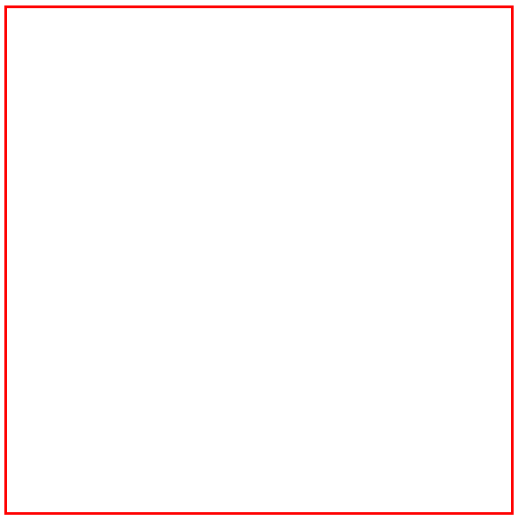
<



Microsoft PowerPoint 97 File Format SDK

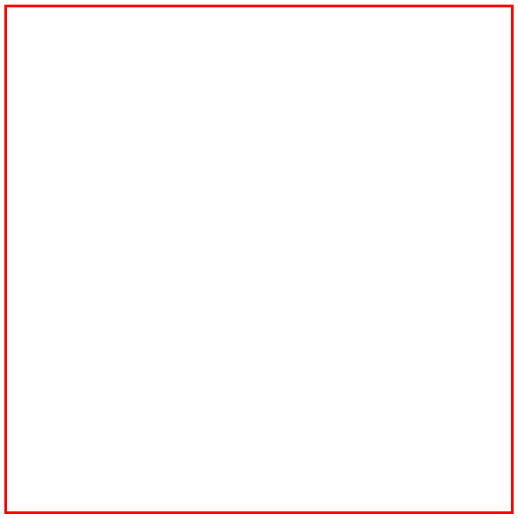
Graphics Product Unit

Anomolies:

(????) msofbtClientData

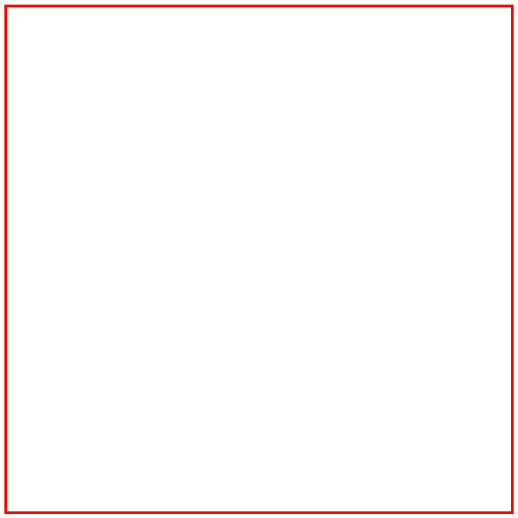
Alphabetically:

By record type:



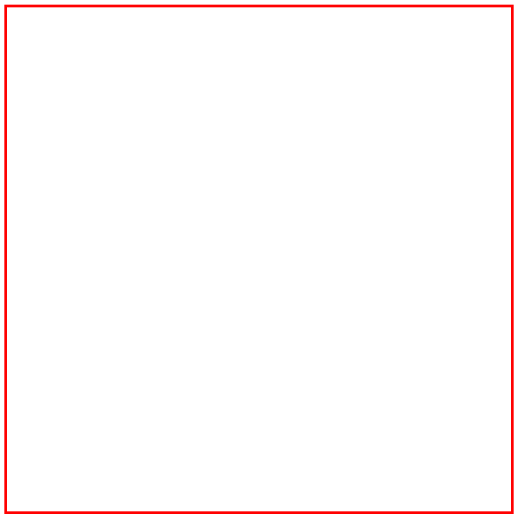
Introduction

Microsoft PowerPoint for Windows 97 uses OLE 2 compound files; this is the OLE implementation of the Structured Storage Model standard. An OLE 2 compound file is "a file system within a file"; it contains a hierarchical system of storages and streams. A storage is analogous to a directory because it holds other storages and streams, and a stream is analogous to a file because it holds information but no other storage elements. For more information on this technology, please refer to the *OLE 2 Programmer's Reference Volume one*, and *Inside OLE*, both published by Microsoft Press.



Purpose and Scope

This document describes the PowerPoint 97 file format, and it is intended for use by developers of applications that interact with PowerPoint files. This document is a programming and technical reference. It assumes familiarity with both PowerPoint and a high level programming language like C, C++ or Visual Basic.



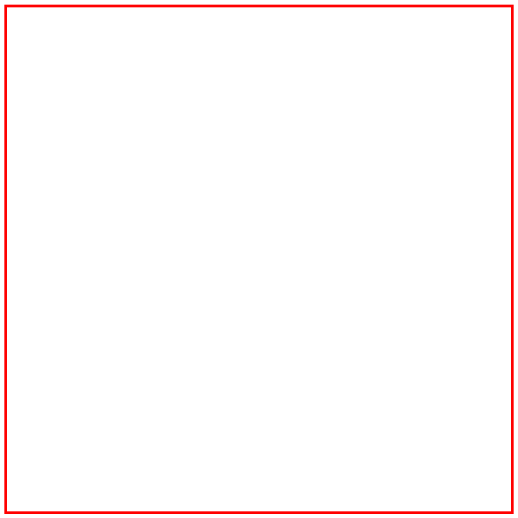
Vocabulary

Collections: Sets of objects. Objects within the set are referenced by their index in the set.

External objects: Objects that can be brought into PowerPoint using the Insert Object dialog. This includes pictures, sounds, movies, etc.

Master Coordinates: The reference system used by PowerPoint to put all objects on the screen. The origin for the system is the center of the slide. There are two axes, X (horizontal) and Y (vertical). Values on the X axis increase when you move to the right and the origin is 0. Values on the Y axis increase when moving down. Master coordinates are always 576 dpi.

View: Refers to the way a presentation is seen on the screen at a particular moment. This includes the current view, whether the guides or rulers are visible, and the view scale.



Abbreviations

The following abbreviations are used throughout the document:

BOOL1: Boolean one-byte value.

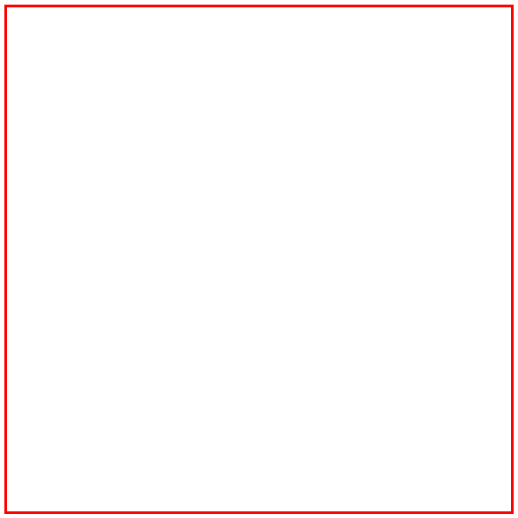
UBYTE: Unsigned one-byte value.

UINT2: Unsigned two-byte integer value.

UINT4: Unsigned four-byte integer value.

SINT2: Signed two-byte integer value.

SINT4: Signed four-byte integer value.



File Format Overview

PowerPoint 97 files are OLE DocObject files consisting of the following streams:

Current User - Keeps the name of the user who last opened the presentation.

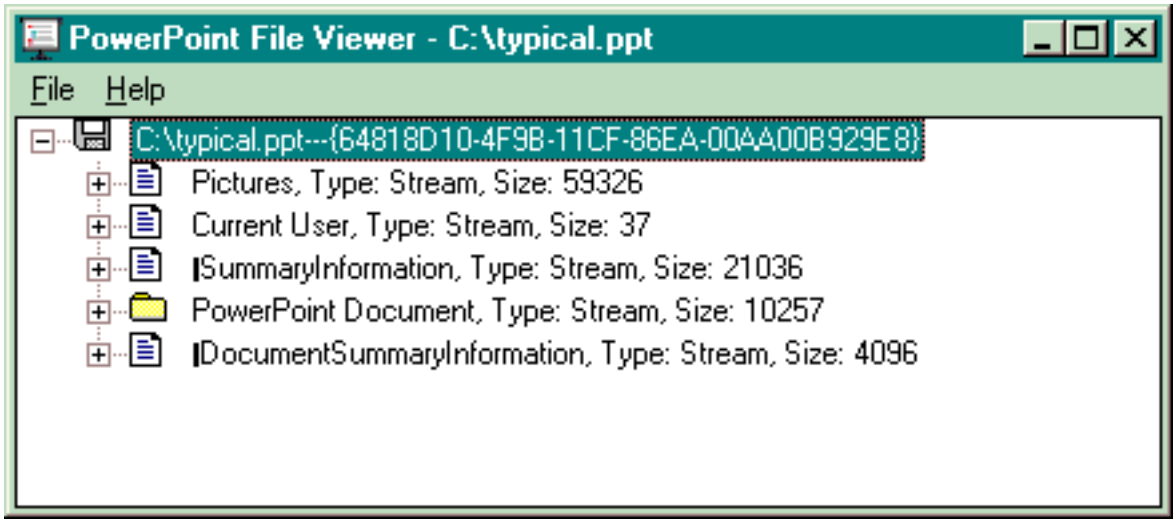
PowerPoint Document - Keeps all of the information about a PowerPoint presentation. This document explains its layout and contents.

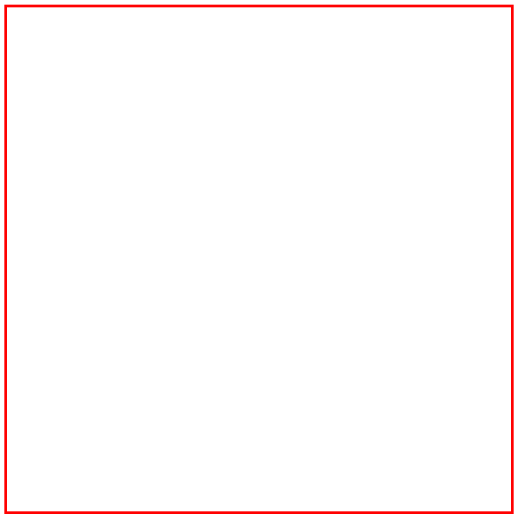
Pictures (Optional)– Contains data about the pictures (metafiles, PNG, JPG, etc) contained in a PowerPoint presentation.

Summary Information and DocumentSummaryInformation (Optional) - Keeps statistics about the document, following a Microsoft Office standard. For more information about these streams refer to the Microsoft Word 97 Binary File Format document.

The storage elements in PowerPoint files can be viewed with the **PowerPoint File View application** that accompanies this document. The following is a screen shot of the file viewer showing the contents of a PowerPoint 97 file.

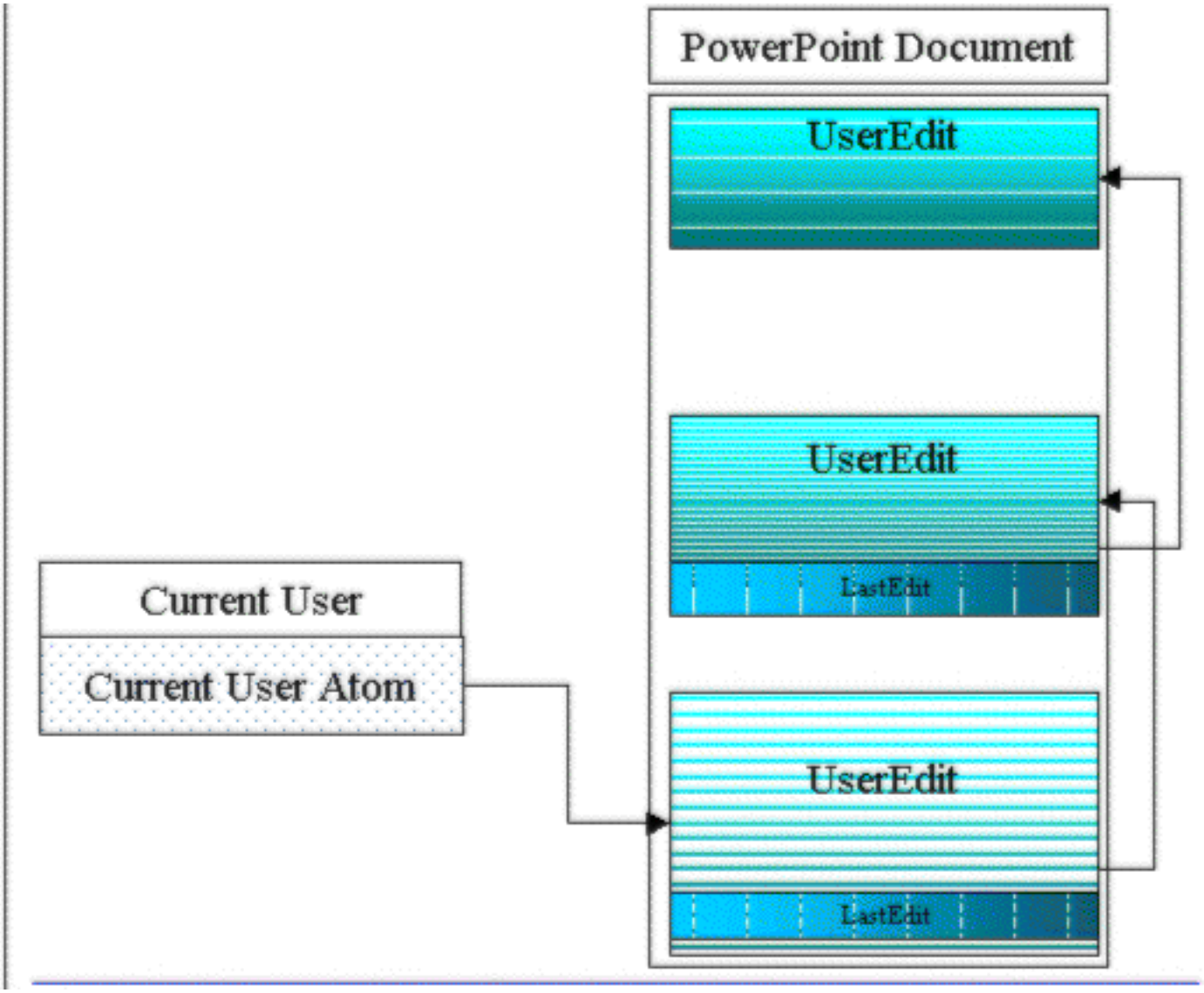
Contents of a PowerPoint 97 file:

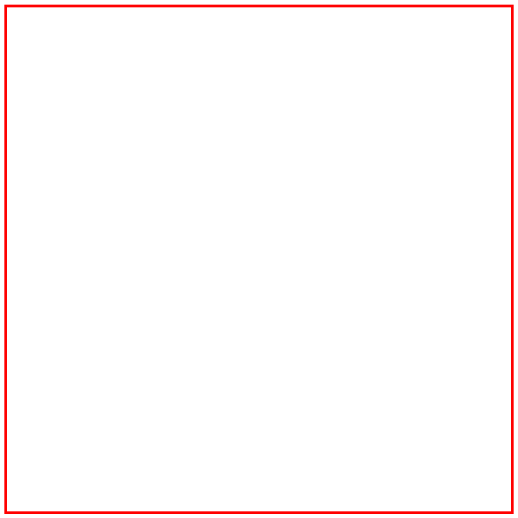

PowerPoint File Viewer - C:\typical.ppt
File
Help
C:\typical.ppt---{64818D10-4F9B-11CF-86EA-00AA00B929E8}
Pictures, Type: Stream, Size: 59326
Current User, Type: Stream, Size: 37
SummaryInformation, Type: Stream, Size: 21036
PowerPoint Document, Type: Stream, Size: 10257
DocumentSummaryInformation, Type: Stream, Size: 4096



Current User Stream

The Current User Stream contains a pointer to the latest saved edit in the document stream. The document stream contains one or more user edit structures. A graphical representation of this looks like:


PowerPoint Document
UserEdit
UserEdit
LastEdit
UserEdit
LastEdit
Current User
Current User Atom



UserEditAtom Structure

The UserEditAtom structure is as follows:

struct PSR_UserEditAtom

{

sint4 lastSlideID; // slideID of last viewed slide

uint4 version; // This is major/minor/build which did the edit

uint4 offsetLastEdit; // File offset of last edit

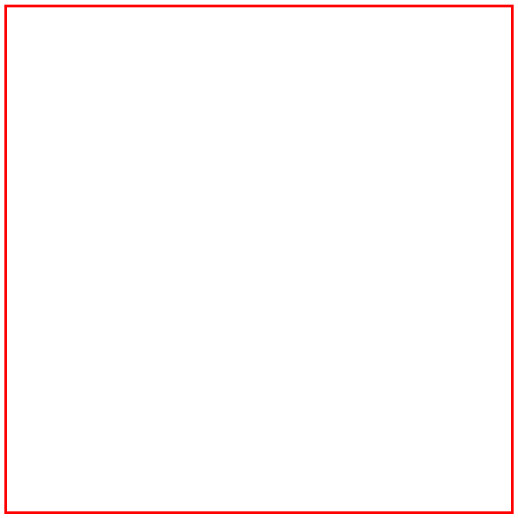
uint4 offsetPersistDirectory; // Offset to PersistPtrs for this edit.

uint4 documentRef; // reference to document atom

uint4 maxPersistWritten; // Addr of last persist ref written to the file (max seen so far).

sint2 lastViewType; // enum view type

};



UserEditAtom Element Descriptions

lastSlideID and lastViewType: SlideID of last slide viewed and view type for saved view, respectively. Allow a document window to be opened in its saved configuration.

version: Major/minor/build which did the edit.

offsetLastEdit: Pointer to the last user edit. This is a 32 bit fixed offset from the beginning of the file. (This is 0 if no previous edits exist. It is illegal to place a LastEdit structure at offset 0 in the file.)

offsetPersistDirectory: Contains the persistent references (32 bit offset from the beginning of the document stream) in the current user edit. References are number sequentially from 1 (0 is not a valid value) and each user edit will contain a persistent directory. This directory contains only the references made by the current user and the document data included in the edit. To find additional references, PowerPoint begins with the directory of the last edit and then searches recursively through the previous edits until the reference is found.

The persistent directory is encoded as follows:

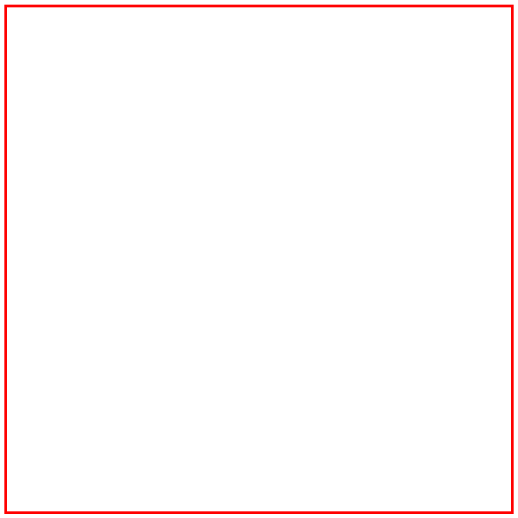
12 bit value which is 20 bit value indicates current reference number

number of sequential

offsets

documentRef: Reverence to the document atom.

MaxPersistWritten: Address of the last persist ref written to the file. This is the maximum value contained in the file, maintained so that new user edits can be properly numbered.



Persistent Directory Example

Suppose the current save of a PowerPoint document contains the following:

Reference File Offset

8 10000

The following would be saved to the file:

Hex Decimal Meaning

1772 6002 PST_PersistPtrIncrementalBlock

24 36 Length of Atom

300001 3145729 3 consecutive offsets starting at 1

400 1024 Offset to ref(1)

800 2048 Offset to ref(2)

1000 4096 Offset to ref(3)

100006 1048582 1 consecutive refs starting at 6

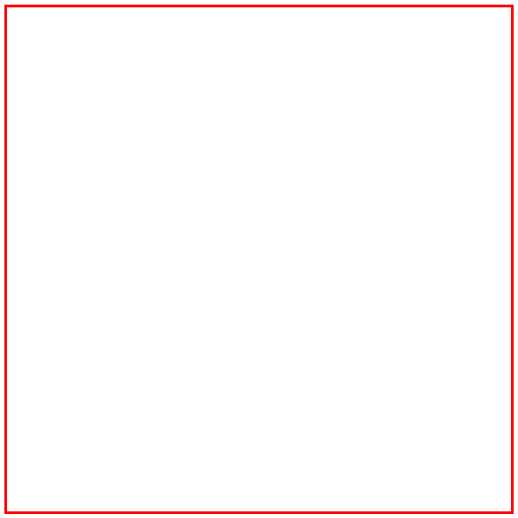
2000 8192 Offset to ref(6)

200008 2087160 2 consecutive refs starting at 8

2710 10000 Offset to ref(8)

4E20 20000 Offset to ref(9)

For an example of an application that tracks user edits see appendix B.



