStatInfo->AddStatInfoList(SSN,category,CU,session,channelcnt,roomcnt,NULL)
```

총 접속자(CU), 세션(session), 채널, 룸 등의 정보를 보낸다.

총 접속자 수는 2.4 의 RegionInfo에 넣은 연령별, 성별, 지역별 CU 의 합계이다.
총 접속자 수는 Launcher 로부터 넘겨 받은 값을 계산해서 연령별, 성별, 지역별 통계와 오차가 생기지 않도록 처리해야 한다.

2.6 **virtual BOOL OnPerformInfoReq(IPMSPerformanceInfo *pPerformanceInfo)**

퍼포먼스 정보 요청 이벤트이다.

실제 게임 서버가 사용하고 있는 CPU 양이나 메모리는 HA 에서 확인한다.
게임 서버가 직접 조사해서 보내는 퍼포먼스 정보는 네오위즈와 협의를 거쳐 결정한다.

```
VLONG vlong;
pPerformanceInfo->AddPerformanceInfo(&vlong);
```

보통의 경우 사용할 일이 없으며, vLong 백터에 0 을 체워서 보내주면 된다.

2.7 **virtual BOOL OnOrderReq(LPCSTR lpszCmdName, LPCSTR lpszCtlVal, LPSTR lpResult, LONG *lpResultLen, DWORD dwSSN, DWORD dwCategoryID)**

운영툴에서 특별히 게임 서버로 명령을 내릴 때 사용한다. 이 때 사용하는 메시지 내용은 사전에 게임 개발사와 네오위즈간에 협의 된 것이어야 하며, 일반적으로 사용하지 않는다. 정해진 것이 없다면 true 를 리턴해주면 된다.

2.8 PMSConn 의 실제 내부 구조에 대해서는 Section 3 에서 확인하면 된다.

3 통계 관련 추가 정보

3.1 Vector 사용이 불가능 한 경우

virtual BOOL OnRegionInfoReq(IPMSReionInfoList* plstRegionInfo)

VLONG vLong;
plstRegionInfo->AddRegionInfo(SSN, category, &vLong) //vLong.size() = 238

RegionInfoReq 에 대한 응답으로 AddRegionInfo를 사용할 때 238개의 LONG 데이터가 들어있는 백터를 인자로 넣게 되어 있는데, Visual C++ 에서 제공하는 STL 외의 제품을 사용하거나 특별한 이유로 STL 을 사용할 수 없다면, Array 버전의 Method 를 사용 가능하다.

AddRegionInfo238(DWORD dwSSN, DWORD dwCategoryID, LONG pvRegionStat[238]) = 0;
AddRegionInfo272(DWORD dwSSN, DWORD dwCategoryID, LONG pvRegionStat[272]) = 0;

3.2 PC방 통계

PC 방 통계를 저장하는 방법은 일반 통계 저장하는 것과 완전히 동일하다.

AddRegionInfoPC(DWORD dwSSN, DWORD dwCategoryID, VLONG* pvRegionStat) = 0;
AddRegionInfoPC238(DWORD dwSSN, DWORD dwCategoryID, LONG pvRegionStat[238]) = 0;
AddRegionInfoPC272(DWORD dwSSN, DWORD dwCategoryID, LONG pvRegionStat[272]) = 0;
위의 Method 를 사용해서 인자로 데이터를 넣어주면 된다.

주의할 점은 PC 방 통계는 전체 통계 데이터와 별도라는 점이다.
전체 통계는 PC 방 통계를 포함한 총 통계 정보를 보내줘야 한다.

예를 들어,
총 접속자 수가 1000 명이고, 그 중 PC 방 접속자가 500명일 경우
AddRegionInfo 에는 1000명에 해당되는 데이터를,
AddRegionInfoPC 에는 500명에 해당되는 데이터를 보내야 한다.
(PC방 통계를 제외한 500명만 보내는 것이 아니다)

4 채널링 관련 추가 정보

4.1 채널링에 대한 데이터는 StatInfo 메시지를 이용한다.

4.1.1 AddStatInfoList(SSN,category,CU,session,channelcnt,roomcnt,NULL)

이 함수를 사용해서 데이터를 Add 할 때, category 항목에 채널링에서 사용하는 ID 값과 통계 정보를 넣으면 된다.

4.1.2 category 에는 기본적으로 게임 내의 채널에 대한 정보(채널ID값)를 넣게 되어 있는데, 채널링이 이 값을 공유하므로, 채널링 데이터를 넣을 때 총 통계값과는 별도로 계산해야 한다

예를 들어 SSN 이 500 번인 게임이 있을 때,
> 기본채널 0 (1000명)
