FDK API Manual for C++

June 2015

FN Pricing

Contents

Overview

This document is the API specification for C++ of the CallFDK.

The CallFDK is the API Library for using FDK engine provided by ITS.

System Environments

C++ compiler

Installation files

Linux

- callfdk shared object binary
  - libcallfdk.so
  - libcallfdk_dat.so
  - libcallfdk_msg.so
  - libxercesc-c*
- MsgDic.exml : FDK Service Message Dictionary file
- UserKey.lic : API identification key file
- callfdk.cfg : CallFDK API configuration file
- test_main.cpp : sample program source file
- Makefile : sample makefile

Runtime Environments

Register the directory which the CallFDK API library is installed at LD_LIBRARY_PATH.

Linux/Unix

```
$ export LD_LIBRARY_PATH=/path/to/lib:$LD_LIBRARY_PATH
```

Sample codes for using CallFDK API

Example : test_main.cpp

```
#include <stdio.h>
#include "dat_class.h"
#include "callfdk.h"

#include "M_IrNoteVanilla.h"
#include "M_IrNoteOut.h"
```

```
#define DUMP_MSG(m) \
     aDump.Clear(); \
     m.Dump(&aDump); \
     printf("--------------\n"); \
     printf("%s\n", (char *)aDump.GetData())


void MakeInMsg(M_IrNoteVanilla *p_pIn);

int
main(int argc, char **argv)
{
     int iRet;
     CCallFdkError aError;
     CStream aDump;

     try {
            iRet = CCallFdk::Initialize("callfdk.cfg", &aError);
            if (iRet < 0) {
                   printf("Error: %s\n", (char *)aError.Repr());
                   return -1;
            }
            M_IrNoteVanilla mIn;
            M_IrNoteOut mOut;

            MakeInMsg(&mIn);

            printf("CallService 2106 ...\n");
            iRet = CCallFdk::CallService(2106, &mIn, &mOut, &aError);
            if (iRet < 0) {
                   printf("Error: %s\n", (char *)aError.Repr());
                   return -2;
            }

            DUMP_MSG(mOut);
            printf("OK.\n");
     } catch (CException &e) {
            printf("Error: %d, %s\n", e.GetCode(), e.GetText());
            return -3;
     }

     return 0;
}


void
MakeInMsg(M_IrNoteVanilla *p_pIn)
{
     int i;
     int iIdx;
     struct _RateCrv {
            char szRateType[10];
            double dRate;
            char szDateType[5];
            char szMatur[10];
            char szCmpndFreq3[10];
            char szDayCntConv[10];
     };
```

```
struct _Sched {
        int iPeriodNo;
        char szFormerDate[9];
        char szAfterDate[9];
        char szCashflowDate[9];
};


p_pIn->mIrNotionalInfo.dNotionAmt = 10000.;
p_pIn->mIrNotionalInfo.sCcy = "KRW";
p_pIn->mIrNotionalInfo.iDisCrvId = 1;

iIdx = p_pIn->mIrNotionalInfo.maIrAmortSchedule.Grow();
M_IrAmortSchedule *pAmort =
        &p_pIn->mIrNotionalInfo.maIrAmortSchedule[iIdx-1];
pAmort->iPeriodNo = 7;
pAmort->dAmortAmt = 3000.;

iIdx = p_pIn->mIrNotionalInfo.maIrAmortSchedule.Grow();
pAmort = &p_pIn->mIrNotionalInfo.maIrAmortSchedule[iIdx-1];
pAmort->iPeriodNo = 8;
pAmort->dAmortAmt = 7000.;

p_pIn->mIrVanilla.sPayRcvSect = "P";
p_pIn->mIrVanilla.sPayTmPtC = "R";
p_pIn->mIrVanilla.iFixedPeriodNo = 1;
p_pIn->mIrVanilla.dFixedRate = 0.03;
p_pIn->mIrVanilla.sDayCntConv = "A365";

struct _Sched taSched[8] = {
        {1, "20150625", "20150925", "20150925"},
        {2, "20150925", "20151225", "20151225"},
        {3, "20151225", "20160325", "20160325"},
        {4, "20160325", "20160625", "20160625"},
        {5, "20160625", "20160925", "20160925"},
        {6, "20160925", "20161225", "20161225"},
        {7, "20160925", "20170325", "20170325"},
        {8, "20170325", "20170625", "20170625"}
};

for (i = 0; i < 8; i++) {
        iIdx = p_pIn->mIrVanilla.maIrCpnSchedule.Grow();
        M_IrCpnSchedule *pSch =
                &p_pIn->mIrVanilla.maIrCpnSchedule[iIdx-1];
        pSch->iPeriodNo = taSched[i].iPeriodNo;
        pSch->nFormerDate   = taSched[i].szFormerDate;
        pSch->nAfterDate    = taSched[i].szAfterDate;
        pSch->nCashflowDate = taSched[i].szCashflowDate;
}

M_FdkRateIndexB *pRateIdx =
                &p_pIn->mIrVanilla.mIrVanillaCpn.mFdkRateIndexB;
pRateIdx->sRateIndexCode = "CD91D";