PowerPoint Document Stream

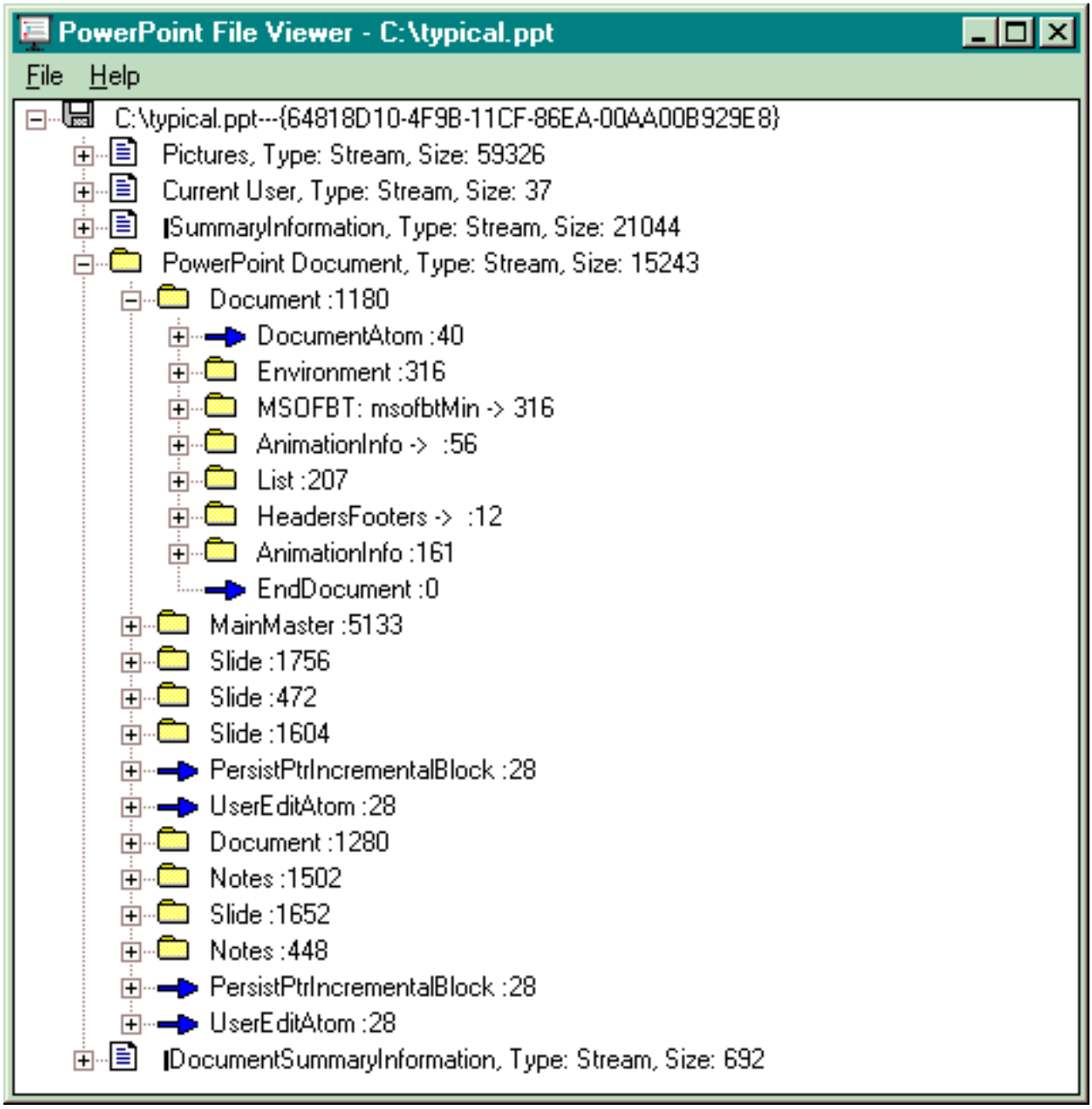
The PowerPoint Document Stream keeps all the information about a PowerPoint presentation. A PowerPoint file stores its data in records, all of which are defined in the accompanying C Header file "Serial.h". There are two different kinds of records in a file: atoms and containers. We could, as with storages and streams, compare atoms and containers to files and directories, respectively. Atoms, like files, keep the actual information. Containers, just like directories, can contain files and other directories.

Atoms: Records that contain information about a PowerPoint object and are kept inside containers.

Containers: Records that keep atoms and other containers in a logical and organized way.

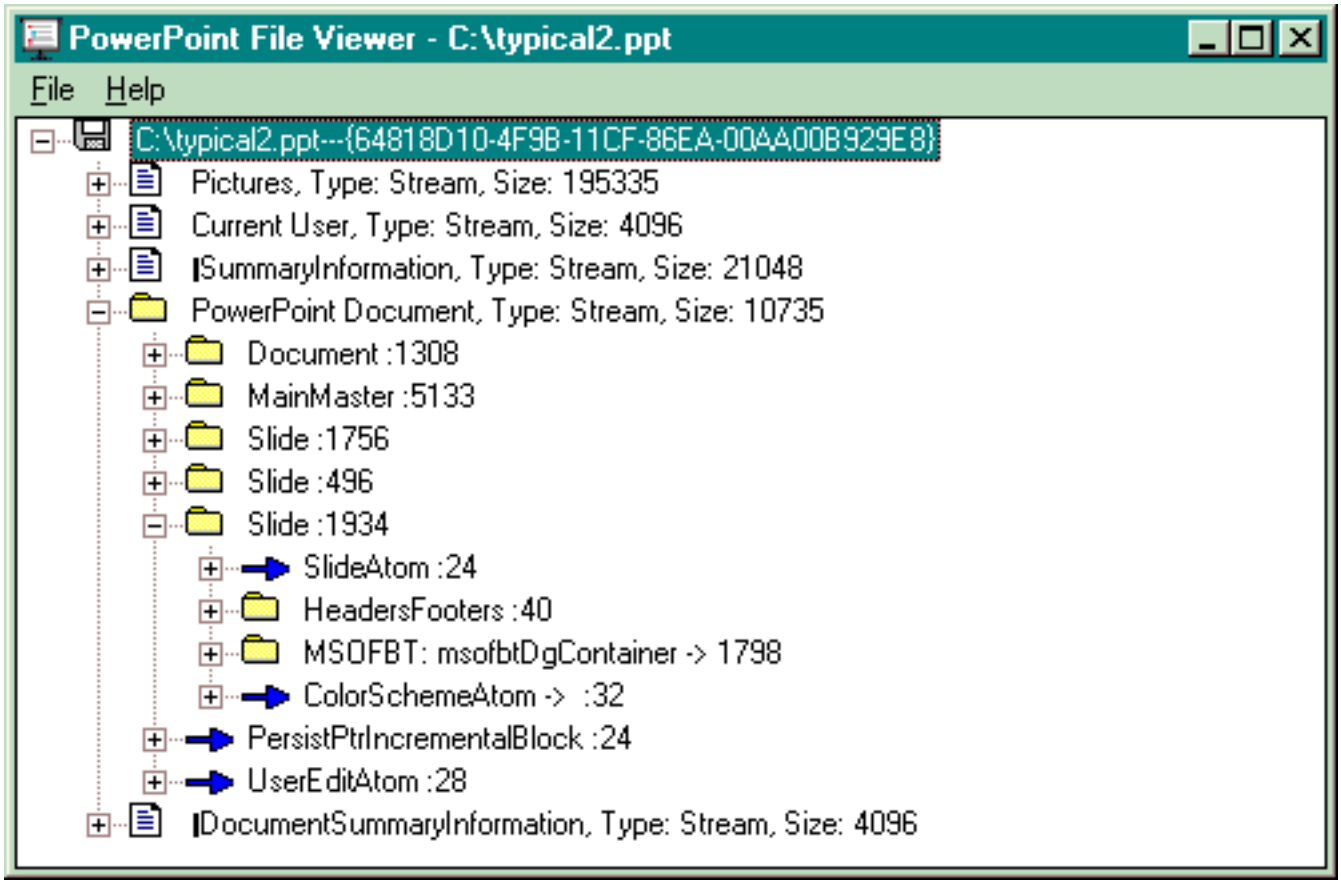
Again, the PowerPoint File Viewer will allow you to view the Document Steam.

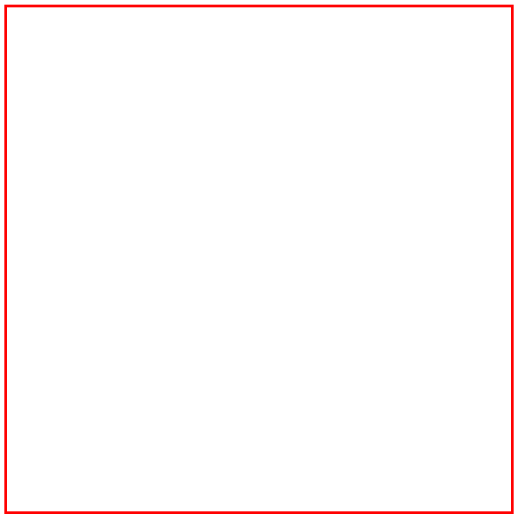
Contents of a PowerPoint 97 Document Stream


PowerPoint File Viewer - C:\typical.ppt
File Help
C:\typical.ppt---{64818D10-4F9B-11CF-86EA-00AA00B929E8}
Pictures, Type: Stream, Size: 59326
Current User, Type: Stream, Size: 37
SummaryInformation, Type: Stream, Size: 21044
PowerPoint Document, Type: Stream, Size: 15243
Document :1180
DocumentAtom :40
Environment :316
MSOFBT: msofbtMin -> 316
AnimationInfo -> :56
List :207
HeadersFooters -> :12
AnimationInfo :161
EndDocument :0
MainMaster :5133
Slide :1756
Slide :472
Slide :1604
PersistPtrIncrementalBlock :28
UserEditAtom :28
Document :1280
Notes :1502
Slide :1652
Notes :448
PersistPtrIncrementalBlock :28
UserEditAtom :28
DocumentSummaryInformation, Type: Stream, Size: 692

A Slide

A typical PowerPoint file will have Slide containers. A Slide container keeps all the atoms and containers necessary to describe a single PowerPoint slide. The following screen shot provides a closer look at what's inside a Slide container:





Appendix A: Records Ordered by Number

Name	*Type*
Unknown	0
SubContainerCompleted	1
IRRAtom	2
PSS	3
SubContainerException	4
ClientSignal1	6
ClientSignal2	7
PowerPointStateInfoAtom	10
Document	1000
DocumentAtom	1001
EndDocument	1002

SlidePersist	1003
SlideBase	1004
SlideBaseAtom	1005
Slide	1006
SlideAtom	1007
Notes	1008
NotesAtom	1009
Environment	1010
SlidePersistAtom	1011
Scheme	1012
SchemeAtom	1013
DocViewInfo	1014
SslideLayoutAtom	1015
MainMaster	1016
SSSlideInfoAtom	1017
SlideViewInfo	1018
GuideAtom	1019
ViewInfo	1020
ViewInfoAtom	1021
SlideViewInfoAtom	1022

VBAInfo	1023
VBAInfoAtom	1024
SSDocInfoAtom	1025
Summary	1026
Texture	1027
VBASlideInfo	1028
VBASlideInfoAtom	1029
DocRoutingSlip	1030
OutlineViewInfo	1031
SorterViewInfo	1032
ExObjList	1033
ExObjListAtom	1034
PPDrawingGroup	1035
PPDrawing	1036
NamedShows	1040
NamedShow	1041
NamedShowSlides	1042
List	2000
FontCollection	2005
ListPlaceholder	2017

BookmarkCollection	2019
SoundCollection	2020
SoundCollAtom	2021
Sound	2022
SoundData	2023
BookmarkSeedAtom	2025
GuideList	2026
RunArray	2028
RunArrayAtom	2029
ArrayElementAtom	2030
Int4ArrayAtom	2031
ColorSchemeAtom	2032
OEShape	3008
ExObjRefAtom	3009
OEPlaceholderAtom	3011
GrColor	3020
GrectAtom	3025
GratioAtom	3031
Gscaling	3032
GpointAtom	3034

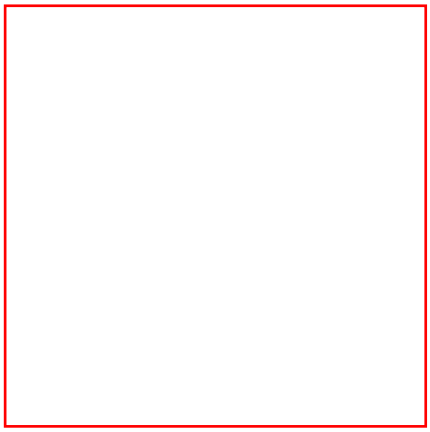
OEShapeAtom	3035
OutlineTextRefAtom	3998
TextHeaderAtom	3999
TextCharsAtom	4000
StyleTextPropAtom	4001
BaseTextPropAtom	4002
TxMasterStyleAtom	4003
TxCFStyleAtom	4004
TxPFStyleAtom	4005
TextRulerAtom	4006
TextBookmarkAtom	4007
TextBytesAtom	4008
TxSIStyleAtom	4009
TextSpecInfoAtom	4010
DefaultRulerAtom	4011
FontEntityAtom	4023
FontEmbedData	4024
TypeFace	4025
CString	4026
ExternalObject	4027

MetaFile	4033
ExOleObj	4034
ExOleObjAtom	4035
ExPlainLinkAtom	4036
CorePict	4037
CorePictAtom	4038
ExPlainAtom	4039
SrKinsoku	4040
Handout	4041
ExEmbed	4044
ExEmbedAtom	4045
ExLink	4046
ExLinkAtom_old	4047
BookmarkEntityAtom	4048
ExLinkAtom	4049
SrKinsokuAtom	4050
ExHyperlinkAtom	4051
ExPlain	4053
ExPlainLink	4054
ExHyperlink	4055

SlideNumberMCAtom	4056
HeadersFooters	4057
HeadersFootersAtom	4058
RecolorEntryAtom	4062
TxInteractiveInfoAtom	4063
EmFormatAtom	4065
CharFormatAtom	4066
ParaFormatAtom	4067
MasterText	4068
RecolorInfoAtom	4071
ExQuickTime	4073
ExQuickTimeMovie	4074
ExQuickTimeMovieData	4075
ExSubscription	4076
ExSubscriptionSection	4077
ExControl	4078
ExControlAtom	4091
SlideListWithText	4080
AnimationInfoAtom	4081
InteractiveInfo	4082

InteractiveInfoAtom	4083
SlideList	4084
UserEditAtom	4085
CurrentUserAtom	4086
DateTimeMCAtom	4087
GenericDateMCAtom	4088
HeaderMCAtom	4089
FooterMCAtom	4090
ExMediaAtom	4100
ExVideo	4101
ExAviMovie	4102
ExMCIMovie	4103
ExMIDIAudio	4109
ExCDAudio	4110
ExWAVAudioEmbedded	4111
ExWAVAudioLink	4112
ExOleObjStg	4113
ExCDAudioAtom	4114
ExWAVAudioEmbeddedAtom	4115
AnimationInfo	4116

RTFDateTimeMCAtom	4117
ProgTags	5000
ProgStringTag	5001
ProgBinaryTag	5002
BinaryTagData	5003
PrintOptions	6000
PersistPtrFullBlock	6001
PersistPtrIncrementalBlock	6002
RulerIndentAtom	10000
GscalingAtom	10001
GrColorAtom	10002
GLPointAtom	10003
GlineAtom	10004