> 초보채널 1 (2000명)
> 프로채널 2 (3000명)
> 세이채널링 1000 (4000명)
이라고 가정하면,

AddStatInfoList(500, 0, 1000, ?, ?, NULL);
AddStatInfoList(500, 1, 2000, ?, ?, NULL);
AddStatInfoList(500, 2, 3000, ?, ?, NULL);
AddStatInfoList(500, 1000, 4000, ?, ?, NULL);

이렇게 추가했을 때, 세이채널링이면서, 프로(또는 기본,초보)채널에 있는 사용자의 경우 2곳에서 CU가 계산 될 수 있으므로, 총 통계 수치는 채널링의 값을 제외한 6천명(기본채널+초보채널+프로채널)이 됩니다.

즉, 채널링 통계 데이터는 실제 채널 통계 데이터와 합쳐지지 않는 독립적으로 존재하는 값으로 생각하면 된다.

4.1.3 채널링에 사용되는 ID값은 담당 SE 에게 문의하면 확인 가능하다.

5 유료 PC 방 통계

5.1 유료 PC 방에 대한 데이터는 StatInfoPC 메시지를 이용한다.

5.1.1 AddStatInfoPC(SSN, category, CU, session, channelcnt, rootcnt, NULL)

이 함수를 사용해서 데이터를 Add 할 때, category 항목에 유료PC 방에서 사용하는 ID 값과 통계 정보를 넣으면 된다.

5.1.2 Category 에는 PC방 통계의 경우 기본적으로 0을 넣어주는데, 전체 PC 방 통계 데이터와 별도로 유료 PC 방 통계 데이터를 넣어주어야 한다.

예를 들어 SSN 이 500 번인 게임이 있을 때,
> 기본채널 0 (3000명)
> 정량제 10001 (1000명)
> 정액제 10002 (1000명)
이라고 가정하면,

AddStatInfoList(500, 0, 2000, ?, ?, NULL);
AddStatInfoList(500, 10001, 1000, ?, ?, NULL);
AddStatInfoList(500, 10002, 1000, ?, ?, NULL);

이렇게 추가하면 된다.

0의 경우, 다른 유료방 PC 정보가 없이도 데이터로 가치를 가져야 하기 때문에, 유료 PC방 통계에 상관없이 반드시 입력해 주어야 한다. (독립적으로 존재하는 값으로 보시면 된다)

5.1.3 정량제, 정액제 외의 추가 요금 제도가 생길 경우 SE 들과 협의 후 상용하면 된다.

6 Using PMSConn native API

6.1 필요 파일들

6.1.1 PMSConn. Dll , PMSConn.lib

1) ADL 프로토콜을 wrapping한 DLL 및 library 파일

6.1.2 PMSConn.h

1) PMSConn Native API가 정의된 헤더 파일

6.1.3 PMSCode.h

1) PMSConn 관련 각종 상태 및 에러 코드가 기술된 파일

6.1.4 PMSObject.h

1) PMSConn에서 이벤트가 발생하였을 때 이벤트를 처리 하기 위한 Call Back Object

6.2 PMS의 기본적인 상태 및 오류 코드

6.2.1 PMSCode.h 의 기술

```
enum PMSCONN_STATUSCODE
{
        PMSC_STATUS_INIT_BEFORE = 0, // PMS초기화전
        PMSC_STATUS_INIT, // PMS 초기화됨
        PMSC_STATUS_RUN,  // PMS 동작중
        PMSC_STATUS_STOP, // PMS 정지중
        PMSC_STATUS_NOUSE, // PMS 사용 하지 않는 MODE 설정됨
        PMSC_STATUS_TRY_CONNECTING, // PMS가 HA에 연결을 시도 하려고 하고 있음
};

enum PMSCONN_ERRORCODE
{
        PMSC_ERR_OK = 0, // 성공
        PMSC_ERR_NOT_INITIALIZED, // PMS가 아직 초기화 되지 않았음
        PMSC_ERR_INIT_INVALIDPARAM, // 잘못된 실행 인자
        PMSC_ERR_RUN_ALREADY,   // PMS가 이미 동작중임
        PMSC_ERR_RUN_THREAD_FAILED, // PMS ADL Message 처리를 위한 Thread 생성 실패
};

enum FAULT_LEVEL
{
        FL_WARNING = 1, // only saved DB
        FL_ALERT ,      // saved DB and MC notified as ALERT Level
        FL_CRITICAL , // saved DB and MC notified as Critical Level
```