pRateIdx->sCcy = "KRW";
pRateIdx->sRateType = "S";
pRateIdx->sRateMatur = "91D";
pRateIdx->sCmpndFreq3 = "I";
pRateIdx->sDayCntConv = "A365";
pRateIdx->iRateCrvId = 1;
```

```
    pRateIdx->dGearing = 1.;
    p_pIn->mIrVanilla.mIrVanillaCpn.dMargin = 0.;
    p_pIn->mIrVanilla.mIrVanillaCpn.sDayCntConv = "A365";

    iIdx = p_pIn->mIrVanilla.maIrVanillaMon.Grow();
    M_IrVanillaMon *pMon = &p_pIn->mIrVanilla.maIrVanillaMon[iIdx-1];
    pMon->iPeriodNo = 2;
    pMon->bCouponFixingYN = true;
    pMon->dCouponRate = 0.023;


    // Parameter setting
    p_pIn->mIrParam.nCalcDate = "20150625";

    iIdx = p_pIn->mIrParam.maFdkRateCrvB.Grow();
    M_FdkRateCurveB *pCrv = &p_pIn->mIrParam.maFdkRateCrvB[iIdx-1];
    pCrv->iRateCrvId = 1;
    pCrv->sCcy = "KRW";
    pCrv->sInterpMethod = "L";

    struct _RateCrv taRateCrv[] = {
           {"S", 0.0223,   "S", "1D", "I", "ACTB"},
           {"Y", 0.0235,   "S", "3M", "Q", "A365"},
           {"Y", 0.022775, "S", "6M", "Q", "A365"},
           {"Y", 0.022575, "S", "9M", "Q", "A365"}
    };

    // Rate Curve
    for (i = 0; i < 4; i++) {
           iIdx = pCrv->maRateCrvTenor.Grow();
           M_RateCurveTenor *pTnr = &pCrv->maRateCrvTenor[iIdx-1];
           pTnr->sRateType = taRateCrv[i].szRateType;
           pTnr->dRate = taRateCrv[i].dRate;
           pTnr->mTivMatur.sDateType = taRateCrv[i].szDateType;
           pTnr->mTivMatur.sMatur = taRateCrv[i].szMatur;
           pTnr->sCmpndFreq3 = taRateCrv[i].szCmpndFreq3;
           pTnr->sDayCntConv = taRateCrv[i].szDayCntConv;
    }

    struct _RateCrv taRateCrv2[] = {
           {"Y", 0.022375,  "S", "1Y",   "Q", "A365"},
           {"Y", 0.022725,  "S", "2Y",   "Q", "A365"},
           {"Y", 0.02305,   "S", "2Y6M", "Q", "A365"},
           {"Y", 0.023375,  "S", "3Y",   "Q", "A365"},
           {"Y", 0.024725,  "S", "5Y",   "Q", "A365"},
           {"Y", 0.02575,   "S", "7Y",   "Q", "A365"},
           {"Y", 0.02705,   "S", "10Y",  "Q", "A365"}
    };

    // Rate Curve 2
    for (i = 0; i < 7; i++) {
           iIdx = pCrv->maRateCrvTenor2.Grow();
           M_RateCurveTenor *pTnr = &pCrv->maRateCrvTenor2[iIdx-1];
           pTnr->sRateType = taRateCrv2[i].szRateType;
           pTnr->dRate = taRateCrv2[i].dRate;
           pTnr->mTivMatur.sDateType = taRateCrv2[i].szDateType;
           pTnr->mTivMatur.sMatur = taRateCrv2[i].szMatur;
           pTnr->sCmpndFreq3 = taRateCrv2[i].szCmpndFreq3;
           pTnr->sDayCntConv = taRateCrv2[i].szDayCntConv;
```

```
      }

      // Forward curve
      iIdx = pCrv->maFwdCrvTenor.Grow();
      M_FwdCurveTenor *pFwd = &pCrv->maFwdCrvTenor[iIdx-1];
      pFwd->dRate = 0.027;
      pFwd->nFwdEndDate = "20160126";
      pFwd->sMatur = "3M";
      pFwd->sCmpndFreq3 = "I";
      pFwd->sDayCntConv = "A365";

      iIdx = pCrv->maFwdCrvTenor.Grow();
      pFwd = &pCrv->maFwdCrvTenor[iIdx-1];
      pFwd->dRate = 0.029;
      pFwd->nFwdEndDate = "20160226";
      pFwd->sMatur = "3M";
      pFwd->sCmpndFreq3 = "I";
      pFwd->sDayCntConv = "A365";


      // flags
      p_pIn->mIrEffSensTreeParam.bDeltaYN = false;
      p_pIn->mIrEffSensTreeParam.sRDeltaType = "RP";
      p_pIn->mIrEffSensTreeParam.bVegaYN = false;
      p_pIn->mIrEffSensTreeParam.bFxVegaYN = false;
      p_pIn->mIrEffSensTreeParam.bThetaYN = true;
      p_pIn->mIrEffSensTreeParam.bReCalibYN = true;
      p_pIn->mIrEffSensTreeParam.bSpotDeltaYN = true;
      p_pIn->mIrEffSensTreeParam.bAccumDeltaYN = true;
      p_pIn->mIrEffSensTreeParam.sThetaType = "T";

      p_pIn->bDiffSensYN = true;
}
```

1. Include Input and Output message header files
2. Call CCallFdk::Initialize()
3. Make new input message and setting data(mIn)
4. Call CCallFdk.CallService()
5. Use output message(mOut)

CCallFdk

This class is the main class for calling the service of FDK engine.

static int Initialize(const char *p_szCfgFile, CCallFdkError *p_pError)

Description

This method is used to initialize the CallFDK API library with a given configuration file. This method must be called only once in the entire program.