Appendix B

```
//
// Sample code to read the text out of a PowerPoint '97 presentation.
//
#include <ole2.h>
#include <stdio.h>
#include <time.h>

// Stolen from app\sertypes.h
// system dependent sizes
// system dependent sizes
//
typedef signed long sint4; // signed 4-byte integral value
typedef signed short sint2; // signed 4-byte integral value
typedef unsigned long uint4; // unsigned 4-byte integral value
typedef unsigned short uint2; // 2-byte
typedef char bool1; // 1-byte boolean
typedef unsigned char ubyte1; // unsigned byte value
typedef uint2 psrType;
typedef uint4 psrSize; // each record is preceeded by
// pssTypeType and pssSizeType.
typedef uint2 psrInstance;
typedef uint2 psrVersion;
typedef uint4 psrReference; // Saved object reference
#define PSFLAG_CONTAINER 0xFF // If the version field of a record
                              // header takes on this value, the
                              // record header marks the start of
                              // a container.
// PowerPoint97 Record Header
typedef unsigned long DWord;
```

```
int AssertionFailed( const char* file, int line, const char* expr )
/*=================*/
{
   // AR: Message box the assert
   return( TRUE );
} /* AssertionFailed */

#define Assert( expr ) \
{ \
static char _str[] = #expr; \
\
if( !(int)(expr) ) \
AssertionFailed( __FILE__, __LINE__, _str ); \
} /* Assert */

static BOOL ReadText( WCHAR* buffer, unsigned long bufferSize, unsigned
long* pSizeRet );
// Returns TRUE if more text exists. Fills buffer upto bufferSize. Actual
size used is
// pSizeRet.

struct RecordHeader
{
   psrVersion recVer : 4; // may be PSFLAG_CONTAINER
   psrInstance recInstance : 12;
   psrType recType;
   psrSize recLen;
};

struct PSR_CurrentUserAtom
{
   uint4 size;
   uint4 magic; // Magic number to ensure this is a PowerPoint file.
   uint4 offsetToCurrentEdit; // Offset in main stream to current edit
field.
   uint2 lenUserName;
   uint2 docFileVersion;
   ubyte1 majorVersion;
   ubyte1 minorVersion;
};

struct PSR_UserEditAtom
{
   sint4 lastSlideID; // slideID
   uint4 version; // This is major/minor/build which did the edit
```

```
   uint4 offsetLastEdit; // File offset of last edit
   uint4 offsetPersistDirectory; // Offset to PersistPtrs for
                                 // this file version.
   uint4 documentRef;
   uint4 maxPersistWritten; // Addr of last persist ref written to the
file (max seen so far).
   sint2 lastViewType; // enum view type
};

struct PSR_SlidePersistAtom
{
   uint4 psrReference;
   uint4 flags;
   sint4 numberTexts;
   sint4 slideId;
   uint4 reserved;
};

#define CURRENT_USER_STREAM L"Current User"
#define DOCUMENT_STREAM L"PowerPoint Document"
#define HEADER_MAGIC_NUM -476987297

const int PST_UserEditAtom = 4085;
const int PST_PersistPtrIncrementalBlock = 6002; // Incremental diffs on
persists
const int PST_SlidePersistAtom = 1011;
const int PST_TextCharsAtom = 4000; // Unicode in text
const int PST_TextBytesAtom = 4008; // non-unicode text
class PPSPersistDirectory;

struct ParseContext
{
   ParseContext(ParseContext *pNext) : m_pNext(pNext), m_nCur(0)
   {
   }
   RecordHeader m_rh;
   uint4 m_nCur;
   ParseContext *m_pNext;
};

const int SLIDELISTCHUNKSIZE=32;
struct SlideListChunk
{
   SlideListChunk( SlideListChunk* next, psrReference newOne ) :
   pNext( next ), numInChunk(1)
```

```
   {
      refs[0] = newOne;
   }
   SlideListChunk *pNext;
   DWord numInChunk;
   psrReference refs[SLIDELISTCHUNKSIZE];
};

class FileReader
{
public:
   FileReader(IStorage *pStg);
   ~FileReader();
   BOOL ReadText( WCHAR *pBuff, ULONG size, ULONG *pSizeRet );
// Reads next size chars from file. Returns TRUE if there is more
// text to read.
   BOOL IsPowerPoint()
   {
      return m_isPP;
   } // Returns true if this is a PowerPoint '97 file.
   void ReadPersistDirectory();
   void PPSReadUserEditAtom( DWord offset, PSR_UserEditAtom& userEdit );
   void ReadSlideList();
protected:
   BOOL ReadCurrentUser(IStream *pStm);
   void *ReadRecord( RecordHeader& rh );
   BOOL Parse();
   IStream *GetDocStream();
   BOOL DoesClientRead( psrType type )
   {
      return FALSE;
   }
   void ReleaseRecord( RecordHeader& rh, void* diskRecBuf );
   DWord ParseForSlideLists();
   void AddSlideToList( psrReference refToAdd );
   BOOL StartParse( DWord offset );
   BOOL FillBufferWithText();
   BOOL FindNextSlide( DWord& offset );
private:
   PSR_CurrentUserAtom m_currentUser;
   IStream * m_pDocStream;
   IStorage * m_pPowerPointStg;
   BOOL m_isPP;
   ParseContext* m_pParseContexts;
   WCHAR* m_pCurText;
```

```
   unsigned long m_curTextPos;
   unsigned long m_curTextLength;
   PSR_UserEditAtom* m_pLastUserEdit;
   PPSPersistDirectory* m_pPersistDirectory;
   SlideListChunk* m_pFirstChunk;
   int m_curSlideNum;
   WCHAR* m_pClientBuf;
   unsigned long m_clientBufSize;
   unsigned long m_clientBufPos;
   ULONG* m_pSizeRet;
};

FileReader::FileReader(IStorage *pStg) :
m_pPowerPointStg(pStg),
m_isPP(FALSE),
m_pParseContexts(NULL),
m_curTextPos(0),
m_pLastUserEdit( NULL ),
m_pPersistDirectory( NULL ),
m_pDocStream( NULL ),
m_pFirstChunk( NULL ),
m_curSlideNum(0),
m_pCurText( NULL ),
m_pClientBuf( NULL ),
m_clientBufSize( 0 ),
m_clientBufPos( 0 )
{
   IStream *pStm = NULL;
   m_pPowerPointStg->AddRef();
   HRESULT hr = pStg->OpenStream( CURRENT_USER_STREAM, NULL, STGM_READ |
STGM_DIRECT | STGM_SHARE_EXCLUSIVE, NULL, &pStm );
   if ( SUCCEEDED(hr) && ReadCurrentUser(pStm) )
      m_isPP = TRUE;
   pStm->Release();
}

FileReader::~FileReader()
{
   m_pPowerPointStg->Release();
}

BOOL FileReader::FillBufferWithText()
{
   unsigned long amtToCopy = min( (m_curTextLength - m_curTextPos),
(m_clientBufSize - m_clientBufPos) );
```

```
   unsigned long loop = amtToCopy;
   while ( loop-- )
      m_pClientBuf[ m_clientBufPos++ ] = m_pCurText[ m_curTextPos++ ];
   if ( m_curTextPos == m_curTextLength )
   {
      delete [] m_pCurText;
      m_pCurText = NULL;
      m_curTextPos = 0;
      m_curTextLength = 0;
   }
   *m_pSizeRet += amtToCopy;
   return(m_clientBufSize == m_clientBufPos); // If client's buffer is
full return TRUE.
}

void FileReader::AddSlideToList( psrReference refToAdd )
{
   if ( m_pFirstChunk == NULL )
      m_pFirstChunk = new SlideListChunk(NULL, refToAdd);
   else
   {
      if ( m_pFirstChunk->numInChunk+1 > SLIDELISTCHUNKSIZE )
         m_pFirstChunk = new SlideListChunk(m_pFirstChunk, refToAdd);
      else
      {
         m_pFirstChunk->refs[m_pFirstChunk->numInChunk] = refToAdd;
         m_pFirstChunk->numInChunk++;
      }
   }
}

IStream *FileReader::GetDocStream()
{
   if ( m_pDocStream == NULL )
   {
      if ( !m_isPP )
         return NULL;
      HRESULT hr = m_pPowerPointStg->OpenStream( DOCUMENT_STREAM, NULL,
STGM_READ | STGM_DIRECT | STGM_SHARE_EXCLUSIVE, NULL, &m_pDocStream );
      if (FAILED(hr))
      {
         fprintf(stderr,"Error (%d) opening PowerPoint Document Stream.
\n",(int)hr);
         return NULL;
      }
```

```
   }
   return m_pDocStream;
}

BOOL FileReader::ReadCurrentUser(IStream *pStm)
{
   ULONG nRd=0;
   RecordHeader rh;
   BOOL isPP = FALSE;
   if ( SUCCEEDED( pStm->Read(&rh, sizeof(rh), &nRd) ) )
   {
      if ( SUCCEEDED( pStm->Read(&m_currentUser, sizeof
(PSR_CurrentUserAtom), &nRd) ) )
      {
         if ( nRd != sizeof(PSR_CurrentUserAtom) )
            return FALSE;
      }
      isPP = ( m_currentUser.size == sizeof( m_currentUser ) )&&
             ( m_currentUser.magic == HEADER_MAGIC_NUM )&&
             ( m_currentUser.lenUserName <= 255 );
   }
   return isPP;
}

class PPSDirEntry
{
   PPSDirEntry()
   : m_pNext( NULL ), m_pOffsets( NULL ), m_tableSize( 0 )
   {
   }
   PPSDirEntry* m_pNext;
   DWord* m_pOffsets;
   DWord m_tableSize;
public:
   ~PPSDirEntry()
   {
      delete m_pOffsets; m_pOffsets = NULL;
   }
   friend class PPSPersistDirectory;
};

// class PPSDirEntry
class PPSPersistDirectory
{
public:
```

```
   PPSPersistDirectory();
   ~PPSPersistDirectory();
   void AddEntry( DWord cOffsets, DWord* pOffsets );
   DWord GetPersistObjStreamPos( DWord ref );
   DWord NumberOfAlreadySavedPersists();
private:
   PPSDirEntry* m_pFirstDirEntry;
};

PPSPersistDirectory::PPSPersistDirectory() : m_pFirstDirEntry( NULL )
{
}

PPSPersistDirectory::~PPSPersistDirectory()
{
   while ( m_pFirstDirEntry )
   {
      PPSDirEntry* pDirEntry = m_pFirstDirEntry;
      m_pFirstDirEntry = m_pFirstDirEntry->m_pNext;
      delete pDirEntry;
   }
}

void PPSPersistDirectory::AddEntry( DWord cOffsets, DWord* pOffsets )
{
   PPSDirEntry* pDirEntry = new PPSDirEntry();
   pDirEntry->m_tableSize = cOffsets;
   pDirEntry->m_pOffsets = new DWord[cOffsets];
   memcpy( pDirEntry->m_pOffsets, pOffsets, cOffsets * sizeof( DWord ) );
// append to the end of the entry list
   PPSDirEntry** ppDirEntry = &m_pFirstDirEntry;
   while ( NULL != *ppDirEntry )
      ppDirEntry = &(*ppDirEntry)->m_pNext;
   *ppDirEntry = pDirEntry;
}

DWord PPSPersistDirectory::GetPersistObjStreamPos( DWord ref )
{
   PPSDirEntry* pEntry = m_pFirstDirEntry;
   while ( pEntry )
   {
      DWord* pOffsets = pEntry->m_pOffsets;
      while ( (DWord)( (char*)pOffsets - (char*)pEntry->m_pOffsets ) <
pEntry->m_tableSize * sizeof( DWord ) )
      {
```

```
         DWord nRefs = pOffsets[0] >> 20;
         DWord base = pOffsets[0] & 0xFFFFF; // 1-based
         if ( ( base <= ref )&&( ref < base + nRefs ) )
            return pOffsets[ 1 + ref - base ];
         pOffsets += nRefs + 1;
      }
      pEntry = pEntry->m_pNext;
   }
   return(DWord) -1;
}

DWord PPSPersistDirectory::NumberOfAlreadySavedPersists()
{
   DWord count = 0;
   PPSDirEntry* pEntry = m_pFirstDirEntry;
   while ( pEntry )
   {
      DWord* pOffsets = pEntry->m_pOffsets;
      while ( (DWord)( pEntry->m_pOffsets - pOffsets ) < pEntry-
>m_tableSize * sizeof( DWord ) )
      {
         DWord nRefs = pOffsets[0] >> 20;
         count += nRefs;
         pOffsets += nRefs + 1;
      }
      pEntry = pEntry->m_pNext;
   }
   return count;
}

void FileReader::PPSReadUserEditAtom( DWord offset, PSR_UserEditAtom&
userEdit )
{
   LARGE_INTEGER li;
   li.LowPart = offset;
   li.HighPart = 0;
   GetDocStream()->Seek(li,STREAM_SEEK_SET, NULL);
   RecordHeader rh;
   GetDocStream()->Read(&rh, sizeof(rh), NULL);
   Assert( rh.recType == PST_UserEditAtom );
   Assert( rh.recLen == sizeof( PSR_UserEditAtom ) );
   li.LowPart = offset;
   GetDocStream()->Read(&userEdit, sizeof(userEdit), NULL);
}
```

```
void *FileReader::ReadRecord( RecordHeader& rh )
// Return values:
// NULL and rh.recVer == PSFLAG_CONTAINER: no record was read in.
// record header indicated start of container.
// NULL and rh.recVer != PSFLAG_CONTAINER: client must read in record.
{
   IStream *pStm = GetDocStream();
// read record header, verify
   pStm->Read(&rh, sizeof(rh), NULL); //AR: Check Error
// if client will read, do not read in record
   if ( DoesClientRead( rh.recType ) )
      return NULL;
// If container, return NULL
   if (rh.recVer == PSFLAG_CONTAINER)
      return NULL;
// Allocate buffer for disk record. Client must call ReleaseRecord() or
// pass the atom up to CObject::ConstructContents() which will
// then release it.
   void* buffer = new char[rh.recLen];
// read in record
   pStm->Read(buffer, rh.recLen, NULL);
// NOTE: ByteSwapping & versioning not done by this simple reader.
   return(buffer);
}

void FileReader::ReleaseRecord( RecordHeader& rh, void* diskRecBuf )
{
   if (rh.recType && rh.recVer!=PSFLAG_CONTAINER)
      delete [] (char*)diskRecBuf;
   rh.recType = 0; // consume the record so that record doesn't
// get processed again.
}

void FileReader::ReadPersistDirectory()
{
   if ( NULL != m_pLastUserEdit )
      return; // already read
   PSR_UserEditAtom userEdit;
   DWord offsetToEdit = m_currentUser.offsetToCurrentEdit;
   while ( 0 < offsetToEdit )
   {
      PPSReadUserEditAtom( offsetToEdit, userEdit );
      if ( NULL == m_pLastUserEdit )
      {
         m_pPersistDirectory = new PPSPersistDirectory();
```

```
         m_pLastUserEdit = new PSR_UserEditAtom;
         *m_pLastUserEdit = userEdit;
      }
      LARGE_INTEGER li;
      li.LowPart = userEdit.offsetPersistDirectory;
      li.HighPart = 0;
      GetDocStream()->Seek(li,STREAM_SEEK_SET, NULL); // AR: check that
seek succeeded.
      RecordHeader rh;
      DWord *pDiskRecord = (DWord*) ReadRecord(rh);
      Assert( PST_PersistPtrIncrementalBlock == rh.recType );
      m_pPersistDirectory->AddEntry( rh.recLen / sizeof( DWord ),
pDiskRecord );
      ReleaseRecord( rh, pDiskRecord );
      offsetToEdit = userEdit.offsetLastEdit;
   }
}

// PPStorage::ReadPersistDirectory
void FileReader::ReadSlideList()
{
   Assert( m_pLastUserEdit != NULL );
   DWord offsetToDoc = m_pPersistDirectory->GetPersistObjStreamPos
( m_pLastUserEdit->documentRef );
   LARGE_INTEGER li;
   li.LowPart = offsetToDoc;
   li.HighPart = 0;
   GetDocStream()->Seek(li,STREAM_SEEK_SET, NULL);
   ParseForSlideLists();
}

DWord FileReader::ParseForSlideLists()
{
   IStream *pStm = GetDocStream();
   RecordHeader rh;
   DWord nRd=0;
// Stack based parsing for SlideLists
   pStm->Read(&rh, sizeof(rh), &nRd);
   if ( ( rh.recVer != PSFLAG_CONTAINER ) && ( (rh.recVer & 0x0F)!
=0x0F ) )
   {
      if ( rh.recType == PST_SlidePersistAtom )
      {
         PSR_SlidePersistAtom spa;
         Assert( sizeof(spa) == rh.recLen );
```

```
            pStm->Read(&spa, sizeof(spa), &nRd);
            AddSlideToList( spa.psrReference );
         }
         else
         {
            LARGE_INTEGER li;
            li.LowPart = rh.recLen;
            li.HighPart = 0;
            pStm->Seek(li,STREAM_SEEK_CUR, NULL);
         }
         nRd += rh.recLen;
      }
      else
      {
         DWord nCur = 0;
         while ( nCur < rh.recLen )
         {
            nCur += ParseForSlideLists();
         }
         nRd += nCur;
      }
      return nRd;
}

BOOL FileReader::ReadText( WCHAR *pBuff, ULONG size, ULONG *pSizeRet )
{
   DWord offset;
   *pSizeRet = 0;
   m_pSizeRet = pSizeRet;
   m_pClientBuf = pBuff;
   m_clientBufSize = size;
   m_clientBufPos = 0;
   for ( ;; )
   {
      if ( ( m_pParseContexts == NULL ) )
      {
         if ( FindNextSlide(offset) )
         {
            if ( StartParse( offset ) )
               return TRUE;
         }
         else
            return FALSE; // DONE parsing, no more slides
      }
      else
```

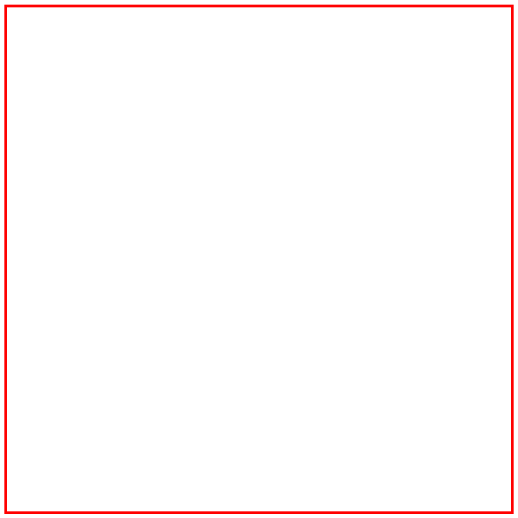
```
      {
         if ( m_pClientBuf )
         {
            if ( FillBufferWithText() ) // Use existing text first.
               return TRUE;
         }
         if ( Parse() ) // restart parse where we left off.
            return TRUE;
      }
   }
}

BOOL FileReader::StartParse( DWord offset )
{
   LARGE_INTEGER li;
   li.LowPart = offset;
   li.HighPart = 0;
   GetDocStream()->Seek(li,STREAM_SEEK_SET, NULL);
   m_pParseContexts = new ParseContext( NULL );
   GetDocStream()->Read(&m_pParseContexts->m_rh, sizeof(RecordHeader),
NULL);
   return Parse();
}

BOOL FileReader::Parse()
{
   IStream *pStm = GetDocStream();
   RecordHeader rh;
   DWord nRd=0;
   Assert( m_pParseContexts );
// Restarting a parse might complete a container so we test this
initially.
   if ( m_pParseContexts->m_nCur >= m_pParseContexts->m_rh.recLen )
   {
      Assert( m_pParseContexts->m_nCur == m_pParseContexts->m_rh.recLen );
      ParseContext* pParseContext = m_pParseContexts;
      m_pParseContexts = m_pParseContexts->m_pNext;
      delete pParseContext;
   }
   do
   {
      pStm->Read(&rh, sizeof(RecordHeader), NULL);
      if ( ( rh.recVer != PSFLAG_CONTAINER ) && ( (rh.recVer & 0x0F)!
=0x0F ) )
      {
```

```
         if ( rh.recType == PST_TextCharsAtom )
         {
            m_curTextPos = 0;
            m_curTextLength = rh.recLen/2;
            Assert( m_pCurText == NULL );
            m_pCurText = new WCHAR[rh.recLen/2];
            pStm->Read(m_pCurText, rh.recLen, &nRd);
            wprintf( L"-%s-\n", m_pCurText );
            if ( FillBufferWithText() )
               return TRUE; // Stop parsing if buffer is full, and return
control to client
         }
         else if ( rh.recType == PST_TextBytesAtom )
         {
            Assert( m_pCurText == NULL );
            m_curTextPos = 0;
            m_curTextLength = rh.recLen;
            m_pCurText = new WCHAR[rh.recLen];
            pStm->Read(m_pCurText, rh.recLen, &nRd);
            char *pHack = (char *) m_pCurText;
            unsigned int back2 = rh.recLen*2-1;
            unsigned int back1 = rh.recLen-1;
            for (unsigned int i=0;i<rh.recLen;i++)
            {
               pHack[back2-1] = pHack[back1];
               pHack[back2] = 0;
               back2 -=2;
               back1--;
            }
            if ( FillBufferWithText() )
               return TRUE; // Stop parsing if buffer is full, and return
control to client
         }
         else
         {
            LARGE_INTEGER li;
            ULARGE_INTEGER ul;
            li.LowPart = rh.recLen;
            li.HighPart = 0;
            pStm->Seek(li,STREAM_SEEK_CUR,&ul);
         }
         m_pParseContexts->m_nCur += rh.recLen;
         m_pParseContexts->m_nCur += sizeof( RecordHeader ); // Atom rh's
add towards containing container's size.
      }
```

```
      else
      {
         m_pParseContexts = new ParseContext( m_pParseContexts );
         m_pParseContexts->m_rh = rh;
      }
      if ( m_pParseContexts->m_nCur >= m_pParseContexts->m_rh.recLen )
      {
         Assert( m_pParseContexts->m_nCur == m_pParseContexts->m_rh.
recLen );
         ParseContext* pParseContext = m_pParseContexts;
         m_pParseContexts = m_pParseContexts->m_pNext;
         delete pParseContext;
      }
   } while ( m_pParseContexts && ( m_pParseContexts->m_nCur <
m_pParseContexts->m_rh.recLen ) );
   return FALSE;
}

BOOL FileReader::FindNextSlide( DWord& offset )
{
   if ( m_curSlideNum == 0 )
   {
      Assert( m_pLastUserEdit != NULL );
      offset = m_pPersistDirectory->GetPersistObjStreamPos
( m_pLastUserEdit->documentRef );
      m_curSlideNum++;
      return TRUE;
   }
   else
   {
      uint4 curSlideNum = m_curSlideNum++;
      SlideListChunk *pCur = m_pFirstChunk;
      while ( pCur && ( curSlideNum > pCur->numInChunk ) )
      {
         curSlideNum -= pCur->numInChunk;
         pCur = pCur->pNext;
      }
      if ( pCur == NULL )
         return FALSE;
      offset = m_pPersistDirectory->GetPersistObjStreamPos( pCur->refs
[curSlideNum-1] );
      return TRUE;
   }
}
```

```
static BOOL ReadText( void** ppContext, IStorage* pStgFrom, WCHAR*
buffer, unsigned long bufferSize, unsigned long* pSizeRet )
{
   FileReader* pFI = NULL;
   if ( *ppContext == NULL )
   {
      pFI = new FileReader( pStgFrom );
      *ppContext = pFI;
      if ( !pFI->IsPowerPoint() )
      {
         delete pFI;
         *pSizeRet = 0;
         return FALSE;
      }
      pFI->ReadPersistDirectory();
      pFI->ReadSlideList();
   }
   else
   {
      pFI = (FileReader *)*ppContext;
   }
   BOOL bRet = pFI->ReadText(buffer, bufferSize, pSizeRet);
   if ( !bRet )
   {
      delete pFI;
      *ppContext = NULL;
   }
   return bRet;
}

void main(int argc, char **argv)
{
   OLECHAR wc[256];
   HRESULT hr;
   IStorage *pStgFrom = NULL;
   if (argc < 2)
   {
      fprintf(stderr,"Usage dblock <file to be read>\n");
      exit(0);
   }
   MultiByteToWideChar( CP_ACP, MB_PRECOMPOSED, argv[1], -1, wc, 255);
   hr = StgOpenStorage(wc, NULL, STGM_READ | STGM_DIRECT |
                       STGM_SHARE_DENY_WRITE, NULL, 0, &pStgFrom);
   if (FAILED(hr))
   {
```

```
      fprintf(stderr,"Error (%d) opening docfile: %s\n",(int)hr,argv[1]);
   }
   else
   {
      WCHAR wcBuf[6];
      ULONG sizeUsed;
      BOOL fContinue = TRUE;
      void *pContext = NULL;
      while ( fContinue )
      {
         fContinue = ReadText( &pContext, pStgFrom, wcBuf, 5, &sizeUsed );
         wcBuf[sizeUsed] = 0;
         wprintf(L"-%s-\n", wcBuf);
      }
   }
}
```



Physical File Format

Each record, weather its an atom or a container, has a Record Header. The record header is a structure defined in "Serial.h" as follows:

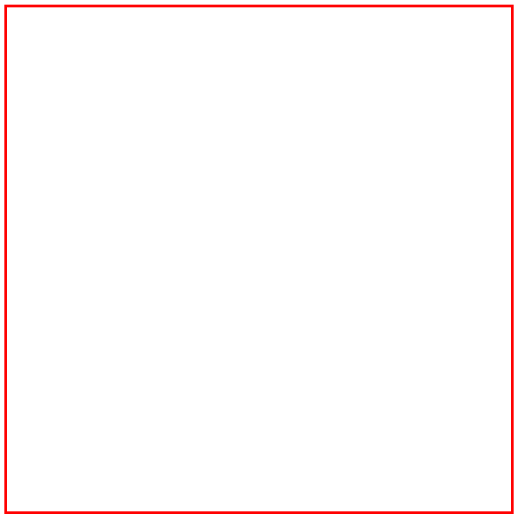
```
struct RecordHeader

{

psrVersion recVer : 4

psrInstance recInstance : 12;

psrType recType;

psrSize recLen;

};
```

Record Version: (recVer) Indicates the version if the record is an atom. All versions are prefixed by VER and are enumerated in "Serial.h". If the record is a container, this field has a value of 0xFFFF.

Record Instance: (recInstance) Differentiates atoms. Depending on the instance a record's contents it can have different meanings. For example a list container can store a list of slides or a list of fonts, and it's instance would vary accordingly. Instances are prefixed in "Serial.h" by INS. The instance of a record is useful for differentiating atoms when there is more than one atom of the same type in a particular container.

Record Type: (recType) Indicates the signature or type of the record. Each record has a symbolic and a numeric signature in "Serial.h". All the symbolic signatures are prefixed by PST. For example, the symbolic signature for a slide is PST_Slide which has a value of 1006. A description of each of the different types can be found in the Record Descriptions section.

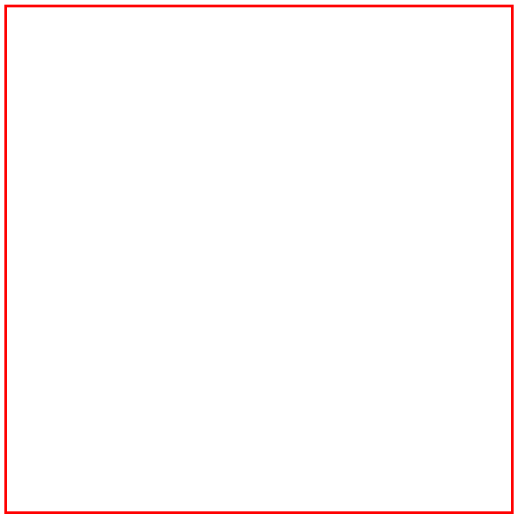
Record Length: (recLen) Stores the length of the record in bytes. If the record is an atom, it refers to the length of the atom excluding the header. If the record is a container, it refers to the sum of the lengths of the atoms inside it, plus the length of the record headers.



Record Descriptions

This section describes each of the storage types defined in "Serial.h". It contains the symbolic and numeric signature for each record. It is organized alphabetically by symbolic signatures, with the numeric signatures in parentheses next to it. For an index organized by number, please refer to Appendix A.

As stated before there are two kinds of storage elements in a PowerPoint file: atoms and containers. Atoms are described by indicating each of the fields' contents and their meaning. An atom's description is done in this section using types and offsets; but in "Serial.h" it is done using C++ language syntax. Containers are described in this section by indicating their use and the atoms and containers that they hold.



msofbtClientData (????)

Is a container for PowerPoint specific data of a shape.