```
        FL_MAX,
};
```

6.3 PMSConn Native API 설명 - PMSConn.h 에 있음

6.3.1 DWORD PMSInitConn(int argc, LPTSTR argv[])

1) 프로그램 시작 시 전달 받은 인자를 PMSInitConn에 그대로 넘겨 준다.

6.3.2 DWORD PMSRunConn(IPMSObject *pObj)

1) PMSConn을 구동 시키는 함수이다. IPMSObject는 PMSObject.h 참조

6.3.3 DWORD PMSStopConn()

1) PMSConn의 구동을 종료 시키는 함수 이다.

6.3.4 BOOL PMSSendWarningMsg(DWORD dwErrLvl, LPCSTR pszWarningMsg, LPCSTR pszTreatMsg, DWORD dwSSN, DWORD dwCategory);

1) PMS로 Game Server가 감지한 문제 및 중요 정보를 기록으로 남기기 위한 API 장애 레벨에 따라서 즉시 MC로 통보 된다.

6.3.5 LPCTSTRPMSGetConfigFileName();

1) PMS로 넘겨진 성능 정보 Config 파일의 이름을 가져 오는 함수 - 현재 사용되지 않음

6.3.6 DWORD PMSGetStatus()

1) PMSConn의 현재 상태를 가져 오기 위한 함수

6.3.7 PMSConn.h 파일의 기술

```
/*
PMSInitConn : PMSConn 초기화를 하기 위한 함수, Console 모드일때는 main함수의 인자를 ,
Service Mode일때는 ServiceMain 함수의 인자를 전달
@param1 : 실행 인자 개수
@param2 : 실행인자의 포인터 배열
@return value : PMSCONN_ERRORCODE 참조
*/
extern PMSCONN_API      DWORD   PMSInitConn(DWORD argc, LPTSTR argv[]);
/*
PMSRunConn : PMSConn을 실제적으로 동작 시키는 함수
@param1 : Call Back을위한 IPMSObject 포인터 IPMSObject는 IPMSObject관련 설명 참조
@return value : PMSCONN_ERRORCODE 참조
*/
extern PMSCONN_API  DWORD       PMSRunConn(IPMSObject *pObj);
/*
PMSStopConn : PMSConn을 종료 하기 위한 함수
*/
extern PMSCONN_API  void        PMSStopConn();
/*
```

```
PMSSendWarningMsg : PMS로 Warning 메시지를 보내기 위한 함수
@param1 : 장애 레벨 FAULT_LEVEL 참조
@param2 : 장애 내용
@param3 : 장애 처리 방법
@param4 : 장애 발생 게임 Serial
@param5 : 장애 발생 카테고리
*/
extern PMSCONN_API      BOOL    PMSSendWarningMsg(DWORD dwErrLvl, LPCSTR pszWarningMsg,
LPCSTR pszTreatMsg, DWORD dwSSN, DWORD dwCategory);

/*
PMSGetConfigFileName PMS로 넘겨진 성능 정보 Config 파일의 이름을 가져 오는 함수
@return value PMS인자중 /CNFGFILE:xxx.xxx 중 xxx.xxx값을 리턴
*/
extern PMSCONN_API      LPCTSTR PMSGetConfigFileName();
/*
PMSGetStatus PMSConn의 현재 상태를 가져 오기 위한 함수
@return value : PMSCONN_STATUS_CODE 값을 리턴
*/
extern PMSCONN_API      DWORD   PMSGetStatus();
```

6.4 IPMSObject Inteface 설명

6.4.1 HA로부터 요청이 왔을 때 실제적인 처리를 위해 만들어진 Interface이다. Call Back 함수로 구현이 가능 하다. 하지만 Wrapping하기에는 Call Back 함수보다 Interface 형태가 편리 하기 때문에 Inteface로 구현 하였다. 자세한 내용은 PMSObject.h를 참조 한다.

6.4.2 IPMSObject.h 의 기술