Parameters

- const char *p_szCfgFile : Configuration file path
- CCallFdkError *p_pError : CCallFdkError instance.

Return : int

If this return value is less than zero, it means that an error occurs in this method call.
If an error occurs in this method, the reason of error is stored in CCallFdkError instance.

Example

```
CCallFdkError aError;
try {
  iRet = CCallFdk::Initialize("/path/to/callfdk.cfg", &aError);
  if (iRet < 0) {
    printf("Error: %s\n", (char *)aError.Repr());
    return -1;
  }
} catch (CException &e) {
  printf("Error: %s\n", e.GetText());
  return -1;
}
```

Configuration file sample

```
key_file=UserKey.lic
msg_dic=MsgDic.exml
server=sq.fnpricing.com 5300
timeout=90
```

key_file and msg_dic keywords both in full or relative path can be expressed.

static int Initialize(…)

Description

This method is the alternatives of the previous Initialize() method. This is used to initialize the CallFDK API library with each argument. This method must be called only once in the entire program.

This is useful when a developer want to manage own configuration file.

Parameters

- const char *p_szKeyFile : API Identification key file path
- const char *p_szDicFile : FDK Service Message Dictionary file path
- const char *p_szServer : FDK server address(DNS name or IP address)
- int p_iPort : FDK server port number
- int p_iTimeout : timeout(seconds) which the API waits for the response of the FDK server
- CCallFdkError *p_pError : CCallFDKError instance for error

Return : int

If this value is less than zero, an error occurs in this method call. The reason of error is stored in CCallFdkError instance.

Example

```
CCallFdkError aError;
try {
  iRet = CCallFdk::Initialize(
      "/path/to/UserKey.lic",
      "/path/to/MsgDic.exml",
      "sq.fnpricing.com",
      5300,
      90,
      &aError
  );
  if (iRet < 0) {
    printf("Error: %s\n", (char *)aError.Repr());
    return -1;
  }
} catch (CException e) {
  printf("Error: %s\n", e.GetText());
  return -1;
}
```

static int CallService(…)

Description

This method is used to call service which falls on p_iSvcNum with input message(p_pIn) instance. The response from the FDK server is stored in ouput message(p_pOut) instance

Parameters

- int p_iSvcNum : Service number
- CMsgBase *p_pIn : Input message instance
- CMsgBase *p_pOut : Output message instance
- CCallFdkError *p_pError : CCallFdkError instance

For the service number, input message and output message, refer to FDK Service Message Reference Manual

Return : int

If this return value is less than zero, an error occurs in this method call. The reason of error is stored in CCallFdkError instance

Example

```
try {
  int iIdx;
  mIn.mIrNotionalInfo.dNotionAmt = 100000.;
  mIn.mIrNotionalInfo.sCcy = "USD";
  iIdx = mIn.mIrNotionalInfo.maIrAmortSchedule.Grow();
  mIn.mIrNotionalInfo.maIrAmortSchedule[iIdx-1].iPeriodNo = 1;
  mIn.mIrNotionalInfo.maIrAmortSchedule[iIdx-1].dAmortAmt = 364000;

  iIdx = mIn.mIrNotionalInfo.maIrAmortSchedule.Grow();
  mIn.mIrNotionalInfo.maIrAmortSchedule[iIdx-1].iPeriodNo = 2;
  mIn.mIrNotionalInfo.maIrAmortSchedule[iIdx-1].dAmortAmt = 664000;

  iRet = CCallFdk::CallService(2106, &mIn, &mOut, &aError);
  if (iRet < 0) {
    printf("Error: %s\n", (char *)aError.Repr());
    return -1;
  }
  printf("%f\n", mOut.maIrNotePrice[0].dDirtyPrice);
  printf("%f\n", mOut.maIrNotePrice[0].dCleanPrice);
  printf("%s\n", (char *)mOut.maIrNotePrice[0].sPriceType);
  // ...
} catch (CException &e) {
  printf("Error: %d, %s\n", e.GetCode(), (char *)e.Message());
  return -1;
}
```

CCallFdkError

This class is used to handle errors occurring when CCallFDK method is called.

int GetCode()

Description

This method returns the error code CCallFdkError instance hold.

Return : int

Error code value

Example

```
CCallFdkError aError;
int iRet = CCallFdk.CallService(..., &aError);
if (iRet < 0) {
  printf("Error:%d,%s\n", aError.GetCode(),
                          (char *)aError.GetText());
  return -1;
}
```

CString GetText()

Description

This method returns the error text which CCallFdkError instance hold.