The atoms that an msofbtClientData container has are:

An OEShapeAtom if any

An ExObjRefAtom if any

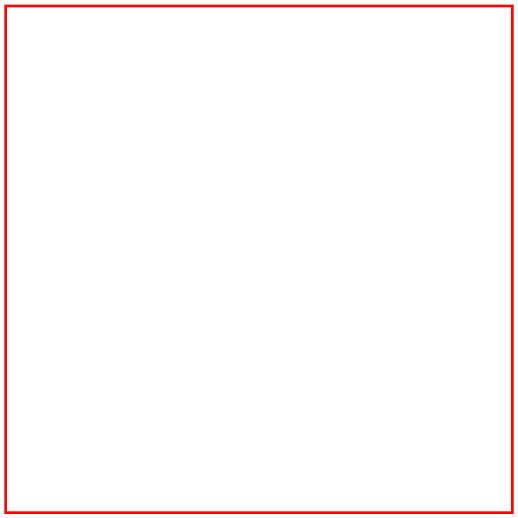
An AnimationInfo atom if any

An InteractiveInfo atom if any (Instance: MouseClick)

An InteractiveInfo atom if any (Instance: MouseMove)

A RecolorInfoAtom if any

A ProgTags container for OLE Automation



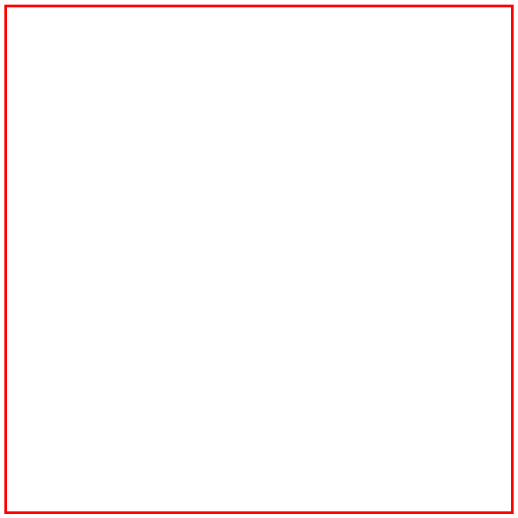
AnimationInfoAtom (4116)

Atom that contained all the data in AnimationInfo except reference to Sound object. AnimationInfo is part of OEShape.

AnimationInfoAtom Fields

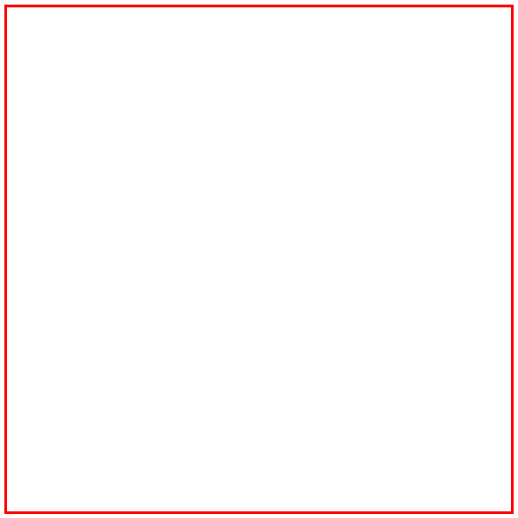
Offset	Type	Name	Contents
0	PSR_GrColorAtom	dimColor;	color to use for dimming
4	uint4	flags	set of flags that determine type of build
8	uint4	soundRef;	0 if storage is from clipboard. Otherwise index(ID) in SoundCollection list.
12	sint4	delayTime;	delay before playing object
16	uint2	orderID	order of build
18	uint2	slideCount	number of slides to play object
20	bool1	buildType	type of build
21	bool1	flyMethod	animation effect(fly, zoom, appear, etc)
22	bool1	flyDirection	Animation direction(left, right, up, down, etc)

23	bool1	afterEffect	what to do after build
24	bool1	subEffect	build by word or letter
25	bool1	oleVerb	Determines object's class (sound, video, other)



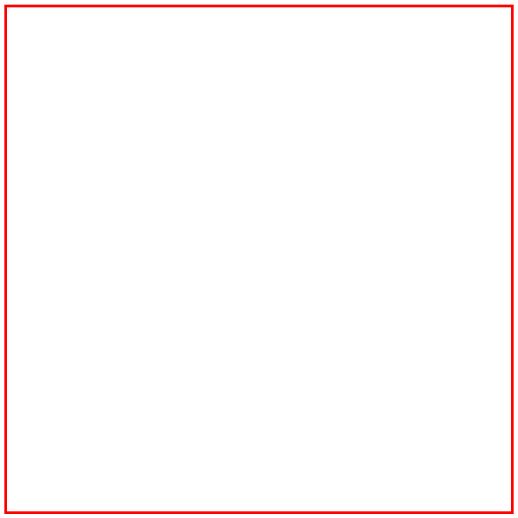
BaseTextPropAtom(4002)

Same as PST_StyleTextPropAtom but used for the master text. Since the attributes of a master text always reflect the ones of the style, all we need to store is a runlist with demotion levels. Special parsing code is needed to parse content of this atom.



BinaryTagData (5003)

The value of PST_ProgBinaryTag is the size of the binary data.



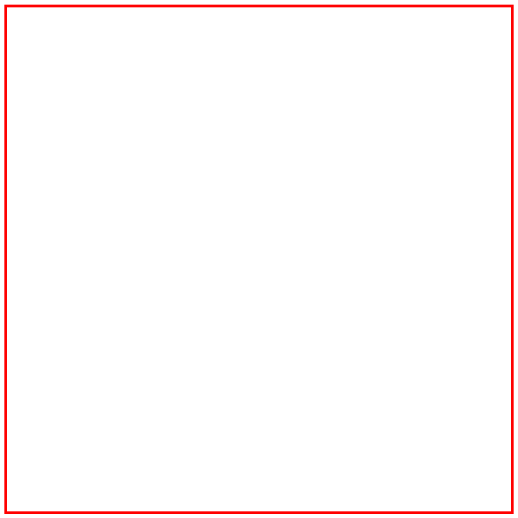
BookmarkCollection (2019)

A container for bookmark related atoms. Bookmarks are text links used mainly for exporting PowerPoint property fields to Lotus Notes fields or columns. The contents of a Bookmark Collection depend on whether the presentation has bookmarks or not. When the presentation doesn't have bookmarks, a bookmarkCollection contains only a bookmarkSeedAtom. When the presentation has bookmarks, it also contains a bookmarkSeedAtom, and in addition it contains a set of the following per bookmark:

A BookmarkEntityAtom

BookmarkEntityEnd

ListPlaceholder



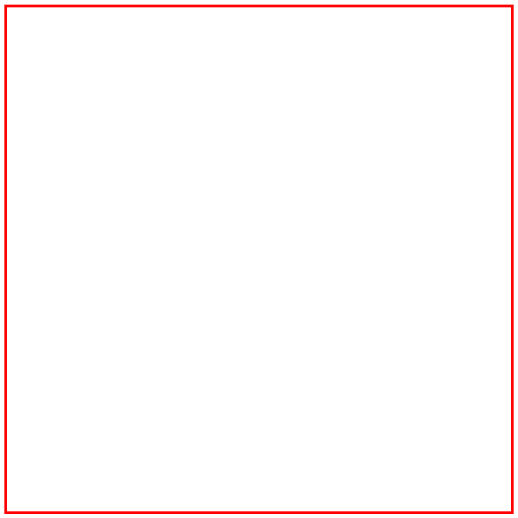
BookmarkEntityAtom (4048)

Atom that tracks bookmarks.

BookmarkEntityAtom Fields

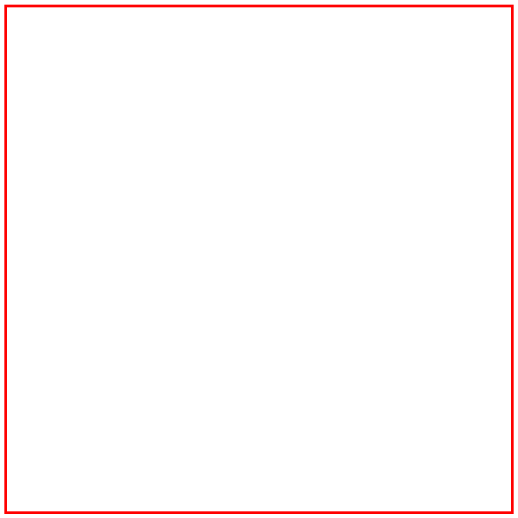
Offset	Type	Name	Contents
0	uint4	bookmarkID	Unique ID used to keep track of bookmarks.
4	uint2	bookmarkName	User-friendly bookmark name

Note: There has to be a one-to-one correspondence between bookmarks in the PowerPoint data and in the properties saved by the properties dialog (which is done by Office). If PowerPoint detects any discrepancy between the two sets of data, PowerPoint will delete the bookmark. This situation can arise naturally if the user employs a third party tool to change the properties of a presentation.



BookmarkSeedAtom (2025)

This atom is a 4 byte unsigned integer that contains the bookmark ID. This ID is a number used internally by PowerPoint to compute a unique ID for the bookmark. If you are trying to create a new bookmark outside of PowerPoint, you should give the bookmark ID a number higher than this one.

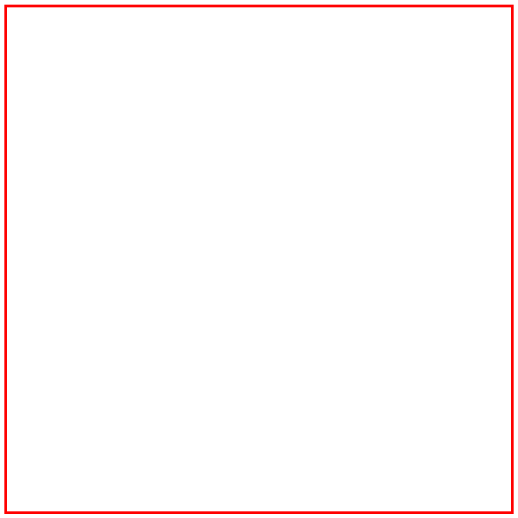


CharFormatAtom : Character Format Atom (4066)

Atom that keeps data for text. See ArrayElementAtom container for cases when it is used.

CharFormatAtom Fields

Offset	Type	Name	Contents
0	sint2	cfSBCTypeface	Single byte typeface reference
2	sint2	cfDBCTypeface	Double byte typeface reference
4	sint4	cfSize	Font size
8	uint2	cfStyle	Font style (bold, italic, etc)
10	GrColorAtom	cfColor	The RGB value for color
14	Padding	padding	Padding
16	sint4	cfPosition	Baseline of subscript or superscript
20	sint4	cfKern	Amount to kern between characters



ColorSchemeAtom (2032)

The color scheme atom is an array of 8 color references (COLORREF), which contain the RGB value for each color in the color scheme. The order of scheme colors is as in the custom tab of the Color Scheme dialog:

[0] Background

[1] Text and lines

[2] Shadows

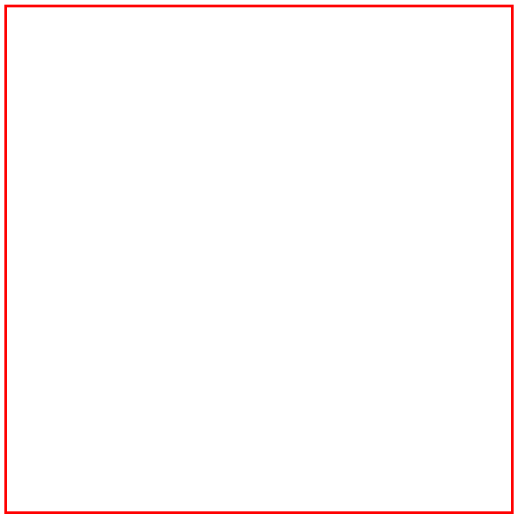
[3] Title text

[4] Fills

[5] Accent

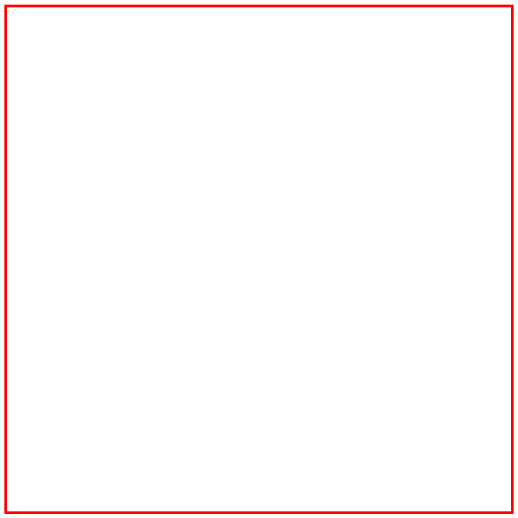
[6] Accent and hyperlink

[7] Accent and followed hyperlink



CString (4026)

CString is a special container, its size is variable depending on the length on the string. CString characters are stored in UNICODE. The unit of the size is in bytes so it is twice the number of characters in the string.



CurrentUserAtom (4086)

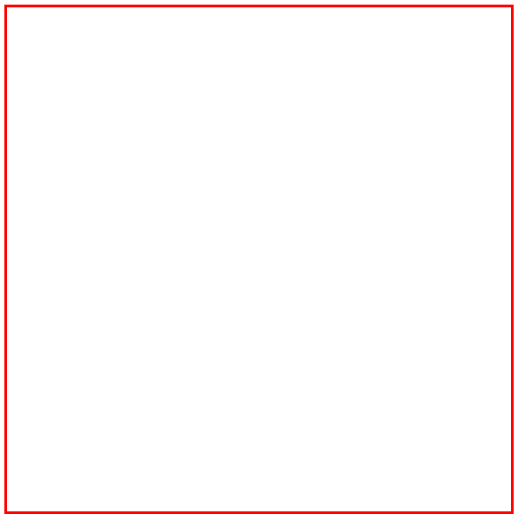
This is written to the current user stream. The interpretation of the OffsetToCurrentEdit is crucial to locate the top level UserEditAtom.

CurrentUserAtom Fields:

Offset	Type	Name	Contents
0	uint4	Size	sizeof(PSR_CurrentUserAtom)
4	uint4	Magic	Magic number to ensure this is a PowerPoint file(-476987297)
8	uint4	OffsettoCurrentEdit	Offset in main stream to current edit field
12	uint2	LenUserName	Length of user name
14	uint4	DocFileVersion	1012 for PP97
18	Ubyte1	majorVersion	3 for PP97
19	Ubyte1	minorVersion	0 for PP97

After the atom the name of the user who last edited the document is written. It is used to display a proper warning message, when multiple users open the same physical file on a network at the same time. After the user name a four byte integer value is written that

contains the release version. Its value is 8,9 or 10 for PP97 files.

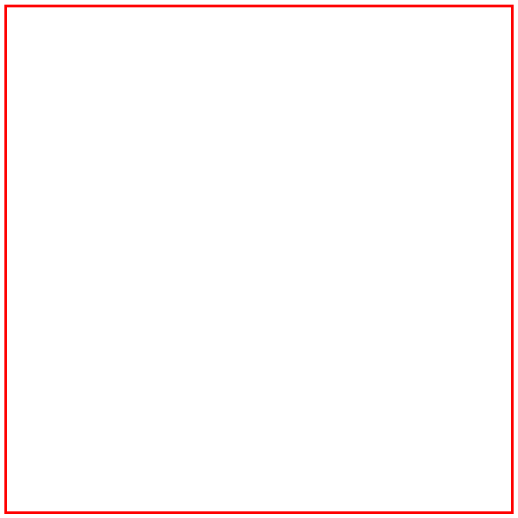


DateTimeMCAtom (4087)

DateTimeMCAtom is a record that stores the position of a date in a text and it also stores which of thirteen standard PowerPoint formats the date takes the form of. See the Date and Time dialog for all these different formats.

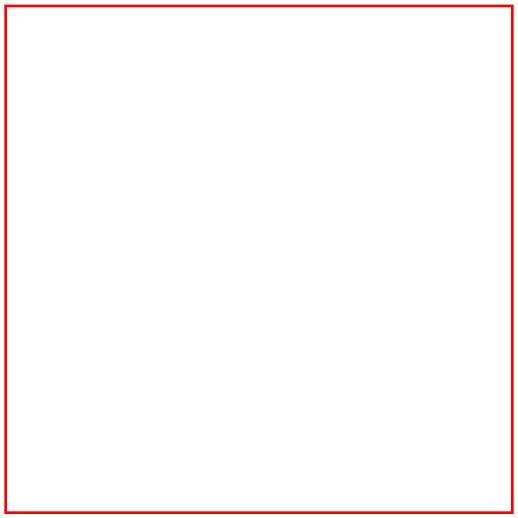
DateTimeMCAtom fields

Offset	Type	Name	Content
0	sint4	position	The position of the character in a text.
4	ubyte1	index	A number from 0-12 that specifies a date format.



DefaultRulerAtom (4011)

Single ruler property container. Storing differences to a style. Used only within PST_Environment container the to store the default ruler for new texts. Special parsing code is needed to parse content of this atom.



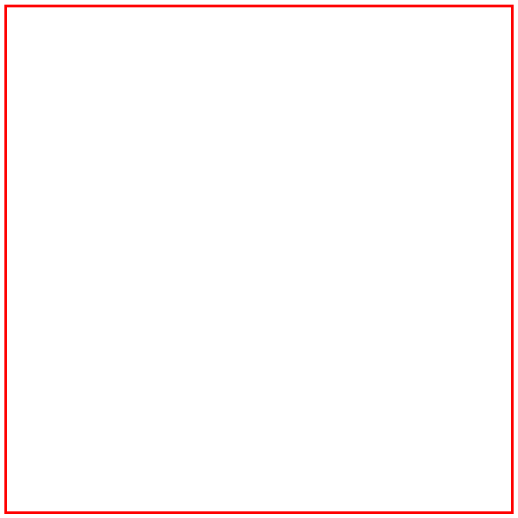
DocRoutingSlip (1030)

This is a variable length atom used by PowerPoint to keep the information needed to send a document by electronic mail.

DocRoutingSlip Fields

Offset	Type	Name	Contents
0	uint4	length	Length of this structure, including this field
4	uint4	version	Version of the router software
8	uint4	recipCount	Number of recipients
12	uint4	currentRecipient	Serial number of the current recipient

16	uint4	routingOptions	Options used for routing, the following are the valid values, which can be added together: 0x00000000 - all at once 0x00000001 - one after another 0x00000002 - return when done 0x00000004 - track status 0x00000008 - document dirty 0x00000010 - document routed 0x00000020 - cycle_completed
20	uint4	noStrings	Number of string elements in the next field
24	Cstring	Recipients[noStrings]	Array containing the list of recipients



Document : Powerpoint Document (1000)

Document is a container that marks the beginning of the PowerPoint document. It contains:

A document atom with miscellaneous information (Type: DocumentAtom)

A list of external objects descriptors if any

A subcontainer for the document's environment (Type: Environment & Instance: DocEnvironment)

A list of sounds if any

Document global Office Art information

A list of the master slides (Type: List & Instance: DocMasterList)

A list of information about the view of the slide. (Type: List & Instance: DocInfoList)

Slide header and footer information if any

Notes header and footer information if any

A list of the slides in the presentation (Type: List & Instance: DocSlideList) if any

A list of the notes slides (Type: List & Instance: DocNotesList) if any

Slide Show Information (Type: SSDocInfoAtom & Instance: DocSlideShowInfo) if any

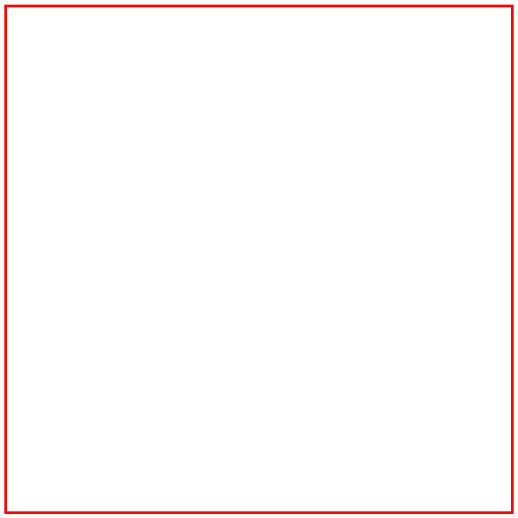
A names shows collection if any

Summary info information if any

Document routing information (Type: DocRoutingSlip) if any

Printer information if any

An end of document marker (Type: EndDocument)



DocumentAtom (1001)

A document atom is a record that stores miscellaneous information about the PowerPoint presentation.

DocumentAtom Fields

Offset	Type	Name	Contents
0	PointAtom	slideSize	Slide size in Master coordinates
8	PointAtom	NotesSize	Notes page size
16	RatioAtom	serverZoom	The scale used when the Powerpoint document is embedded. The default is 1: 2
24	Uint4	NotesMasterPersist	Reference to NotesMaster (0 if none)
28	Uint4	HandoutMasterPersist	Reference to HandoutMaster(0 if none)
32	uint2	firstSlideNum	Number of the first slide
34	sint2	slideSizeType	Size of the document's slides. Valid values are from 0-6. *See SlideSize field values table below for valid values.*

36	bool1	saveWithFonts	indicates if document was saved with embedded true type fonts
37	bool1	omitTitlePlace	Set if the placeholders on the title slide are omitted
38	bool1	RightToLeft	Flag for Bidi version
39	bool1	showComments	Visibility of comment shapes

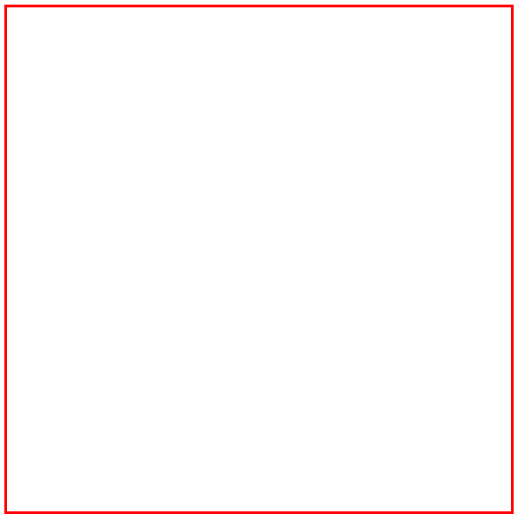
SlideSize Field Values

Value	**Meaning**
0	On screen
1	Letter sized paper
2	A4 paper
3	35mm
4	Overhead
5	Banner
6	Custom

Last View Field Values

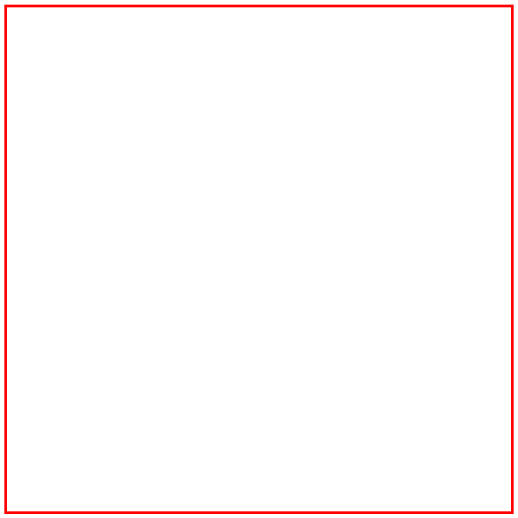
Value	**Meaning**
0	none
1	slide
2	slide master

3	notes
4	handout page
5	notes master
6	outline master
7	outline view
8	sorter view
9	not used
10	title master
11	slide show



EndDocument (1002)

Marks the end of the Document container.