```
typedef std::vector<LONG> VLONG;
/*
OnRegionInfoReq 함수가 불리워 졌을때 사용자 지역 정보를 PMSConn을 통해 HA로 전송하기 위한 데이터를 생성하는 Interface
@param1 : 게임 Serial Number
@param2 : 게임별 카테고리
@param3 : 사용자 지역정보를 가지고 있는 Vector<LONG> 일반적으로 UserInfo[i][j][k] Array로 되어 있을 경우 (i, j, k) 일때 (0,0,0) (0,0,1) ~ (0,1,0) ~ (1,0.0) 의 순으로 입력 하면 된다.
* 반드시 [지역/Region/17] [연령/Age/7] [성별/Gender/2] 순서로 넣어야 한다 *
@return value 없음
```

```
*/
interface IPMSReionInfoList
{
	virtual void AddRegionInfo(DWORD dwSSN, DWORD dwCategoryID, VLONG* pvResionStat)
= 0;
};
/*
IPMSStatInfoList 현재 게임 서버의 통계 정보를 PMSConn을 이용하여 전송하기 위한 데이터를
생성 하는 Interface
@param1 : 게임 Serial Number
@param2 : 게임별 카테고리
@param3 : @param1, @param2의 조건을 만족하는 사용자 수
@param4 : 현재 게임 서버에 생성되어 있고 @param1, @param2의 조건을 만족하는 Channel 수
@param5 : 현재 게임 서버에 생성되어 있고 @param1, @param2의 조건을 만족하는 게임 방 수
@param6 : Optional 한 데이터
@return value 없음
*/
interface IPMSStatInfoList
{
	virtual void AddStatInfoList( DWORD dwSSN, DWORD dwCategory, DWORD dwCU,
		DWORD dwSession,	DWORD dwChannelCnt, DWORD dwRoomCnt, LPCSTR
pszOptionInfo) = 0;
};
/*
IPMSPerformanceInfo현재 게임 서버의 성능 정보를 PMSConn을 이용하여 전송하기 위한 데이터
를 생성 하는 Interface
@param1 : 게임 서버의 상태 정보를 담고 있는 long type의 vector pointer
@return value 없음
*/

interface IPMSPerformanceInfo
{
	virtual void AddPerformanceInfo(VLONG* pvPerformance) = 0;
};

/*
IPMSObject interface
```

```
PMSConn에서 호출을 위하여 생성된 Call Back Object Interface
*/
interface IPMSObject
{
public:
	/*
		Heart Beat Request 가 왔을때 불리는 함수
		@param1 : sequence index
		@return value : 정상 상태 TRUE , 비정상 FALSE
	*/
	virtual BOOL OnHeartbeatReq(LONG lIndex) = 0;
	/*
		Order Request 가 왔을때 불리는 함수
		@param1 : Order 이름 - NULL Terminate String
		@param2 : Order 인자 - NULL Terminate String
		@param3 : 결과값 버퍼 - 성공일때 무시, 실패일때 실패 사유 입력
		@param4 : 결과값 버퍼 크기
		@param5 : 카테고리 ID
		@return value : 성공 TRUE , 실패 FALSE
	*/
	virtual BOOL OnOrderReq(LPCSTR lpszCmdName, LPCSTR lpszCtlVal, LPSTR lpResult,
LONG *lpResultLen, DWORD dwSSN, DWORD dwCategoryID) = 0;
	/*
	공지 메시지가 왔음을 알리는 함수
	@param1 : 게임 Serial Number
	@param2 : 게임별 카테고리
	@param3 : 공지 메시지 - NULL Terminate String
	@return value : 성공 TRUE , 실패 FALSE
	*/
	virtual BOOL OnAnnounceReq(DWORD dwSSN, DWORD dwCategoryID, LPCSTR lpszMsg) = 0;
	/*
	@param1 : IPMSPerformanceInfo * 의 포인터
	@return value : 성능정보값이 있을때 TRUE, 없을때 FALSE 리턴
	*/
	virtual BOOL OnPerformInfoReq(IPMSPerformanceInfo *pvPerform) = 0;
	/*
	@param1 : (IPMSReionInfoList* * 의 포인터
```

