“C+ + ” SAMPLE CODE

This chapter provides sample “C++” code.

Structure definitions

```
#pragma pack(1)

const long lPM_SERVER_ALIVE = 3;
const long lPM_REQUEST_RETRANSMIT = 4;
const long lPM_VALUE_ATTRIBUTE_UPDATE = 6;
const USHORT RID_VALUE_UPDATE = 11;
const USHORT RID_ATTRIBUTE_UPDATE = 12;
const USHORT RID_END = 0xdead;

//------------------------------
//   ParamMsgHdrDef
//------------------------------
typedef struct
{
   unsigned short   usCurrentMsg;
   unsigned short   usTotalMsg;
   long             lSeqNo;
   long             lMsgID;

} ParamMsgHdrDef;

//------------------------------
//   RecHdrDef
//------------------------------
typedef struct
{
   USHORT   usRecID;
   USHORT   usRecLen;
} RecHdrDef;

//------------------------------
//   STRDef
//------------------------------
typedef struct
{
   long   lStrLen;
   char   aStr[1];

   long sizeOf() const
   {
      return sizeof(lStrLen) + lStrLen;
   }
   long addressOfNext() const
   {
      return ((long)&aStr[0]) + lStrLen;
   }
void    copyTo(BSTR& bs) const
   {
   USES_CONVERSION;
   bs = ::SysAllocStringLen(A2CW(&aStr[0]),
         lStrLen);
   }
```

```
} STRDef;

//------------------------------
//   ValueDef
//------------------------------
typedef struct
{
   long   lValueLen;
   BYTE   aValue[1];

   long   sizeOf() const
   {
      return sizeof(lValueLen) + lValueLen;
   }
   long   addressOfNext() const
   {
      return ((long)&aStr[0]) + lValueLen;
   }
} ValueDef;

//------------------------------
//   AttribDef
//------------------------------
typedef struct
{
   long    lCookie;
   STRDef  sAttributeName;
   long    lAttributeLen;
   BYTE    aAttribute[1];

   long    getAttribLen() const
   {
   return (long)*(long*)sAttributeName.addressOfNext();
   }
   BYTE*   getAttrib() const
   {
   return (BYTE*)(sAttributeName.addressOfNext()
   + sizeof(lAttributeLen));
   }
   long   sizeOf() const
   {
   return sizeof(AttribDef)
      - sizeof(STRDef)
      -1
      + sAttributeName.sizeOf()
      + getAttribLen();
   }
   long   addressOfNext() const
   {
   return (long)(getAttrib()
      + getAttribLen());
   }
} AttribDef;

//------------------------------
//   AttributesRecDef
//------------------------------
typedef struct
```

```
{
   RecHdrDef   hdr;
   Long        lAttribCount;
   AttribDef   aAttribs[1];

long sizeOf() const
{
   if (lAttribCount)
   {
   const AttribDef* pa = &aAttribs[0];
   long l(0);
   for (int i = 0; i < lAttribCount; i++)
   {
   l += pa->sizeOf();
   pa = (AttribDef*) pa->addressOfNext();
   }
   return l + sizeof(lAttribCount)
      + sizeof(RecHdrDef);
   }
   else
   return sizeof(lAttribCount)
      + sizeof(RecHdrDef);
   }
   long addressOfNext() const
   {
   return ((long)&hdr + sizeOf());
   }
} AttributesRecDef;

//------------------------------
//   ParamDef
//------------------------------
typedef struct
{
   long      lCookie;
   DWORDLONG dwlTimeStamp;
   ValueDef  sValue;
   long      unused;

   long      sizeOf() const
   {
   long l;
   l = sizeof(ParamDef);
   l -= sizeof(ValueDef);
   l += sValue.sizeOf();
   return l;
   }
   long   addressOfNext() const
   {
   return (long) &lCookie + sizeOf();
   }
} ParamDef;

//------------------------------
//   ParametersRecDef
//------------------------------
typedef struct
{
```

```
   RecHdrDef   hdr;
   long        lParamCount;
   ParamDef    aParams[1];

   long sizeOf() const
   {
   if (lParamCount)
   {
   const ParamDef* pp = &aParams[0];
   long l(0);
   for (int i = 0; i < lParamCount; i++)
   {
      l += pp->sizeOf();
      pp = (ParamDef*) pp->addressOfNext();
   }
   return l + sizeof(lParamCount) + sizeof(RecHdrDef);
   }
   else
   return sizeof(lParamCount) + sizeof(RecHdrDef);
   }
   long   addressOfNext() const
   {
   return ((long)&hdr + sizeOf());
   }
} ParametersRecDef;

//------------------------------
//   AliveMessageDef
//------------------------------
typedef struct
{
   ParamMsgHdrDef sHdr;
} AliveMessageDef;

//------------------------------
//   RequestRetransmitMessageDef
//------------------------------
typedef struct
{
   ParamMsgHdrDef sHdr;
} RequestRetransmitMessageDef;

#pragma pack()
```

Handling parameter notifications

```
//------------------------------
//   DataReceived
//------------------------------
int DataReceived (INT nLength, BYTE pData[])
{
   ParamMsgHdrDef* pHdr = (ParamMsgHdrDef*) pData;
   if (pHdr->lMsgID == lPM_VALUE_ATTRIBUTE_UPDATE)
   {
   if (!CheckSequenceNumber(pHdr))
   return 0;
```

```
   RecHdrDef* recHdr = (RecHdrDef*) (pData +
   sizeof(ParamMsgHdrDef));

      while (recHdr->usRecID != RID_END)
      {
      switch (recHdr->usRecID)
      {
         case RID_VALUE_UPDATE:
         {
         ParametersRecDef* ppr =
         (ParametersRecDef*) recHdr;
         HandleValueUpdate(ppr);
         }
         break;
         case RID_ATTRIBUTE_UPDATE:
         {
            AttributesRecDef* par =
            (AttributesRecDef*) recHdr;
            HandleAttributeUpdate(par);
         }
         break;
      }
      recHdr = (RecHdrDef*) ((long)(recHdr) + recHdr->usRecLen);
      }

      m_lNextSequenceNo++;
      m_lNextMsgNo = 1;
   }
   else if (pHdr->lMsgID == lPM_SERVER_ALIVE)
   {
      if (pHdr->lSeqNo != m_lNextSequenceNo)
      SendRetransmitRequest();
   }
   return S_OK;
}

//------------------------------
//   HandleValueUpdate
//------------------------------
void HandleValueUpdate(ParametersRecDef* pPM)
{
   ParamDef* pP = &pPM>aParams[0];

   for (int i = 0; (i < pPM->lParamCount); i++)
   {
   CParameter* parameter = FindParameter(pP->lCookie);
      if (parameter != NULL)
      {
         parameter->UpdateValue(pp);
      }
      pP = (ParamDef*) pP->addressOfNext();
   }
}

//------------------------------
//   HandleAttributeUpdate
//------------------------------
void HandleAttributeUpdate(AttributesRecDef* pAD)
```

```
{
   AttribDef* pA = &pAD>aAttribs[0];

   for (int i = 0; (i < pAD->lAttribCount); i++)
   {
      CParameter* parameter = FindParameter(pA->lCookie);
      if (parameter != null)
      {
   parameter->UpdateAttribute(pA);
      }
      pA = (AttribDef*) pA->addressOfNext();
      }
}
```