Environment (1010)

The container for shared text entities, such as fonts, styles, rulers, etc. This container has:

Kinsoku container that stores data for Japanese word wrap. (Type: SrKinsoku & Instance: DocKinsoku)

Font list container

Ruler collection container.

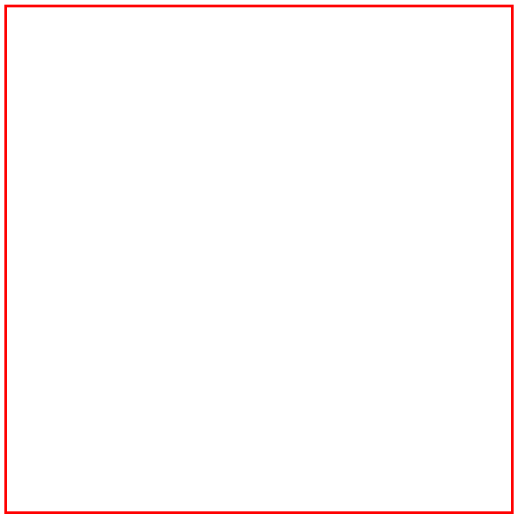
Default character attributes

Default paragraph attributes

Default text ruler

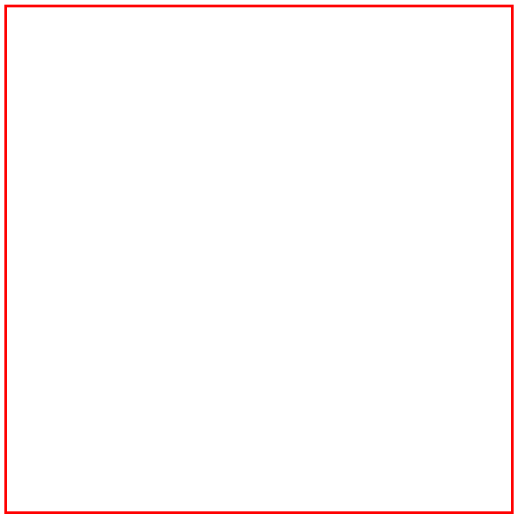
Spellchecking defaults

A container for the text styles.



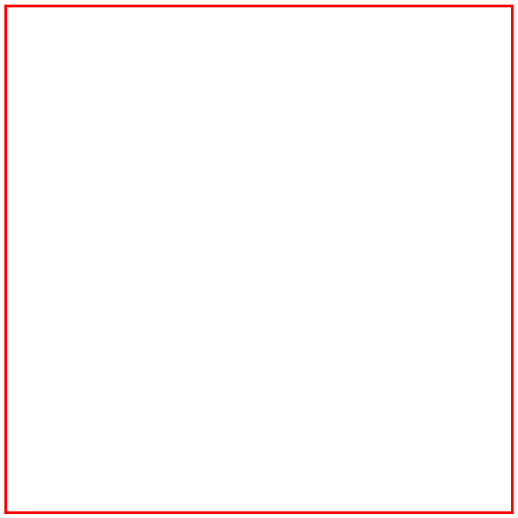
ExAviMovie (4102)

The ExAviMovie container stores data relating to an .avi movie and will contain an ExMediaAtom record followed by a ExVideo subcontainer.



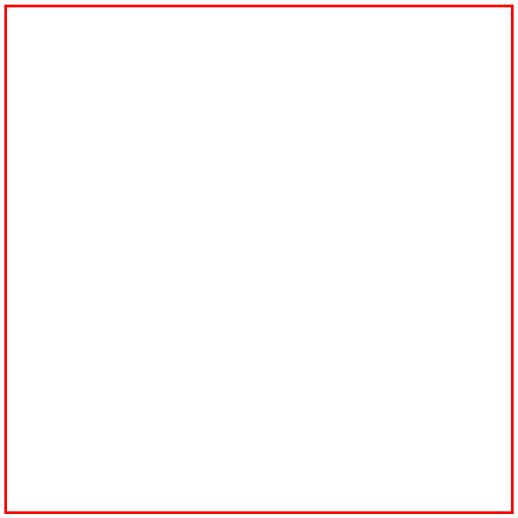
ExCDAudio (4110)

The ExCDAudio container consists of an ExMediaAtom record followed by an ExCDAudioAtom record.



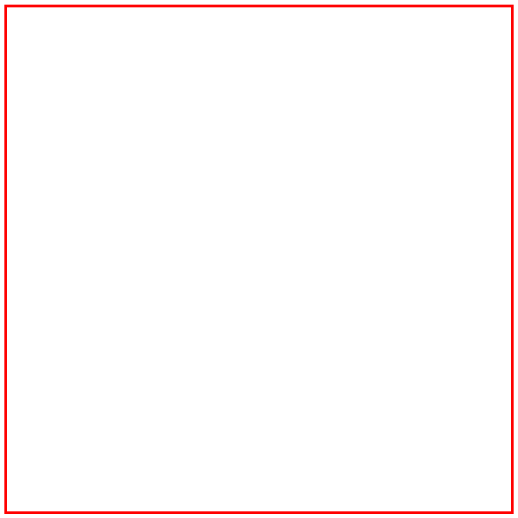
ExCDAudioAtom (4114)

The PSR_ExCDAudioAtom structure has 2 members, 'start' and 'end'. Both members are DWords in tmsf format -- frame:minute:seconds:track.



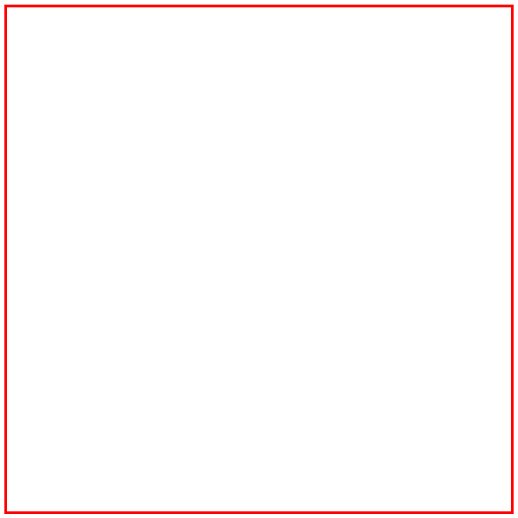
ExControl (4078)

Container for OLE Control object.



ExControlAtom (4091)

Contains a long integer, slideID, which stores the unique slide identifier of the slide where this control resides.



ExEmbed (4044)

A container for embedded objects. It contains:

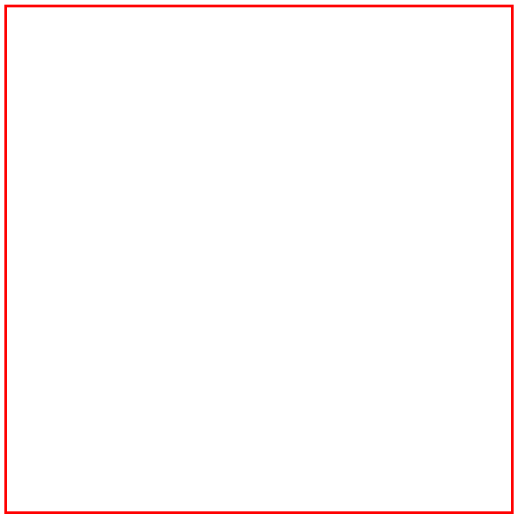
An ExEmbedAtom.

ExOleObjAtom that contains OLE information.

CString and Instance MenuName(2) used for menus and the Links dialog box.

CString and Instance ProgID (3) that stores the OLE Programmatic Identifier. A ProgID is a string that uniquely identifies a given object.

CString and Instance ClipboardName (45) that appears in the paste special dialog.

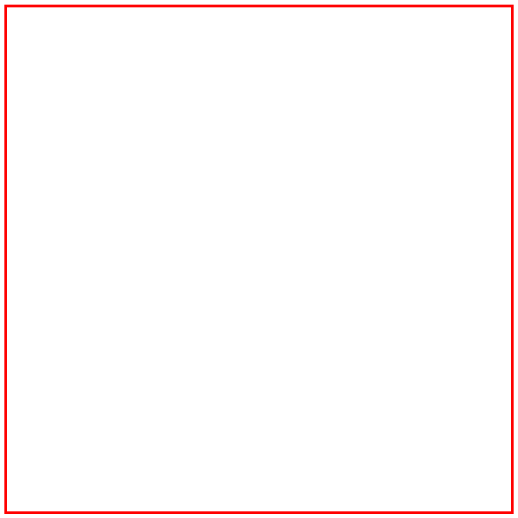


ExEmbedAtom (4045)

This atom contains information about an embedded object.

ExEmbededAtom Fields

Offset	Type	Name	Contents
0	sint4	followColorScheme	This field indicates how the object follows the color scheme. Valid values are: 0 - doesn't follow the color scheme 1 - follows the entire color scheme 2 - follows the text and background scheme
4	bool1	cantLockServerB	Set if the embedded server can not be locked
5	bool1	noSizeToServerB	Set if don't need to send the dimension to the embedded object
6	Bool1	isTable	Set if the object is a Word table



ExHyperlink (4055)

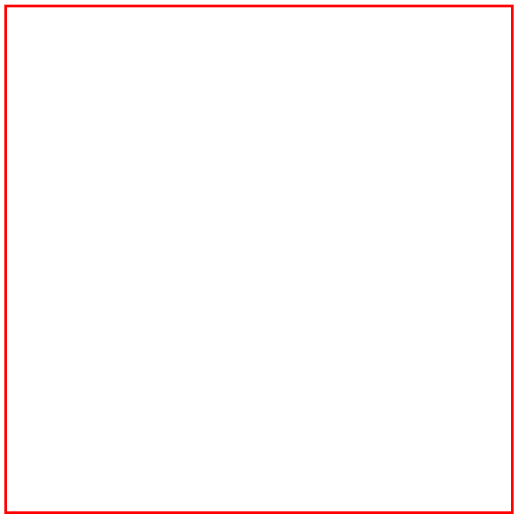
The PST_ExHyperlink container consists of an PST_ExHyperlinkAtom

record followed by three serialized CStrings:

INS_FriendlyName The hyperlink's user-readable name;

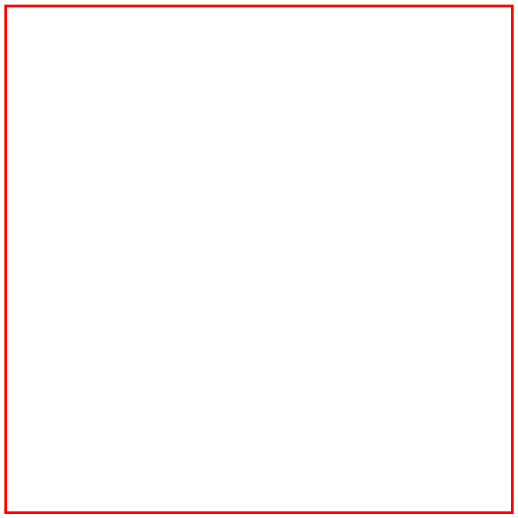
_Target The full path of the hyperlink destination file;(clipboard only)

INS_Location The hyperlink's location within the destination file (in app-specific format, clipboard only)



ExHyperlinkAtom (4051)

The 'objID' member is a persistent unique identifier to an ExternalObject.



ExLink (4046)

A container for OLE linked objects. It contains:

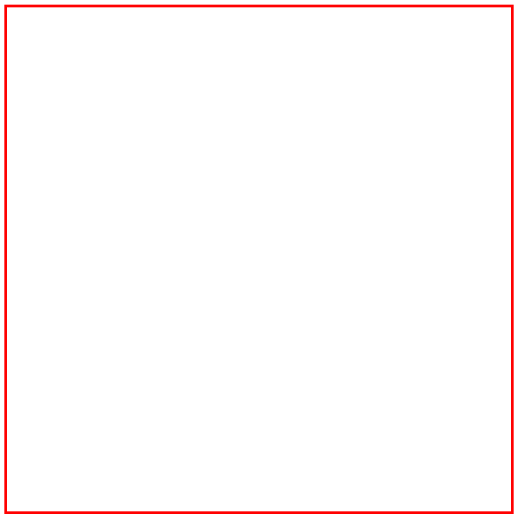
An ExLinkAtom that has information about the linked object.

ExOleObjAtom that contains OLE information.

CString and Instance MenuName(2) used for menus and the Links dialog box.

CString and Instance ProgID (3) that stores the OLE Programmatic Identifier. A ProgID is a string that uniquely identifies an OLE object.

CString and Instance ClipboardName (45) that appears in the paste special dialog.

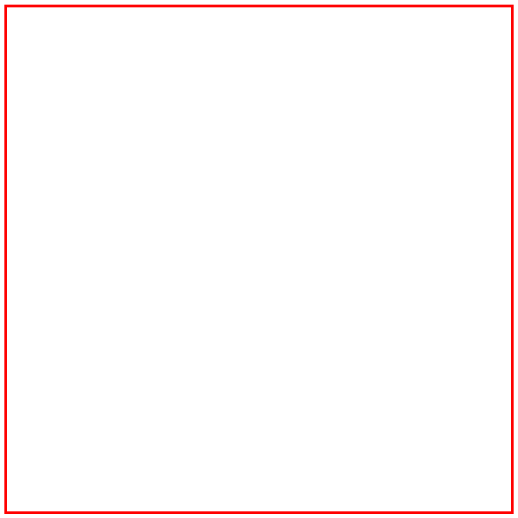


ExLinkAtom (4049)

This atom contains information about an OLE linked object.

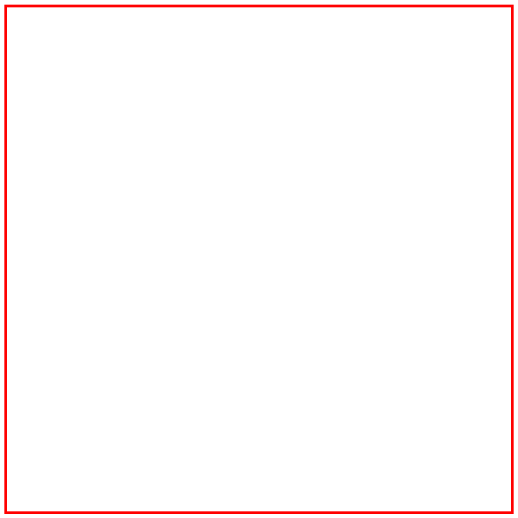
ExLinkAtom Fields

Offset	Type	Name	Contents
0	uint4	exObjId	Unique external object identifier
4	Uint2	flags	Stores the way the link is updated. This can be changed with the links dialog in the edit menu. The valid values are: 1 - automatic 3 - manual
5	bool1	unavailable	Set if the linked object is not available



ExMCIMovie (4103)

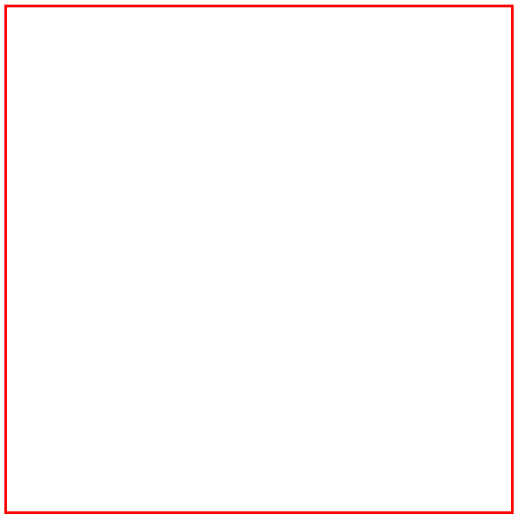
The ExMCIMovie container stores data relating to an MCI movie will contain an ExMediaAtom record followed by a ExVideo subcontainer.



ExMediaAtom (4100)

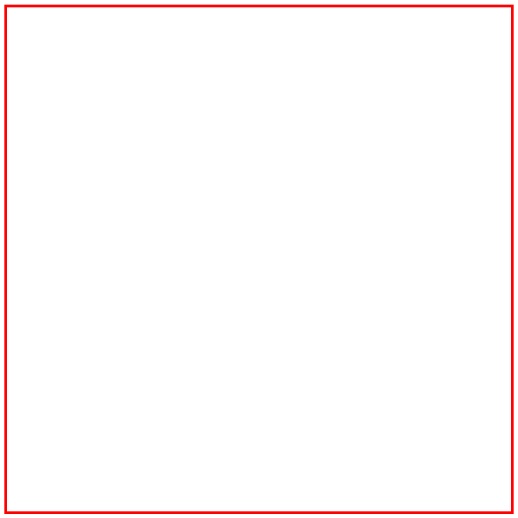
ExMediaAtom Fields

Offset	Type	Name	Contents
0	uint4	exObjId	Unique external object identifier
4	Uint2	flags	Bit1: Loop continuously Bit2: Rewind after play Bit3: Media is a narration



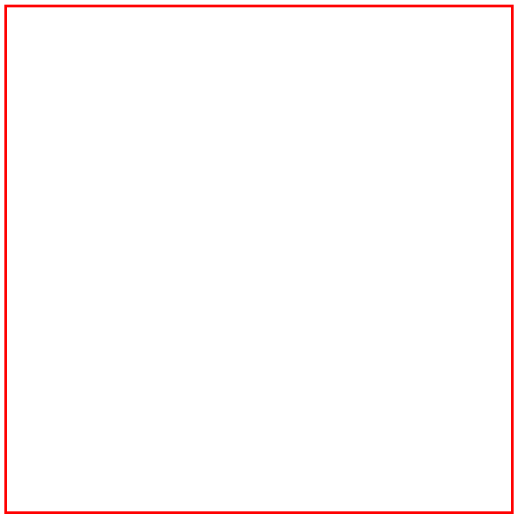
ExMIDIAudio (4109)

The ExMIDIAudio container consists of an ExMediaAtom record followed by a serialized CString which is the path to the audio file



ExObjList (1033)

Contains an ExObjListAtom and a list of all ExternalObject in a document. External Objects are ExEmbeds, ExLinks, ExControls, ExHyperlinks, ExAviMovie, ExCDAudo, EXWavAudioEmbedded, ExVavAudioLinked, ExMidiAudo, ExMCIMovie



Slide: (1006)

This container represents a PowerPoint slide. Its contents are:

A SlideAtom

A SlideShowInfoAtom if any

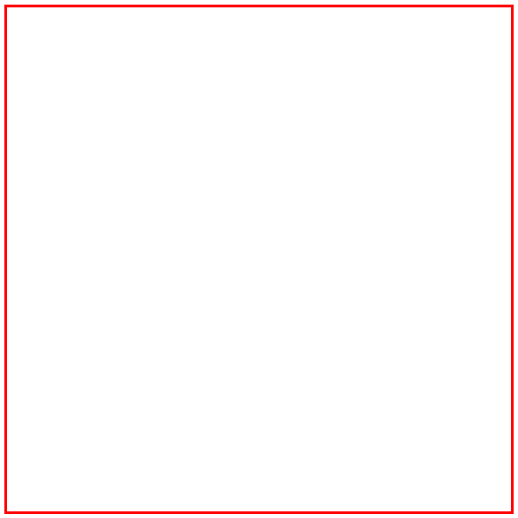
A HeaderFooter if any

A PPDrawing container

A slide color scheme container (Type: Scheme and Instance: 1)

A CString representing the slide name if any (Instance: 3)

A TagInfo container if any

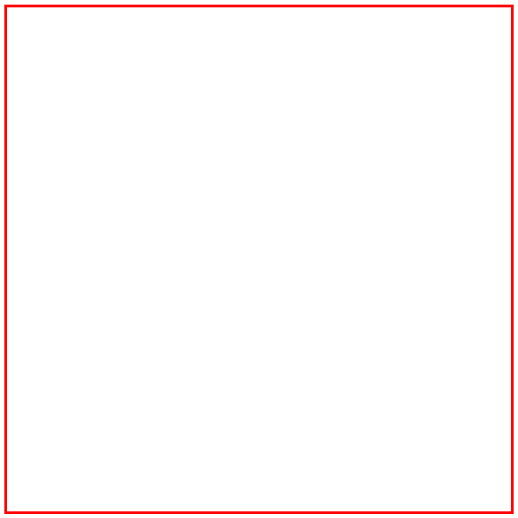


SlideAtom: (1007)

This atom stores the slide id and the slide master id.