```
        @return value : 지역정보값이 있을때 TRUE, 없을때 FALSE 리턴
        */
        virtual BOOL OnRegionInfoReq(IPMSReionInfoList* plstRegionInfo) = 0;
        /*
        @param1 : IPMSStatInfoList * 의 포인터
        @return value : 통계정보값이 있을때 TRUE, 없을때 FALSE 리턴
        */
        virtual BOOL OnStatInfoReq(IPMSStatInfoList* plstStatInfo) = 0;
};
```

7 Using PMSConn API Wrapper Class

7.1 PMSConn API를 보다 효율적이고 쉽게 적용가능 하도록 하기 위하여 만들었다. 이 Wrapper Class를 사용한 예제는 PMSConnTester에 모두 구현이 되어 있다.

7.1.1 CPMSObject Class의 경우는 IPMSObject Interface의 필수 구현 함수인인 OnHeartBeatReq 요청을 제외하고 모두 구현이 되어 있다. 하지만 실제적으로 각각의 요청에 대해서 처리를 해야 하는 모듈의 경우에는 모두 재정의 하여 구현 해야 한다.

7.1.2 CPMSConnWrapper 클래스는 기본적인 PMSConn API를 Wrapping한 Class로 되어있다. 각각의 API의 인자는 3번 항목에서 모두 설명하였으므로 설명을 생략한다.

7.2 PMSConnWrapper.h의 기술

```
struct CPMSObject : public IPMSObject
{
// Heart Beat 부분은 반드시 직접 구현 해야 한다.
//      virtual BOOL OnHeartbeatReq(LONG lIndex) = 0;
        virtual BOOL OnOrderReq(LPCSTR lpszCmdName, LPCSTR lpszCtlVal, LPSTR lpResult,
LONG *lpResultLen, DWORD dwSSN, DWORD dwCategoryID){ return TRUE;}
        virtual BOOL OnAnnounceReq(DWORD dwSSN, DWORD dwCategoryID, LPCSTR
lpszMsg){ return TRUE;}
        virtual BOOL OnPerformInfoReq(IPMSPerformanceInfo *pPerformanceInfo) { return
TRUE;}
        virtual BOOL OnRegionInfoReq(IPMSReionInfoList* plstRegionInfo){ return TRUE;}
        virtual BOOL OnStatInfoReq(IPMSStatInfoList* plstStatInfo){ return TRUE;}
};

template<typename BASE = CPMSObject>
class CPMSConnWrapper : public BASE
{
//      CPMSConnWrapper()
public:
        virtual ~CPMSConnWrapper(){ Stop();}
public:
        // PMSConn API를 호출 하기 위한 함수
        DWORD   Init(DWORD argc, LPTSTR argv[]){return ::PMSInitConn(argc, argv); }
        DWORD   Run(){ return ::PMSRunConn(this); }
        void    Stop(){ ::PMSStopConn(); }
        BOOL    SendWarningMsg(DWORD dwErrLvl, LPCSTR pszWarningMsg, LPCSTR pszTreatMsg,
DWORD dwSSN, DWORD dwCategory)
```

```
        {
                return ::PMSSendWarningMsg(dwErrLvl, pszWarningMsg, pszTreatMsg, dwSSN,
dwCategory);
        }
        LPCTSTR GetConfigFileName();
        DWORD   GetStatus();
};
```

8 Attention!

8.1 PMSConn은 지금 현재 Single Thread로 모든 것이 동작 하도록 구현이 되어 있다.

8.1.1 하나의 메시지 요청을 처리 하는데 시간이 많이 걸리게 될 경우 예기치 않은 HeartBeat에러가 날 수 있음으로 주의해야 한다.

8.1.2 또한 게임서버와는 다른 Thread에서 처리되므로 반드시 Heart Beat 응답을 받았을 때는 다른 Thread or Thread Pool이 정상 동작하는지 체크를 하고 PMS로 Heart Beat 응답을 보내도록 해야 한다. 그렇지 않을 경우 PMS에서 Hang과 같은 장애를 감지할 수 없다.