Return : CString

Error text string

Example

```
CCallFdkError aError;
int iRet = CCallFdk.CallService(..., &aError);
if (iRet < 0) {
  printf("Error:%d,%s\n", aError.GetCode(),
                          (char *)aError.GetText());
  return -1;
}
```

CString Repr()

Description

This method returns the error text and the code as the formatted string which CCallFdkError instance hold.

Example

```
CCallFdkError aError;
int iRet = CCallFdk.CallService(..., &aError);
if (iRet < 0) {
  printf("Error:%d,%s\n", (char *)aError.Repr());
  return -1;
}
```

CException

This class is the exception class which the CallFDK API occurs.

int GetCode()

Description

This method returns the error code which CException instance hold.

Return : int

Error code value

Example

```
CCallFdkError aError;
}
try {
  int iRet = CCallFdk.CallService(..., &aError);
  if (iRet < 0) {
    printf("Error: %s\n", (char *)aError.Repr());
    return -1;
  }
} catch (CException &e) {
  printf("Error: %d, %s\n", e.GetCode(), (char *)e.GetText());
  return -1;
}
```

char *GetText()

Description

This method returns the error text which CException instance hold.

Return : char *

Error text string

Example

```
CCallFdkError aError;
}
try {
  int iRet = CCallFdk.CallService(..., &aError);
  if (iRet < 0) {
    printf("Error: %s\n", (char *)aError.Repr());
    return -1;
  }
} catch (CException &e) {
  printf("Error: %d, %s\n", e.GetCode(), (char *)e.GetText());
  return -1;
}
```

Service / Message

Service

The Service is a basic unit of the transaction with the FDK engine. This is represented by the service number. Services are classified by the product type, the algorithm and the feature in detail. Refer to Service Message Reference Manual.

Message Object

The Message Object is a basic unit of the data block which the CallFDK API exchanges with the FDK server. This is divided into two objects such as the request and the response message.
Request message : the data which the API sends to the FDK server.
Response message : the data which the API receives from the FDK server.
Each Message Object can hold other message object as sub message.
All of Message Objects are implemented as classes and they are in jar file which the API provides.
The name of Message classes starts with "M_"

The Structure of Message Object

The Message Object can hold the data types as following.

Data type	API Data Type	API defined prefix	Example of variable
string	CString	s	sPriceType
integer	CInt	i	iPreiodNo
double	CDbl	d	dNotionalAmt
boolean	CBool	b	bFixedYN
date(YYYYMMDD)	CDate	n	nMaturDate
sub message	M_* class	m	mNotionalInfo
string array	CStrArray	sa	saCcyList
integer array	CIntArray	ia	iaTermNo
double array	CDblArray	da	daGearing
boolean array	CBoolArray	ba	baGreeksYN
date array	CDateArray	na	naExpiryDate
sub message array	M_*_Array class	ma	maCashFlowInfo

All of data fields in which a message class holds are declared as public.

Setting/Getting of Message data

Example of setting a single message

```
M_IrNoteVanilla mIn;
mIn.mIrNotionalInfo.dNotionAmt = 100000.;
mIn.mIrNotionalInfo.sCcy = “USD”;
```

Example of setting sub message array

```
int iIdx;
M_IrNoteVanilla mIn;
mIn.mIrNotionalInfo.dNotionAmt = 100000.;
mIn.mIrNotionalInfo.sCcy = “USD”;
iIdx = mIn.maIrAmortSchedule.Grow();
mIn.maIrAmortSchedule[iIdx-1].iPeroidNo = 1;
mIn.maIrAmortSchedule[iIdx-1].dAmortAmt = 100000.;

iIdx = mIn.maIrAmortSchedule.Grow();
mIn.maIrAmortSchedule[iIdx-1].iPeroidNo = 2;
mIn.maIrAmortSchedule[iIdx-1].dAmortAmt = 500000.;
//...
```

Example of refering message data

```
// ...
int iRet = CCallFdk::CallService(2106, &mIn, &mOut, &aError);
//...

for (i = 0; i < mOut.maIrNotePrice.GetSize(); i++) {
  printf(“PriceType : %s“, (char *)mOut.maIrNotePrice[i].sPriceType);
  printf(“dDirtyPrice : %d“, (double)mOut.maIrNotePrice[i].dDirtyPrice);
  // ...
  for (i = 0; i < mOut.maIrNoteCF.GetSize(); i++) {
    M_IrNoteCF *m2 = &mOut.maIrNoteCF[i];
    printf(“PeriodNo : %d”, (int)m2->iPeriodNo);
    printf(“nFormerDate : %s“, m2->nFormerDate.ToStr());
  }
}
```

Data Classes

This section explains about API data classes which the message object holds.