SlideAtom Fields

Offset	Type	Name	Contents
0	SSSlideLayoutAtom	layout	Slide layout descriptor
12	sint4	masterId	This number identifies the master of the slide. It's null for a master slide
16	sint4	notesId	id referencing the corresponding notes slide. 0 if slide has no notes slide.
20	Uint2	Flags	Bit 1: Follow master objects Bit 2: Follow master scheme Bit 3: Follow master background



Notes (1008)

The Notes container is very similar to the Slide container and it represents the Notes pages of a presentation. Notes is a container for atoms and containers as follows:

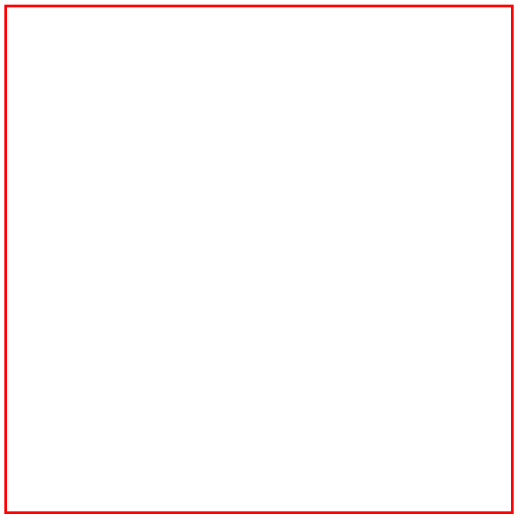
Notes Atom

A PPDrawing container

A slide color scheme container (Type: Scheme and Instance: 1)

A CString representing the slide name if any (Instance: 3)

A TagInfo container if any

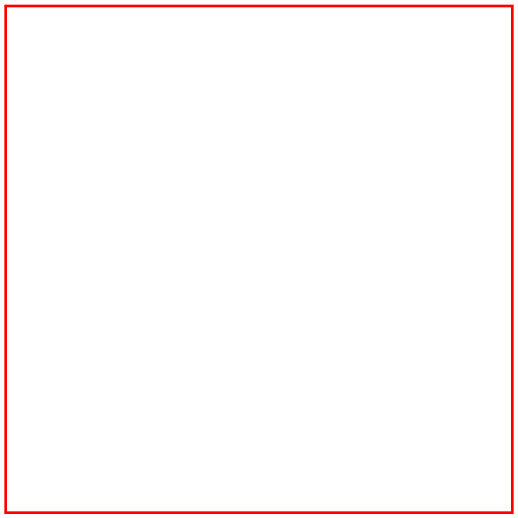


NotesAtom (1009)

A NotesAtom stores the id of the slide that owns the notes.

NotesAtom Fields

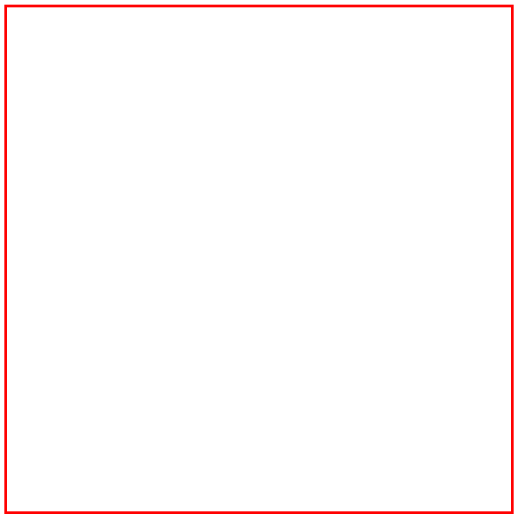
Offset	Type	Name	Contents
0	sint4	slideId	Number that identifies the slide
4	Uint2	Flags	Bit 1: follow master objects Bit 2: follow master scheme Bit 3: follow master background



SlidePersistAtom (1011)

SlidePersistAtom contains the information for the slide stub objects in the slide lists. The real slide data is stored in a different persist object which can be loaded\saved incrementally. The document saves all SlidePersistObjects in its persist stream so if you launch the number of slides and it's titles are available without loading all the slides.

Offset	Type	Name	Contents
0	uint4	psrReference	logical reference to the slide persist object
4	uint4	flags	only bit 3 used, if set then slide contains shapes other than placeholders
8	sint4	numberTexts	number of placeholder texts stored with the persist object. Allows to display outline view without loading the slide persist objects
12	sint4	slideId	Unique slide identifier, used for OLE link monikers for example
16	Uint4	Reserved	Unused field, always 0



SSlideLayoutAtom (1015)

Stores the slide's geometric layout, and the placeholders' ID.

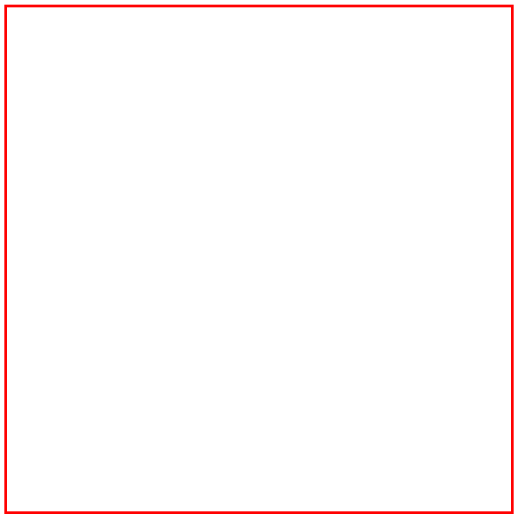
SSlideLayoutAtom Fields

Offset	Type	Name	Contents
0	sint4	Geom	Stores the geometric layout of the slide, this value can go from 0 to 18, and it identifies the position and number of placeholders. *See the Slide Layout table on the next page.*
4	ubyte1	PlaceholderId [8]	This field has an ID that identifies each of the placeholders on the slide. To see the meaning of each slide ID, refer to the PlaceholderID Values table under the OEPlaceholderAtom entry.

Slide Layout Table

Flag	Meaning
0	The slide is a title slide
1	Title and body slide

2	Title master slide
3	Master slide layout
4	Master notes layout
5	Notes title/body layout
6	Handout layout, therefore it doesn't have placeholders except header, footer, and date
7	Only title placeholder
8	Body of the slide has 2 columns and a title
9	Slide's body has 2 rows and a title
10	Body contains 2 columns, right column has 2 rows
11	Body contains 2 columns, left column has 2 rows
12	Body contains 2 rows, bottom row has 2 columns
13	Body contains 2 rows, top row has 2 columns
14	4 objects
15	Big object
16	Blank slide
17	Vertical title on the right, body on the left
18	Vertical title on the right, body on the left split into 2 rows



MainMaster (1016)

This container represents the master slide in a presentation. As such, most of its contents are the ones that a Slide container would have, such as :

A Slide Atom

A SlideBase Atom

Headers and Footers container if any

Slide Scheme objects that represent the predfined schemes for the presentation if any. (Instance: 6)

Master text styles

A SSSlideInfoAtom which keeps the SlideShow transitions and builds if any

A HeadersFootersObject if any

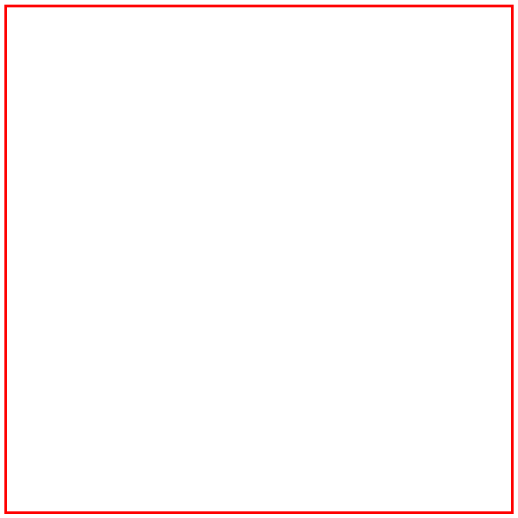
A PPDrawing container

A slide color scheme container (Type: Scheme and Instance: 1)

A CString representing the slide name if any (Instance: 3)

A TagInfo container if any

A CString representing the template name if any (Instance : 2)



SSSlideInfoAtom (1017)

This atom keeps the information for the slide's transitions. The TransType field and the direction field together define a build effect.

SSSlideInfoAtom Fields

Offset	Type	Name	Contents
0	sint4	transType	Type of transition. *See the Transition Type table below.*
4	sint4	speed	Speed of the transition
8	sint4	direction	Direction of the transition. *See Direction table below*
12	sint4	slideTime	How long to show the slide in ticks
16	sint4	buildFlags	Flags that determine the type of build. *See Build Flags table below*
20	sint4	soundRef	Index to a sound in the soundCollection

Transition Types

Flag	Meaning

0	No transition
1	Random
2	Blinds
3	Checker
4	Cover
5	Dissolve
6	Fade
7	Pull
8	Random bars
9	Strips
10	Wipe
11	Zoom
13	Split

Direction Values for Transitions

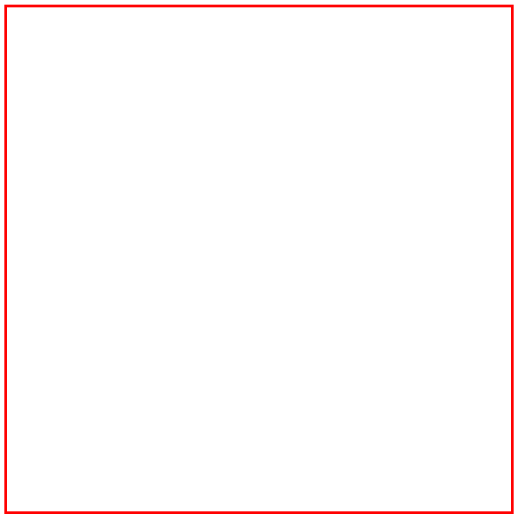
Type of transition	**Value for** direction	**Meaning**
Random & Dissolve	0	Anywhere

Wipes & Covers	0	Left
	1	Up
	2	Right
	3	Down
	4	Left up
	5	Right up
	6	Left down
	7	Right down
Strips	0	Up left
	1	Up right
	2	Down left
	3	Down right
	4	Right up
	5	Left down
	6	Right down
Zoom	0	Out
	1	In
Blinds & Stripes	0	Horizontal
	1	Vertical

Cuts	0 1 2	No black To black Best cut
Splits	0 1 2 3	Horizontal out Horizontal in Vertical out Vertical in
Flash	0 1 2	Fast Medium Slow

BuildFlags Field Values

Flag	**Meaning**
0	Advance on mouse click
2	Hidden slide
4	The slide has sound
6	Loop until next sound
8	Stop the previous sound



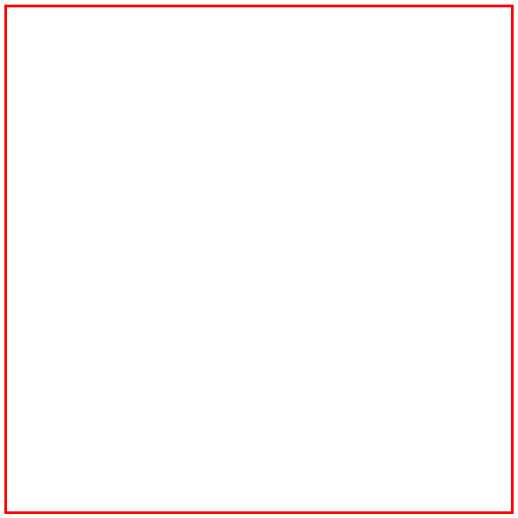
SlideViewInfo (1018)

A container that keeps the state of the grid, guide, and view scale. It's contents are:

A SlideViewInfoAtom which contains information about the guides and the grid

A GuideList container for the guides

A ViewInfoAtom that contains information about the view scale.

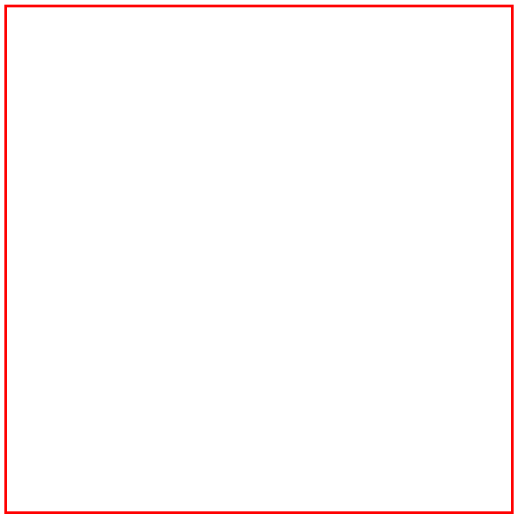


GuideAtom (1019)

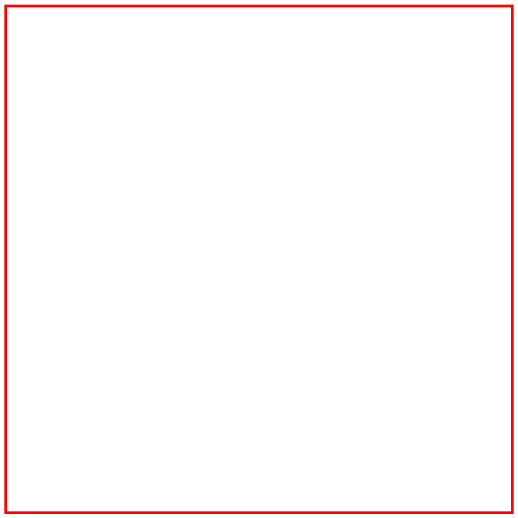
This atom stores information about the guides in a slide.

GuideAtom Fields

Offset	Type	Name	Contents
0	sint4	type	Type of the guide. If the guide is horizontal this value is zero. If it's vertical, it's one.
4	sint4	pos	Position of the guide in master coordinates. X coordinate if it's vertical, and Y coordinate if it's horizontal.



ViewInfo (1020)

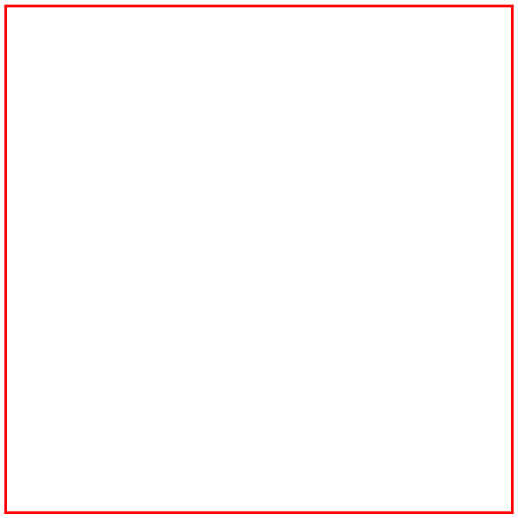


ViewInfoAtom (1021)

Contains information about the scale at which the slide is seen.

ViewInfoAtom Fields

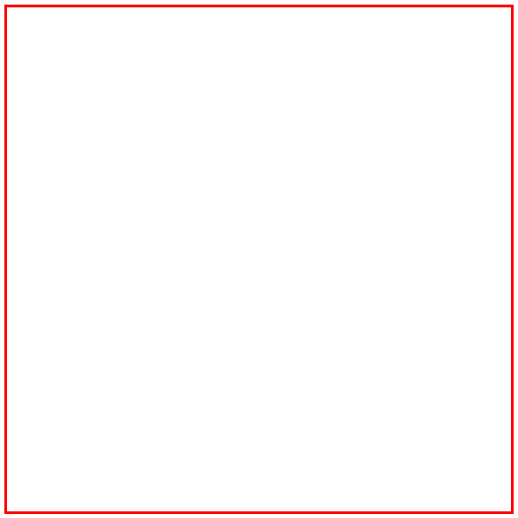
Offset	Type	Name	Contents
0	PSR_GScalingAtom	CurScale	Keeps the current scale
16	PSR_GScalingAtom	PrevScale	Keeps the previous scale
32	PSR_GPointAtom	ViewSize	Keeps the size of the view in master coordinates
40	PSR_GLPointAtom	Origin	Keeps the origin in master coordinates
48	bool1	varScale	Set if zoom to fit is set
49	bool1	draftMode	Not used
50	2 byte padding	padding	Padding



SlideViewInfoAtom (1022)

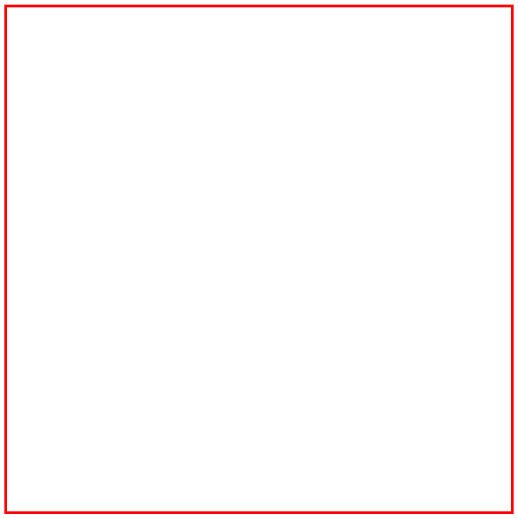
This atom keeps information about the guides and the grid, its fields are:

Offset	Type	Name	Contents
0	bool1	showGuides	Set if the guides are visible.
1	bool1	snapToGrid	Set if snap to grid is on.
2	bool1	SnapToShape	Set if snap to shape is on.



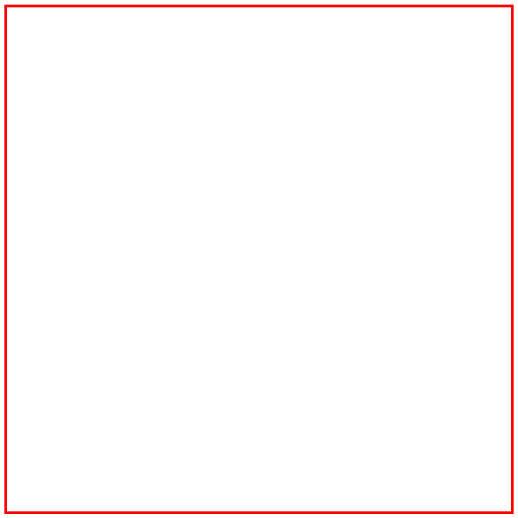
VBAInfo (1023)

Information used internally by PowerPoint.



VBAInfoAtom (1024)

Information used internally by PowerPoint.



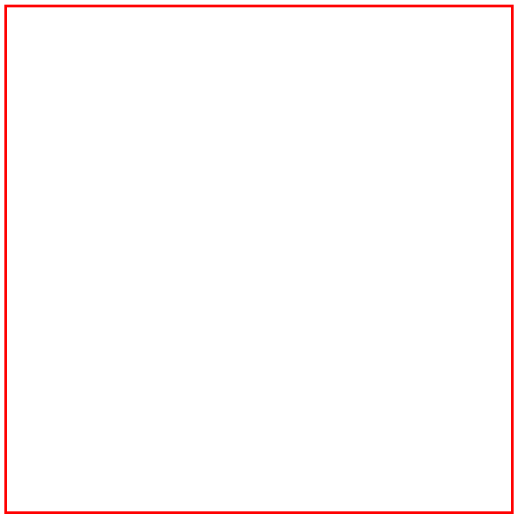
SSDocInfoAtom (1025)

Atom that keeps Slide Show settings for the whole presentation.

SSDocInfoAtom Fields

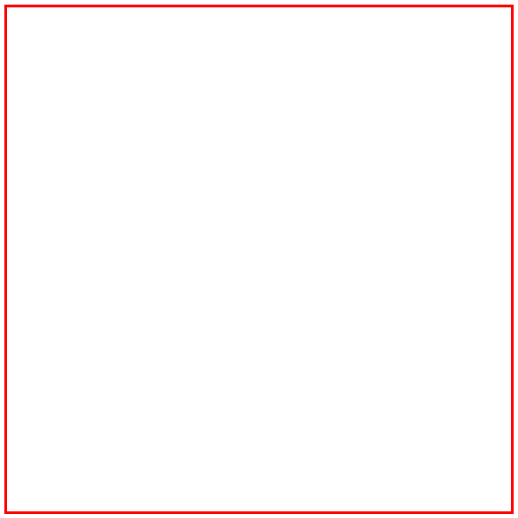
Offset	Type	Name	Contents
0	GrColorAtom	PenColor	Color for the on screen notation pen
4	Sint4	RestartTime	Time for auto restart of slide show in kiosk mode in millisec.
8	Sint2	StartSlide	First slide in slideshow
10	Sint2	EndSlide	Last slide in slideshow
12	Uint2[32]	NamedShow	Named show identifier

76	Uint2	Flags	Bit 1: Auto advance Bit 2: Skip builds Bit3: Use slide range Bit 4: Use named show Bit 5: Browse mode on Bit 6: Kiosk mode on Bit 7: loop continously Bit 8: show scrollbar



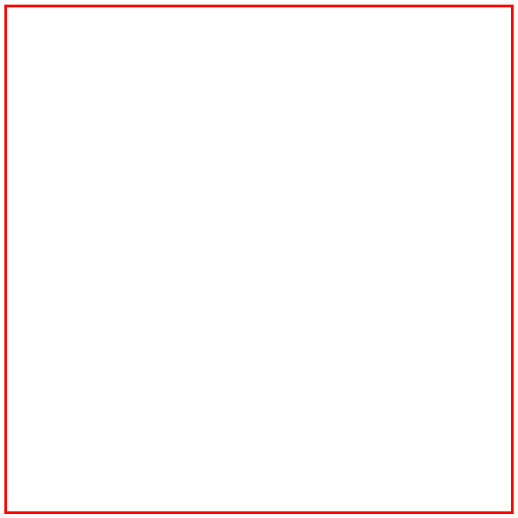
Summary (1026)

A container for the presentation's bookmarks. It contains a BookmarkCollection container.