CString

This class is used to manage the string data of message instances

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CString s;

if (s.IsValid()) {
  printf("valid data(%s)\n", s.ToStr());
} else {
  printf("unset data\n");
}
// printed "unset data"

s = "hello world";
if (s.IsValid()) {
  printf("valid data(%s)\n", s.ToStr());
} else {
  printf("unset data\n");
}
// printed "valid data(hello world)"
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CString s = "Hello World";
if (s.IsValid()) {
  printf("valid data(%s)\n", s.ToStr());
} else {
  printf("unset data\n");
}
// printed "valid data(hello world)"
s.SetNull();
```

```
if (s.IsValid()) {
  printf("valid data(%s)\n", s.ToStr());
} else {
  printf("unset data\n");
}
// printed "unset data"
```

int GetSize()

Description

This method is used to get the length of string

Return : int

the length of string

Example

```
CString s = "Hello world";
printf("length: %d\n", s.GetSize());  // printed "length: 11"
```

char *ToStr()

Description

This method is used to get the pointer to string as 'char *'

Return : char *

The pointer to string

Example

```
CString s = "Hello world";
printf("data : %s\n", s.ToStr());
```

Assignment & concatenation operators

CString &operator=(const CString &);

CString &operator=(const char *);

CString &operator+=(const CString &);

CString &operator+=(const char *);

CString operator+(const CString &);

CString operator+(const char *);

friend CSring operator+(const char *, const CString &);

Description

The operators related to assignment and concatenation are overloaded

Example

```
CString s;
CString t;

s = "Hello";
```

```
s += " world";
t = s + ", What a beautiful";
```

Comparing operators

Description

The comparing operators of between CString and 'char *' are overloaded

CString == CString

CString == char *

char * == CString

!=, >, <, >=, <=

Example

```
CString s = "abc";
CString t = "xyz";

if (s == t) {
  printf("s equals to t\n");
} else (s < t) {
  printf("s is less than t\n");
} else (s > t) {
  printf("s is more than t\n");
}
// printed "s is less than t"

if (s <= "aaa") {
  printf("s is less than aaa\n");
} else {
  printf("s is more than aaa\n");
}
```

CInt

This class is the wrapper class of 'int' data type.

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CInt iVal;

if (iVal.IsValid()) {
  printf("valid data(%d)\n", iVal.ToInt());
} else {
  printf("unset data\n");
}
// printed "unset data"

iVal = 100;
if (iVal.IsValid()) {
  printf("valid data(%d)\n", iVal.ToInt());
} else {
  printf("unset data\n");
}
// printed "valid data(100)"
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CInt iVal = 100;

if (iVal.IsValid()) {
  printf("valid data(%d)\n", iVal.ToInt());
} else {
  printf("unset data\n");
}
// printed "valid data(100)"

iVal.SetNull();
if (iVal.IsValid()) {
  printf("valid data(%d)\n", iVal.ToInt());
} else {
  printf("unset data\n");
}
// printed "unset data"
```

int ToInt()

Description

This method is used to get the integer value from the instance

Return : int

int value

Example

```
CInt iVal = 365;

printf("value : %d\n", iVal.ToInt());
```

Assignment & arithmetic operators

Description

The assignment and arithmetic operators between CInt and 'int' type are overloading.

Example

```
CInt iVal = 100;
CInt iData;
int  a = 300;

iVal += 30;
iData = iVal + a;
iData = 365 + iVal;
```

Comparing operators

Description

The comparing operators of between CInt and 'int' are overloaded.

CInt == CInt

CInt == int

int == CInt

!=, >, <, >=, <=

Example

```
CInt s = 100;
CInt t = 200;

if (s == t) {
  printf("s equals to t\n");
} else (s < t) {
  printf("s is less than t\n");
} else (s > t) {
  printf("s is more than t\n");
}
// printed "s is less than t"

if (s <= 365) {
  printf("s is less than 365\n");
} else {
  printf("s is more than 365\n");
}
```

CDbl

This class is the wrapper class of 'double' data type

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CDbl dVal;

if (dVal.IsValid()) {
  printf("valid data(%f)\n", dVal.ToDbl());
} else {
  printf("unset data\n");
}
// printed "unset data"

dVal = 3.141592;
if (dVal.IsValid()) {
  printf("valid data(%f)\n", dVal.ToDbl());
} else {
  printf("unset data\n");
}
// printed "valid data(3.141592)"
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CDbl dVal = 3.141592;

if (dVal.IsValid()) {
  printf("valid data(%f)\n", dVal.ToDbl());
} else {
  printf("unset data\n");
}
// printed "valid data(3.141592)"

dVal.SetNull();
if (dVal.IsValid()) {
  printf("valid data(%f)\n", dVal.ToDbl());
} else {
  printf("unset data\n");
}
// printed "unset data"
```

double ToDbl()

Description

This method is used to get 'double' type data

Return : double

double value

Example

```
CDbl dVal = 3.141592;

printf("value : %f\n", dVal.ToDbl());
```

Assignment & arithmetic operators

Description

The assignment and arithmetic operators between CDbl and 'double' type are overloading.

Example

```
CDbl dVal = 3.141592;
CInt dData;
double a = 0.015;

dVal += 1.414;
dData = dVal + a;
dData = 1.732 + dVal;
```

Comparing operators

Description

The comparing operators of between CDbl and 'double' are overloaded.

CDbl == CDbl

CDbl == double

double == CDbl

!=, >, <, >=, <=

Example

```
CDbl s = 1.414;
CDbl t = 1.732;

if (s == t) {
  printf("s equals to t\n");
} else (s < t) {
  printf("s is less than t\n");
} else (s > t) {
  printf("s is more than t\n");
}
// printed "s is less than t"

if (s <= 3.141592) {
  printf("s is less than 3.141592\n");
} else {
  printf("s is more than 3.141592\n");
}
```

CBool

This class is the wrapper class of bool type.

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CBool bVal;

if (bVal.IsValid()) {
  printf("valid data(%s)\n", (bVal) ? "true" : "false");
} else {
  printf("unset data\n");
}
// printed "unset data"

bVal = true;
if (bVal.IsValid()) {
  printf("valid data(%s)\n", (bVal) ? "true" : "false");
} else {
  printf("unset data\n");
}
// printed "valid data(true)"
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CBool bVal;

if (bVal.IsValid()) {
  printf("valid data(%s)\n", (bVal) ? "true" : "false");
} else {
  printf("unset data\n");
}
```

```
// printed "unset data"

bVal = true;
if (bVal.IsValid()) {
  printf("valid data(%s)\n", (bVal) ? "true" : "false");
} else {
  printf("unset data\n");
}
// printed "valid data(true)"
```

Assignment operators

Description

The assignment operators between CBool and 'bool' type are overloading.

Example

```
CBool bVal;
bool bState = true;
bVal = bState;
bState = bVal;
```

Comparing operators

Description

The comparing operators of between CDbl and 'double' are overloaded.

CBool == CBool

CBool == bool

bool == CBool

!=

Example

```
Cbool bVal = true;

if (bVal) {
}

if (bVal != true) {
}

if (bVal == false) {
}
```

CDate

This class is used to manage the date type string. A date string is expressed as 'YYYYMMDD'

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CDate nDate;

if (nDate.IsValid()) {
  printf("valid data(%s)\n", nDate.ToStr());
} else {
  printf("unset data\n");
}
// printed "unset data"

nDate = "20150615";
if (nDate.IsValid()) {
  printf("valid data(%d)\n", nDate.ToStr());
} else {
  printf("unset data\n");
}
// printed "valid data(20150615)"
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CDate nDate;

if (nDate.IsValid()) {
  printf("valid data(%s)\n", nDate.ToStr());
} else {
  printf("unset data\n");
}
// printed "unset data"

nDate = "20150615";
if (nDate.IsValid()) {
  printf("valid data(%d)\n", nDate.ToStr());
} else {
  printf("unset data\n");
}
// printed "valid data(20150615)"
```

char *ToStr()

Description

This method is used to get a date string as 'char *"

Return : char *

char *

Example

```
CDate nDate = "20150615"

printf("value : %s\n", nDate.ToStr());  // value : 20150615
```

int Year()

Description

This method is used to get the year as an integer value.

Return : int

integer value of the year

Example

```
CDate nDate = "20150615";
printf("Year : %04d\n", nDate.Year());
```

int Month()

Description

This method is used to get the month as an integer value.

Return : int

integer value of the month

Example

```
CDate nDate = "20150615";
printf("Month : %02d\n", nDate.Month());
```

int Day()

Description

This method is used to get the day as an integer value.

Return : bool

integer value of the day

Example

```
CDate nDate = "20150615";
printf("Day : %02d\n", nDate.Day());
```

Assignment operators

Description

The assignment operators between CDate and 'char *' type are overloading.

Example

```
CDate nDate;
CDate nDate2;

nDate = "20150615";
nDate2 = nDate;
```

Comparing operators

Description

The comparing operators of between CBool and 'char *' are overloaded.

CDate == CDate

CDate == char *

char * == CDate

!=, >, <, >=, <=

Example

```
CDate s = "20150615";
CDate t = "20150630";

if (s == t) {
  printf("s equals to t\n");
} else (s < t) {
  printf("s is less than t\n");
} else (s > t) {
  printf("s is more than t\n");
}
// printed "s is less than t"

if (s <= "20150720") {
  printf("s is less than 20150720\n");
} else {
  printf("s is more than 20150720\n");
}
```

M_{msgname}

This class is the message class which is expressed by FDK Service Message

All of field of this class are declared in public.

void Copy(M_{msgname} *)

Description

This method is used to copy from the given parameter to this instance. This executes deep-copy.

Example

```
M_IrVanilla mIr;
...

M_IrVanilla mIr2;
mIr2.Copy(&mIr);
```

void Dump(CStream *p_pBuffer)

Description

This method is used to dump the contents of this instance to string buffer. This is used to debug usually.

Example

```
M_IrVanilla mIr;
...
CStream aBuff;
mIr.Dump(&aBuff);
printf("%s\n", aBuff.GetData());
```

CStrArray

This class is used to manage the string array

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CStrArray saCcyList
...
if (saCcyList.IsValid()) {
  for (i = 0; i < saCcyList.GetSize(); i++) {
    printf("%s\n", saCcyList[i].ToStr());
  }
}
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CStrArray saCcyList;
...
saCcyList.SetNull();  // make mIn.saCcyList to the state of unset
if (mIn.saCcyList.IsValid() == false) {
  ...
}
```

int GetSize()

Description

This method is used to get the size of this array.