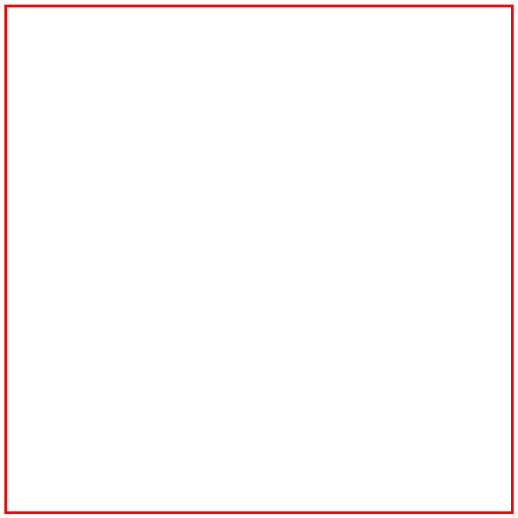
OutlineViewInfo (1031)

This container keeps information about the view settings for Outline view. It contains only a ViewInfoAtom.



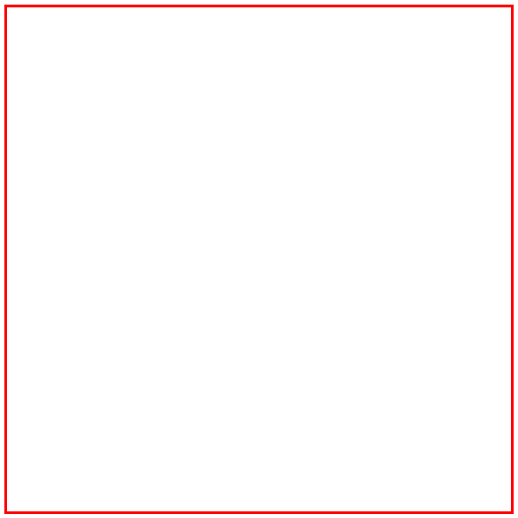
SorterViewInfo (1032)

This container keeps information about the view settings for Slide Sorter view. It contains only a ViewInfoAtom.



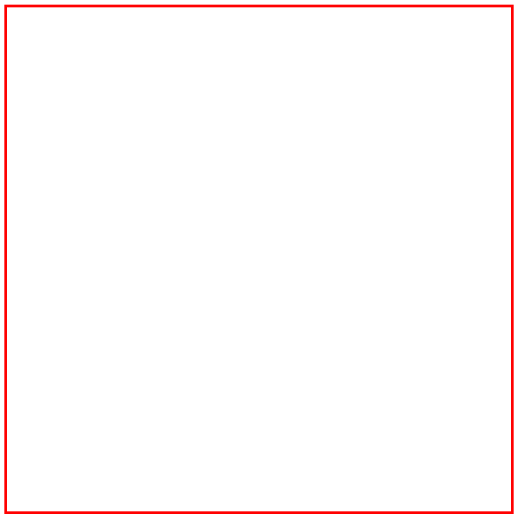
ExObjListAtom (1034)

Contains a long integer, objectIdSeed, which stores value for the next unique identifier for ole objects.



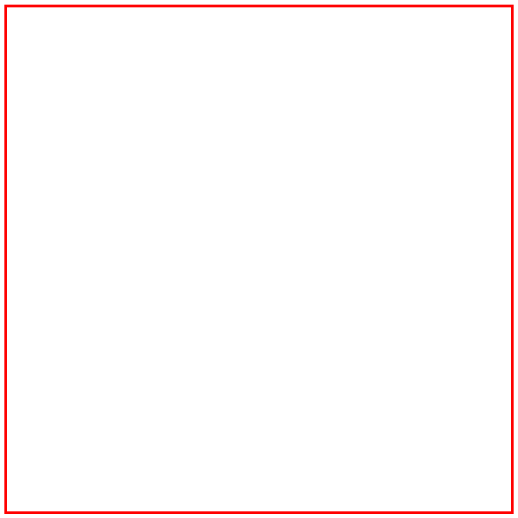
PPDrawingGroup

Please see Office Art File Format Documentation for PPDrawingGroup info.



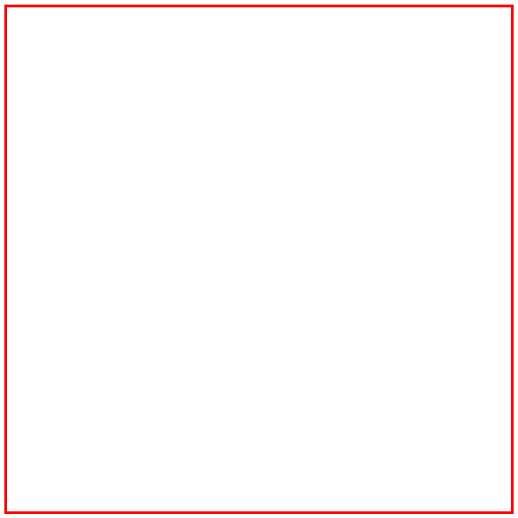
PPDrawing

Please see Office Art File Format Documentation for PPDrawing info.



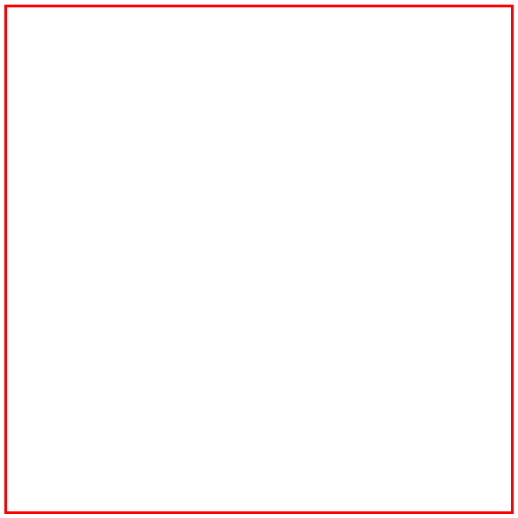
FontCollection (2005)

A container, that holds information about all the fonts in the presentation.



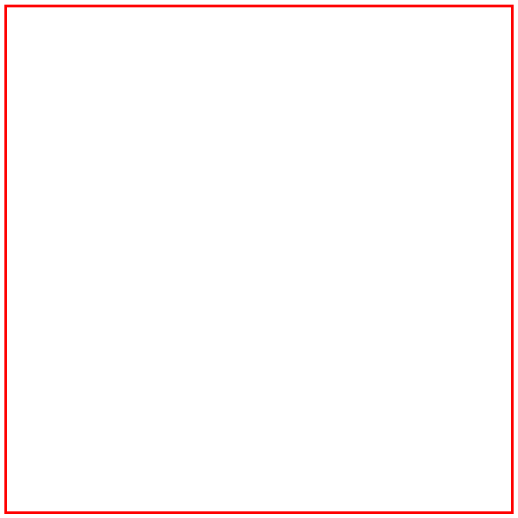
SoundCollection (2020) & Instance Sounds (41)

Is a container for all sound related atoms and containers. It contains one SoundCollAtom and one Sound container for each sound



ExObjRefAtom (3009)

This atom is saved from the OEShape container and refers to external objects that are serialized in the ExObjList. Type: Uint4



ExOleObjAtom (4035)

Atom that stores information for OLE objects.

ExOleObjAtom Fields

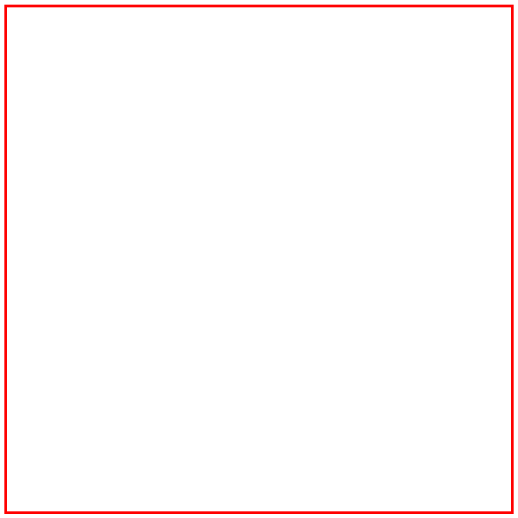
Offset	Type	Name	Contents
0	uint4	drawAspect	Stores whether the object can be completely seen (value of 1), or if only the icon is visible (value of 4).
4	sint4	type	Specifies whether the object is embedded or linked. Valid values are: 0 - embedded 1 - linked
8	sint4	objID	Unique identifier for the OLE object
12	sint4	subType	This specifies the type of ole object. *See subType Values table below.*
16	sint4	objStgDataRef	Reference to persist object

17	bool1	isBlank	Set if the object's image is blank

SubType Values

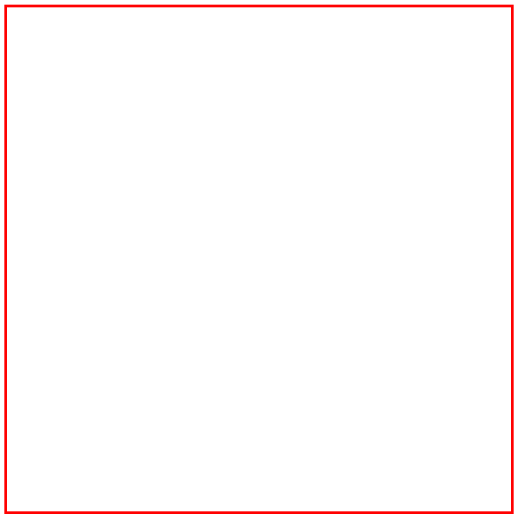
Value	Meaning
0	Default object
1	Microsoft Clipart Gallery
2	Microsoft Word table
3	Microsoft Excel
4	Microsoft Graph
5	Microsoft Organization Chart
6	Microsoft Equation Editor
7	Microsoft Wordart object
8	Sound
9	Imager
10	PowerPoint presentation
11	PowerPoint slide
12	Microsoft Project
13	Microsoft Note-It Ole
14	Microsoft Excel chart
15	Media Player object

ExOleObjAtom (4035)



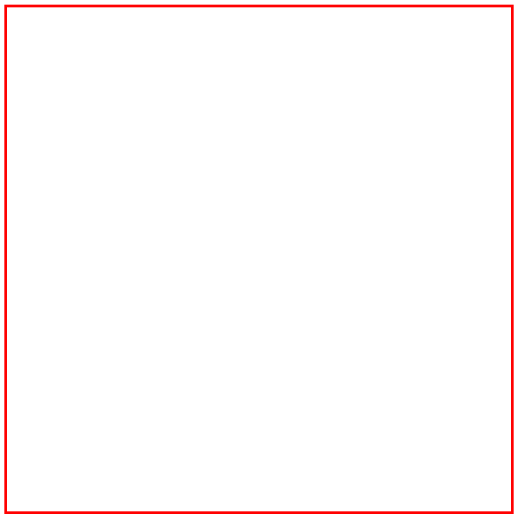
ExOleObjStg (4113)

A variable length container, which has LZW compressed data, which corresponds to the Istorage data for this ole object. The uncompressed data is a docfile, which contains the ole object data.



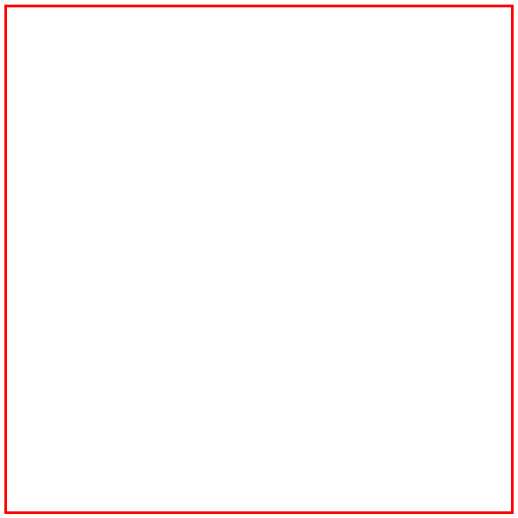
ExQuickTimeMovie (4074)

A container for Macintosh QuickTime movies. Quicktime movies are not supported on Windows, and cannot be played in PowerPoint 97 for Windows. They appear only as pictures, and are stored only for fidelity in round-tripping. The contents of this container are 1 or 2 ExQuickTimeMovieData objects.



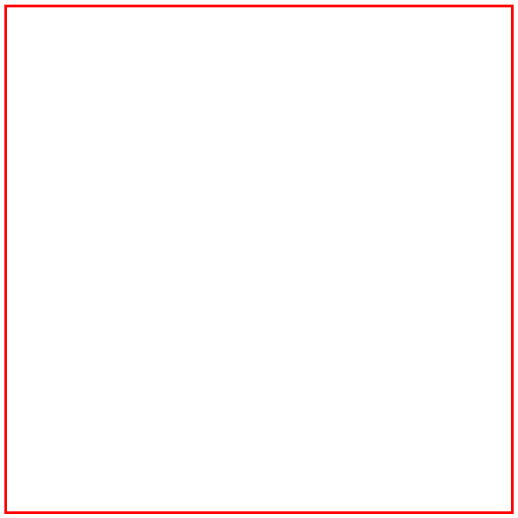
ExVideo (4101)

The contents of this container is a ExMeduaAtom plus a serialized CString, which is the path of the multimedia file.



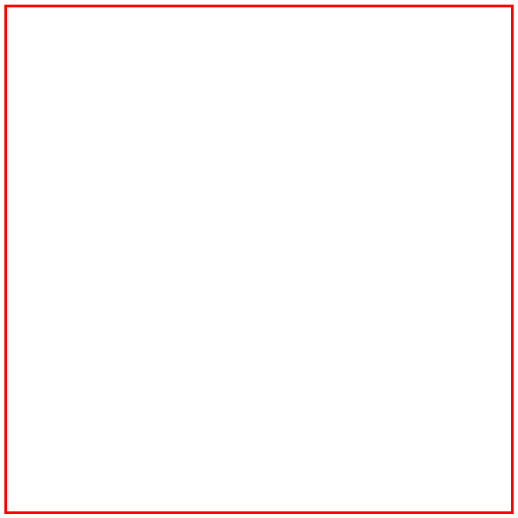
ExWAVAudioEmbedded (4111)

The ExWAVAudioEmbedded container consists of an ExMediaAtom record followed a ExWAVAudioEmbeddedAtom record.



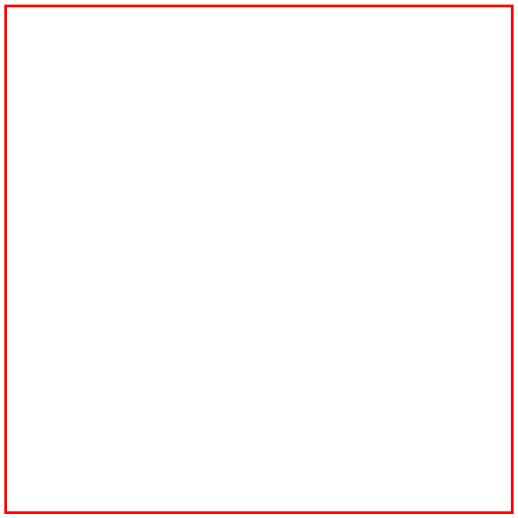
ExWAVAudioLink (4112)

The PST_ExWAVAudioLink container consists of an ExMediaAtom record followed by an a serialized CString which is the path to the audio file.



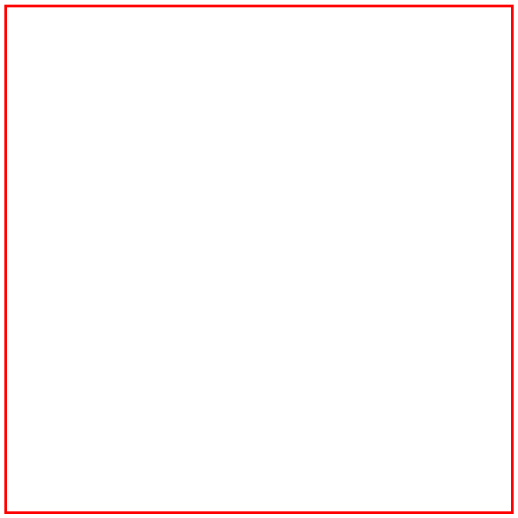
ExWAVeAudioEmbeddedAtom (4115)

The disk record for the ExWAVAudioEmbeddedAtom has 2 members, 'soundId' (which is a persistent reference to an object in the sound collection) and 'soundLength', which is the length of the sound clip in milliseconds.



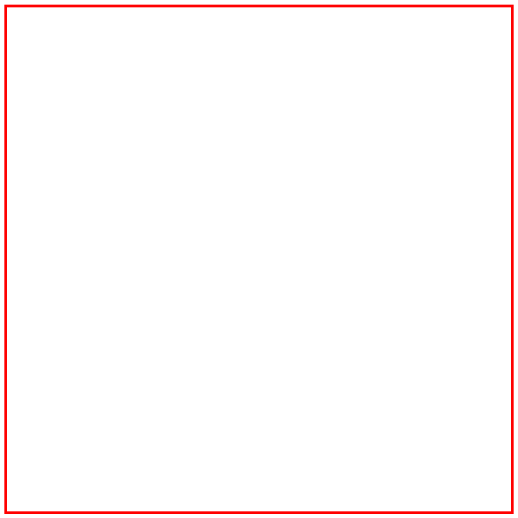
FontEmbedData (4024)

Font embedding data used internally by PowerPoint to save the fonts embedded in a presentation.



FontEntityAtom (4023)

This atom corresponds exactly to a Windows Logical Font (LOGFONT) structure. It keeps all the information needed to define the attributes of a font, such as height, width, etc. For more information, consult the Windows API Programmer's reference.

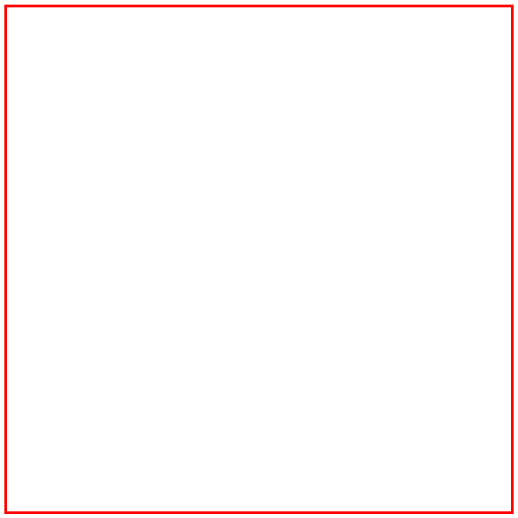


FooterMCAtom (4090)

FooterMCAtom is a record that stores the position of the footer meta character in the text. It needs no more information because this meta character is replaced by the footer string stored in the header and footer structure of the slide. The FooterMCAtom is only used in the footer placeholder on the slide, title, notes, and handout masters.

FooterDateMCAtom's fields

Offset	Type	Name	Content
0	sint4	position	The position of the character in a text.

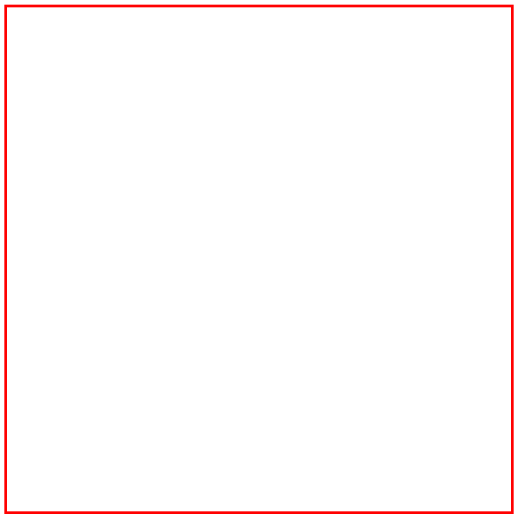


GenericDateMCAtom (4088)

GenericDateMCAtom is a record that stores the position of the generic date character in the text. It needs no more information because this meta character is replaced by the date string stored in the header and footer structure of the slide. The GenericDateMC is only used in one of the header and footer placeholders on slide, title, notes, and handout masters.

GenericDateMCAtom's fields

Offset	Type	Name	Content
0	sint4	position	The position of the character in a text.

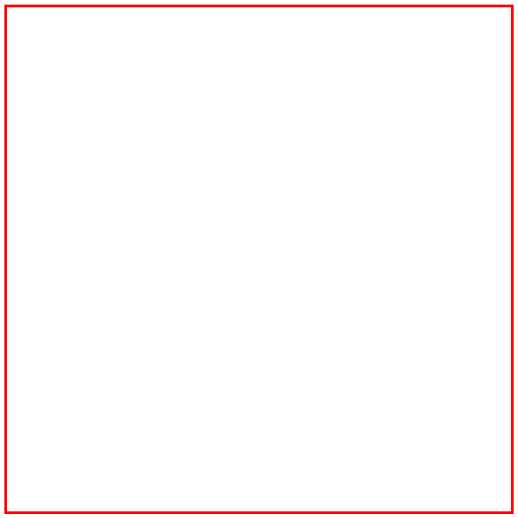


GPointAtom: Point Atom (3034)

This atom keeps the master coordinates of a point.

GPointAtom Fields

Offset	Type	Name	Contents
0	sin4	x	x coordinates
4	sin4	y	y coordinates



GRatioAtom: Ratio Atom (3031)

A Ratio Atom keeps the ratio of one quantity to another.

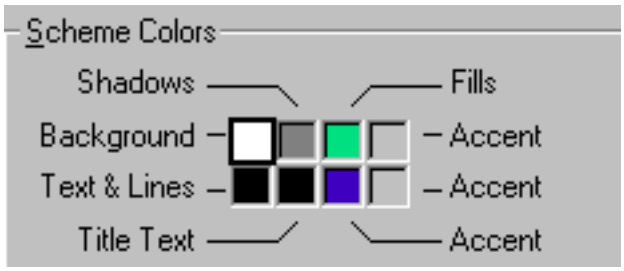
GPointAtom Fields

Offset	Type	Name	Contents
0	sin4	numer	Numerator
4	sin4	denom	Denominator

GRColorAtom: (10002)

This atom does not occur in the file by itself, but it occurs inside other atoms. It contains an index into the Scheme Collection or an RGB color as indicated by its index field.