Return : int

The size of this array

Example

```
CStrArray saCcyList
...
if (saCcyList.IsValid()) {
  for (i = 0; i < saCcyList.GetSize(); i++) {
    printf("%s\n", saCcyList[i].ToStr());
  }
}
```

int Add(const char *p_szData)

Description

This method is used to add a new string to the end of this array.

Return : int

The index of the position of which a new data is added

Example

```
CStrArray saCcyList;
saCcyList.Add("KRW");
saCcyList.Add("USD");
saCcyList.Add("EUR");
```

char *GetAt(int p_iIdx)

Description

This method is used to get the string from this array by the given index.

Return : char *

The pointer to string

Example

```
CStrArray saCcyList;
...
for (i = 0; i < saCcyList.GetSize(); i++) {
  printf("%s\n", saCcyList.GetAt(i));
}
```

CString &operator[](int p_iIdx)

Description

This method is used to get the string from this array by the given index.

Return : CString &

CString instance

Example

```
CStrArray saCcyList;
...
for (i = 0; i < saCcyList.GetSize(); i++) {
  printf("%s\n", saCcyList[i].ToStr());
}
```

CIntArray

This class is used to manage an integer array.

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CIntArray iaVal;
if (iaVal.IsValid()) {
  for (i = 0; i < iaVal.GetSize(); i++) {
    printf("%d\n", iaVal[i]);
  }
}
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CIntArray iaVal;
...
iaVal.SetNull();  // make iaVal to the state of unset
if (iaVal.IsValid() == false) {
  ...
}
```

int GetSize()

Description

This method is used to get the size of this array.

Return : int

The size of this array

Example

```
CIntArray iaVal;
if (iaVal.IsValid()) {
  for (i = 0; i < iaVal.GetSize(); i++) {
    printf("%d\n", iaVal[i]);
  }
}
```

int Add(const int p_iNewData)

Description

This method is used to add a new integer value to the end of this array.

Return : int

The index of the position of which a new data is added

Example

```
CIntArray iaVal;
iaVal.Add(100);
iaVal.Add(200);
iaVal.Add(300);
```

int GetAt(int p_iIdx)

Description

This method is used to get the integer value from this array by the given index.

Return : int

an integer value

Example

```
CIntArray iaVal;
if (iaVal.IsValid()) {
  for (i = 0; i < iaVal.GetSize(); i++) {
    printf("%d\n", iaVal.GetAt(i));
  }
}
```

int &operator[](int p_iIdx)

Description

This method is used to get the string from this array by the given index.

Return : int &

an integer instance

Example

```
CIntArray iaVal;
if (iaVal.IsValid()) {
  for (i = 0; i < iaVal.GetSize(); i++) {
    printf("%d\n", iaVal[i]);
  }
}
```

CDblArray

This class is used to manage a double array

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CDblArray daVal;
if (daVal.IsValid()) {
  for (i = 0; i < daVal.GetSize(); i++) {
    printf("%f\n", daVal[i]);
  }
}
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CIntArray daVal;
...
daVal.SetNull();  // make daVal to the state of unset
if (daVal.IsValid() == false) {
  ...
}
```

int GetSize()

Description

This method is used to get the size of this array.

Return : int

The size of this array

Example

```
CDblArray daVal;
if (daVal.IsValid()) {
  for (i = 0; i < daVal.GetSize(); i++) {
    printf("%f\n", daVal[i]);
  }
}
```

int Add(const double p_dNewData)

Description

This method is used to add a new double value to the end of this array.

Return : int

The index of the position of which a new data is added

Example

```
CDblArray daVal;
daVal.Add(1.414);
daVal.Add(1.732);
daVal.Add(3.1415);
```

double GetAt(int p_iIdx)

Description

This method is used to get the double value from this array by the given index.

Return : double

a double value

Example

```
CDblArray daVal;
...
if (daVal.IsValid()) {
  for (i = 0; i < daVal.GetSize(); i++) {
    printf("%f\n", daVal.GetAt(i));
  }
}
```

double &operator[](int p_iIdx)

Description

This method is used to get the double value from this array by the given index.

Return : double &

a double instance

Example

```
CDblArray daVal;
if (daVal.IsValid()) {
  for (i = 0; i < daVal.GetSize(); i++) {
    printf("%f\n", daVal[i]);
  }
}
```

CBoolArray

This class is used to manage a bool array

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CBoolArray baVal;
if (baVal.IsValid()) {
  for (i = 0; i < baVal.GetSize(); i++) {
    printf("%s\n", (baVal[i]) ? "true" : "false");
  }
}
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CBoolArray baVal;
...
baVal.SetNull();  // make baVal to the state of unset
if (baVal.IsValid() == false) {
  ...
}
```

int GetSize()

Description

This method is used to get the size of this array.

Return : int

The size of this array

Example

```
CBoolArray baVal;
if (baVal.IsValid()) {
  for (i = 0; i < baVal.GetSize(); i++) {
    printf("%s\n", (baVal[i]) ? "true" : "false");
  }
}
```

int Add(const bool p_bNewData)

Description

This method is used to add a new boolean value to the end of this array.

Return : int

The index of the position of which a new data is added

Example

```
CBoolArray baVal;
baVal.Add(true);
baVal.Add(true);
baVal.Add(false);
```

bool GetAt(int p_iIdx)

Description

This method is used to get the boolean value from this array by the given index.

Return : bool

a bool value

Example

```
CBoolArray baVal;
...
if (baVal.IsValid()) {
  for (i = 0; i < baVal.GetSize(); i++) {
    printf("%s\n", (baVal.GetAt(i)) ? "true" : "false");
  }
}
```

bool &operator[](int p_iIdx)

Description

This method is used to get the boolean value from this array by the given index.

Return : bool &

a bool instance

Example

```
CBoolArray baVal;
...
if (baVal.IsValid()) {
  for (i = 0; i < baVal.GetSize(); i++) {
    printf("%s\n", (baVal[i]) ? "true" : "false");
  }
}
```

CDateArray

This class is used to manage a CDate array.

bool IsValid()

Description

This method is used to check if data is valid

Return : bool

If this return value is true, it means that data is valid. Otherwise, it means the data is not set.

Example

```
CDateArary naDate;
...
if (naDate.IsValid()) {
  for (i = 0; i < naDate.GetSize(); i++) {
    printf("%s\n", naDate[i].ToStr());
  }
}
```

void SetNull()

Description

This method is used to make the instance to the intialized state. If the instance has any data, that data will be removed

Example

```
CDateArray naDate;
...
naDate.SetNull();  // make naDate to the state of unset
if (naDate.IsValid() == false) {
  ...
}
```

int GetSize()

Description

This method is used to get the size of this array.

Return : int

The size of this array

Example

```
CDateArary naDate;
...
if (naDate.IsValid()) {
  for (i = 0; i < naDate.GetSize(); i++) {
    printf("%s\n", naDate[i].ToStr());
  }
}
```

int Add(const char *p_szDate)

Description

This method is used to add a new date string to the end of this array.

Return : int

The index of the position of which a new data is added

Example

```
CDateArray naDate;
naDate.Add("20150615");
naDate.Add("20150715");
naDate.Add("20150815");
```

CDate GetAt(int p_iIdx)

Description

This method is used to get the CDate value from this array by the given index.

Return : CDate

a CDate value

Example

```
CDateArary naDate;
...
if (naDate.IsValid()) {
  for (i = 0; i < naDate.GetSize(); i++) {
    printf("%s\n", naDate.GetAt(i).ToStr());
  }
}
```

CString &operator[](int p_iIdx)

Description

This method is used to get the CDate instance from this array by the given index.

Return : CDate &

a CDate instance

Example

```
CDateArary naDate;
...
if (naDate.IsValid()) {
  for (i = 0; i < naDate.GetSize(); i++) {
    printf("%s\n", naDate[i].ToStr());
  }
}
```

M_{msgname}_Array

This class is used to manage a M_{msgname} array

int Grow()

Description

This method is used to grow a new message instance in the array.

Return : int

The size of this array

Example

```
M_IrVanilla_Array maIr;
int iIdx = maIr.Grow();
maIr[iIdx-1].sCcy = "KRW";
maIr[iIdx-1].dAmount = 10000.;

iIdx = maIr.Grow();
maIr[iIdx-1].sCcy = "USD";
maIr[iIdx-1].dAmount = 20000.;
```

int GetSize()

Description

This method is used to get the size of this array.

Return : int

The size of this array

Example

```
M_IrNoteOut mOut;
for (i = 0; i < mOut.maIrNotePrice.GetSize(); i++) {
  ...
}
```

M_{msgname} *GetAt(int p_iIdx)

Description

This method is used to get the pointer to message instance from this array by the given index.

Return : M_{msgname} *

The pointer to message class

Example

```
M_IrNoteOut mOut;
for (i = 0; i < mOut.maIrNotePrice.GetSize(); i++) {
  M_IrNotePrice *p = mOut.maIrNotePrice.GetAt(i);
}
```

M_{msgname} &operator[](int p_iIdx)

Description

This method is used to get the message instance from this array by the given index.

Return : M_{msgname} &

The message instance

Example

```
M_IrNoteOut mOut;
for (i = 0; i < mOut.maIrNotePrice.GetSize(); i++) {
  double d1 = mOut.maIrNotePrice[i].dDirtyPrice;
  double d2 = mOut.maIrNotePrice[i].dCleanPrice;
}
```

void Copy(M_{msgname}_Array *)

Description

This method is used to copy from the given parameter to this instance. This executes deep-copy.

Example

```
M_IrVanilla_Array maIr;
...

M_IrVanilla maIr2;
maIr2.Copy(&maIr);
```

void Dump(CStream *p_pBuffer)

Description

This method is used to dump the contents of this instance to string buffer. This is used to debug usually.

Example

```
M_IrVanilla mIr;
...
CStream aBuff;
mIr.Dump(&aBuff);
printf("%s\n", aBuff.GetData());
```

The End.