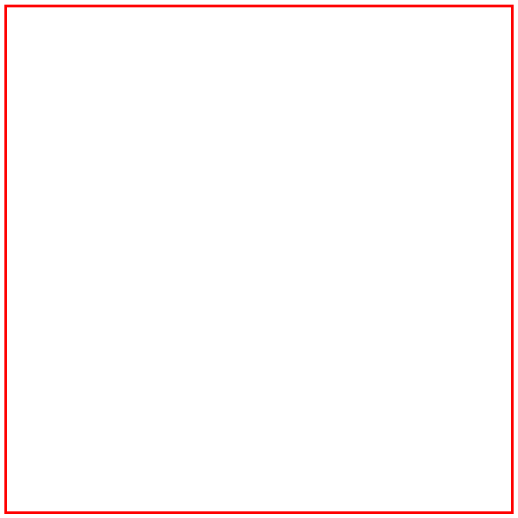
GRColorAtom Fields

Offset	Type	Name	Contents
0	ubyte1	red	Red value (0 - 255)
1	ubyte1	green	Green value (0 - 255)
2	ubyte1	blue	Blue value (0 - 255)
3	ubyte1	index	If this field has a value of OxFE, then the color is an RGB value. If not, it contains an index into the color scheme, with each value describing a color in the Scheme Colors dialog :  See Scheme Colors table below for valid values.

			This field can have a value of 0xFF if the color is undefined.

Scheme Color Values

Value	Meaning
0	Background
1	Text and Lines
2	Shadows
3	Title Text
4	Fills
5	Accent 1
6	Accent 2
7	Accent 3

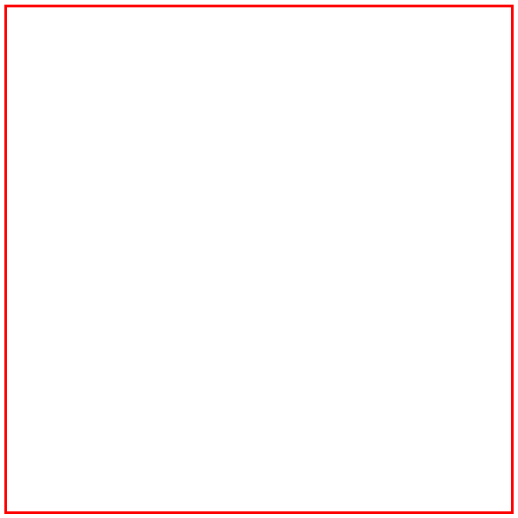


GScalingAtom (10001)

This atom does not occur in a file by itself, but it occurs inside other atoms. It represents a scale using two ratios.

GScalingAtom Fields

Offset	Type	Name	Contents
0	PSR_GRatioAtom	x	Numerator
8	PSR_GRatioAtom	y	Denominator

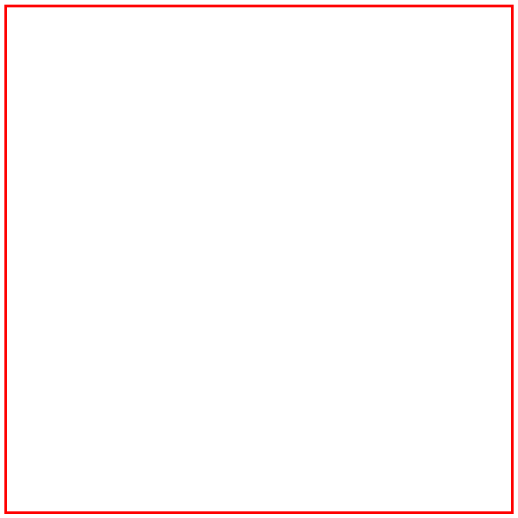


Handout (4041)

This is a container that keeps the information about the handout master.

Office Art drawing

Scheme container for the handout's color scheme

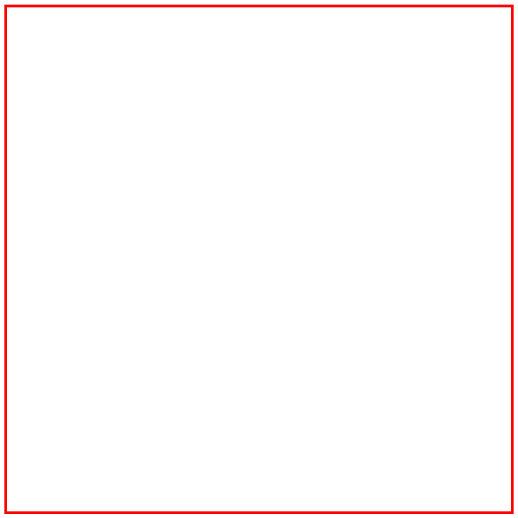


HeaderMCAtom (4089)

HeaderMCAtom is a record that stores the position of the header meta character in the text. It needs no more information because this meta character is replaced by the header string stored in the header and footer structure of the slide. The HeaderMCAtom is only used in the header placeholder on the slide, title, notes, and handout masters.

HeaderDateMCAtom's fields

Offset	Type	Name	Content
0	sint4	position	The position of the character in a text.



HeadersFooters (4057)

A container for:

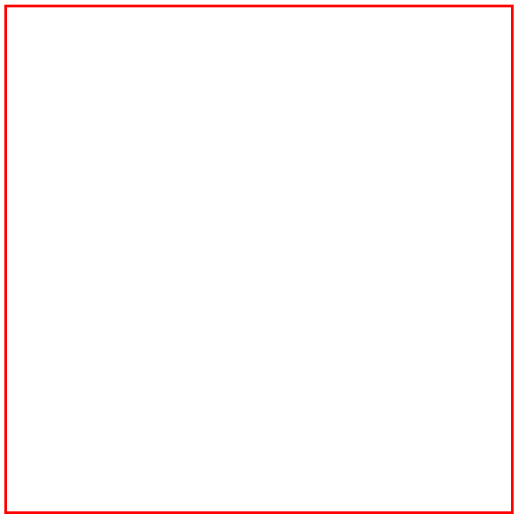
A HeadersFootersAtom.

CString that stores the user's date. This is the date that the user wants in the footers, instead of today's date.

CString that stores the Header's contents.

CString that stores the Footer's contents.

The Cstrings are optional if empty.



HeadersFootersAtom (4058)

HeadersFootersAtom stores the basic information of the header and footer structure. It tells what format of the updated date is and tells the following information.

1) Is the date on for this slide.

2) Are we using a user defined date or are we using the updated date.

3) Is the slide number on for this slide.

4) Is the header on for this slide.

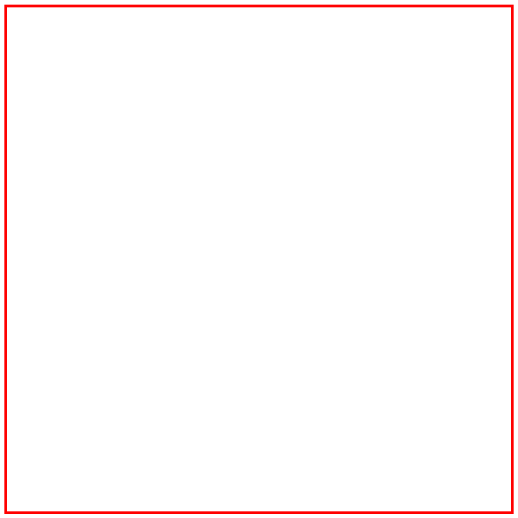
5) Is the footer on for this slide.

HeadersFooterDateAtom's fields

Offset	Type	Name	Content
0	sint2	FormatId	one of the 13 possible formats for the date. 0-12. See the Date and Time Dialog for details.
1	uint2	Flags	what is turned on. date, todayDate, userDate, slideNumber, header, footer

Example ....

```
const int S_HEADERFOOTER_DATE = 0x01;

const int S_HEADERFOOTER_TODAYDATE = 0x02;

const int S_HEADERFOOTER_USERDATE = 0x04;

const int S_HEADERFOOTER_SLIDENUMBER = 0x08;

const int S_HEADERFOOTER_HEADER = 0x10;

const int S_HEADERFOOTER_FOOTER = 0x20;

m_formatId = (unsigned char) rec.formatId;

m_date = ( (rec.flags & S_HEADERFOOTER_DATE) != 0 );

m_todayDate = ( (rec.flags & S_HEADERFOOTER_TODAYDATE) != 0 );

m_userDate = ( (rec.flags & S_HEADERFOOTER_USERDATE) != 0 );

m_slideNumber = ( (rec.flags & S_HEADERFOOTER_SLIDENUMBER) != 0 );

m_header = ( (rec.flags & S_HEADERFOOTER_HEADER) != 0 );

m_footer = ( (rec.flags & S_HEADERFOOTER_FOOTER) != 0 );
```



InteractiveInfo (4082)

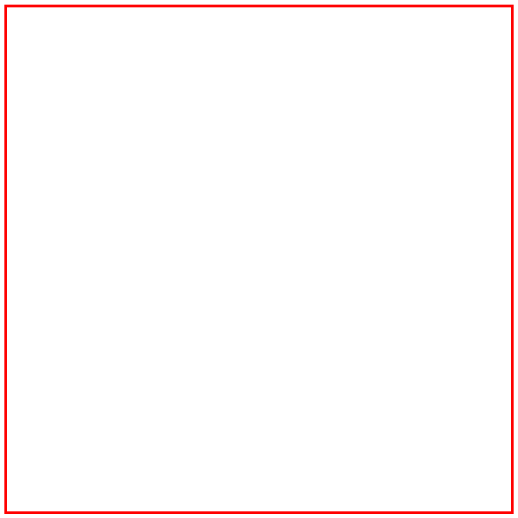
Interactive settings for mouse-over and mouse-down on an object in slideshow. Contains:

PST_InteractiveInfoAtom

CString macro name (may be empty)

PST_Sound (only when serializing to Clipboard)

PST_ExHyperLink (only when serializing to Clipboard)



InteractiveInfoAtom (4083)

Interactive settings for mouse-over and mouse-down on an object in slideshow

InteractiveInfoAtom Fields

Offset	Type	Name	Contents
0	uint4	SoundRef	a reference to a sound in the sound collection, or NULL.
4	uint4	ExHyperlinkID	a persistent unique identifier to an external hyperlink object (only valid when action == HyperlinkAction).
8	Ubyte1	Action	See Action Table
9	ubyte1	OleVerb	Only valid when action == OLEAction. OLE verb to use, 0 = first verb, 1 = second verb, etc.
10	ubyte1	Jump	See Jump Table

11	ubyte1	Flags	Bit 1: Animated. If 1, then button is animated Bit 2: Stop sound. If 1, then stop current sound when button is pressed. Bit 3: CustomShowReturn. If 1, and this is a jump to custom show, then return to this slide after custom show.
12	ubyte1	HyperlinkType	a value from the LinkTo enum, such as LT_URL (only valid when action == HyperlinkAction).

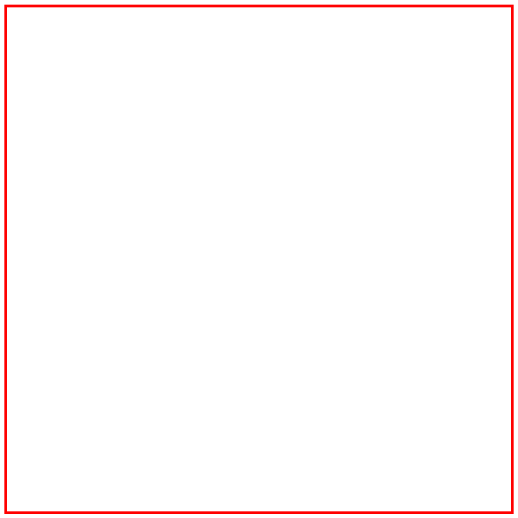
Action Table:

Action	**Value**
NoAction	0
MacroAction	1
RunProgramAction	2
JumpAction	3
HyperlinkAction	4
OLEAction	5
MediaAction	6
CustomShowAction	7

Jump Table:

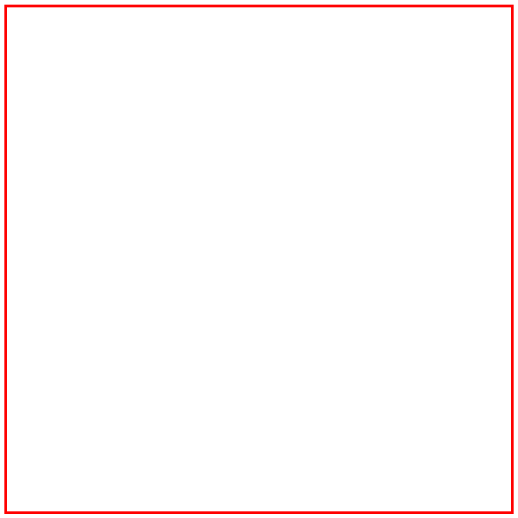
Jump	**Value**

NoJump	0
NextSlide,	1
PreviousSlide,	2
FirstSlide,	3
LastSlide,	4
LastSlideViewed,	5
EndShow	6



MetaFile (4033)

This is an atom that occurs inside an ExEmbed or an ExLinkcontainer and is used for icons for linked or embedded OLE objects only. It contains a picture in a presentation stored as a 16-bit Windows metafile.



OEPlaceholderAtom (3011)

Atom that describes the placeholder.

OEPlaceholderAtom Fields

Offset	Type	Name	Contents
0	uint4	placementId	The placement Id is a number assigned to the placeholder. It goes from -1 to the number of placeholders. *See note below.*
4	ubyte1	PlaceholderId	Type of placeholder. *See the Placeholder ID Values table below for valid values.*
1	ubyte1	size	Size of the placeholder, which can be: 0 - full size 1 - half size 2 - quart of the slide

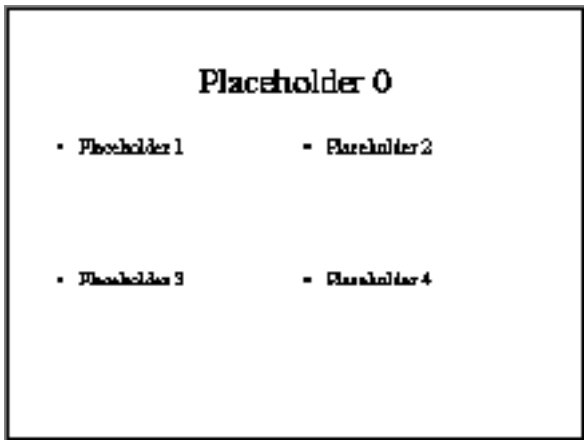
Placeholder ID Values

Value	Type of Placeholder

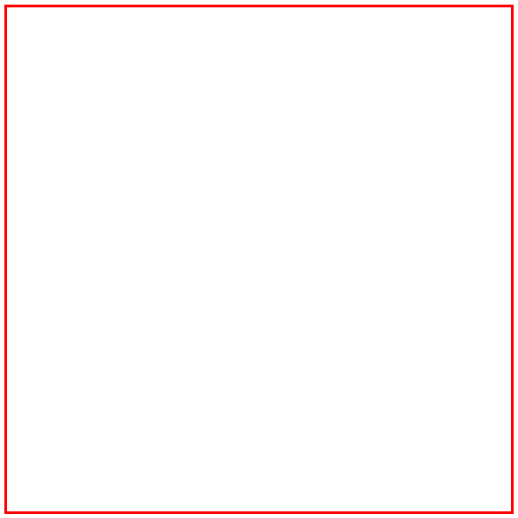
0	None
1	Master title
2	Master body
3	Master centered title
4	Master notes slide image
5	Master notes body image
6	Master date
7	Master slide number
8	Master footer
9	Master header
10	Master subtitle
11	Generic text object
12	Title
13	Body
14	Notes body
15	Centered title
16	Subtitle
17	Vertical text title
18	Vertical text body
19	Notes slide image

20	Object (no matter the size)
21	Graph
22	Table
23	Clip Art
24	Organization Chart
25	Media Clip

Note: The placementId is given in order from top to bottom, right to left. As an example, if we used the 4 object slide layout, the placement Ids would be as in the following picture:



There is a special case when the placeholder does not have a position in the layout. This occurs when the user has moved the placeholder from its original position. In this case the placeholder ID is -1.

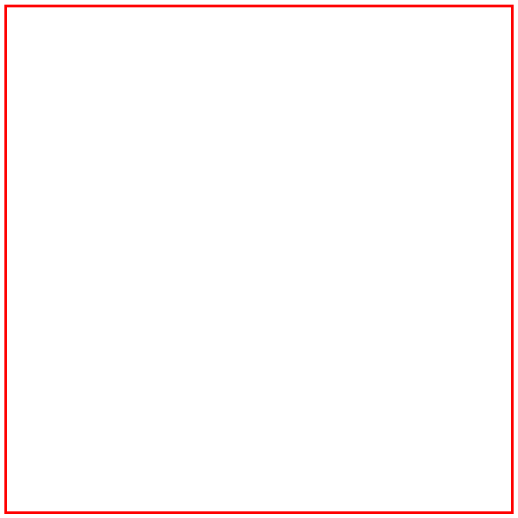


OEShapeAtom (3009)

Atom that contains information that describes a shape client data.

OEShapeAtom Fields

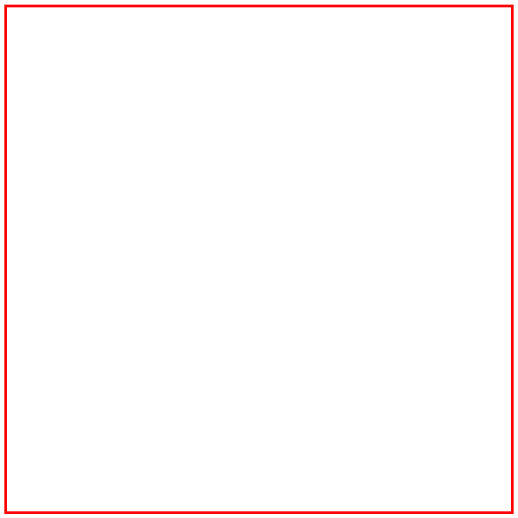
Offset	Type	Name	Contents
0	ubyte1	flags	Bit 1: ialways on top



OutlineTextRefAtom

Appears in a slide to indicate a text that is already contained in the document, in a PST_SlideListWithText containter. Sometimes slide texts are not contained within the slide container to be able to delay loading a slide and still display the title and body text in outline view.

Offset	Type	Name	Contents
0	Sint4	Index	the text's index within the slide (0 for title, 1..n for the nth body)



ParaFormatAtom : Paragraph Format Atom (4067)

This atom keeps a paragraph's formatting. Paragraph data falls into 3 categories:

Bullet Data Fields

Offset	Type	Name	Contents
0	uint2	buHasBullet : 2	Set if the paragraph has a bullet.
	uint2	buHasTypeface : 2[1]	Set if the bullet has a typeface different than the one used by the text.
	uint2	buHasColor : 2	Set if the bullet has different color than the text.
	uint2	buHasSize : 2	Set if the bullet has different size than the text.
1	1 byte padding	padding	Padding
2	GrColorAtom	buColor	RGB color for the bullet
6	uint2	buChar	If the bullet uses a special character, it's stored here.
8	sint2	buSBCTypeface	Single byte typeface reference
10	sint2	buDBCTypeface	Double byte typeface reference [1]

12	sint4	buSize	Size of the bullet

[1] If the buHasTypeface field is set then the Typeface for the bullet can be a Single Byte Character typeface (SBC) or a Double Byte Character (DBC). If the value of the buDBCtypeface is 2 then the value that is used for the bullet is the buSBCtypeface. If the buDBCtypeface is not 2, then the reference to the typeface is obtained by subtracting three from the value in the file.

Other Format Data Fields

Offset	**Type**	**Name**	**Contents**
16	sint4	pfLeftMargin	Indent of lines besides the first one in the paragraph
20	sint4	pfRightMargin	Inset from the right
24	sint4	pfIndent	First line indent from the left
28	sint4	pfAlignment	Alignment for the paragraph: The values can be: 0 - left aligned 1 - centered 2 - right aligned 3 - justified
32	sint4	pfLineSpacing	Fixed or automatic line spacing
36	sint4	pfSpaceBefore	Space before each paragraph
40	sint4	pfSpaceAfter	Space after each paragraph
44	sint4	pfTabCount	Number of tab stops

Features for the Japanese Language

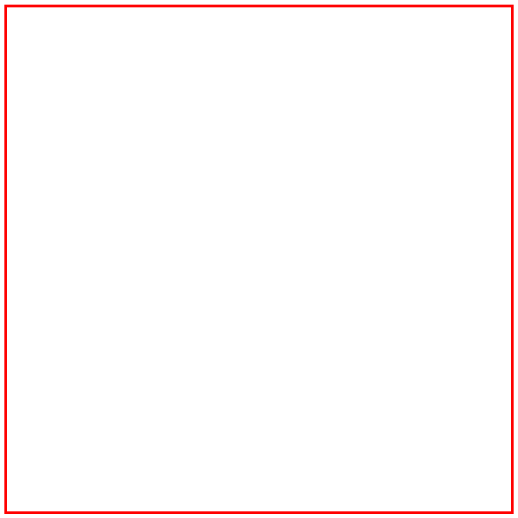
Three out of these four flags deal with the way certain characters, that can not exist at the beginning of a new line (i.e "?", ")"), are handled.

Offset	Type	Name	Contents
48	ubyte1	pfBaseLine[2]	Enumerated value that states the alignment of one letter with respect to the other [2]
49	ubyte1	pfCharWrap	If set, the character preceding the unwanted one goes to the next line. This character becomes the first one in the new line and the unwanted character becomes the second one.
50	ubyte1	pfWordWrap	If set, the paragraph has wordwrap.
51	ubyte1	pfOverflow	If set, the unwanted character is put at the end of the line where it doesn't fit, so it goes past the right margin.

[2] For the pfBaseLine there are three different options: roman, hanging and centered.

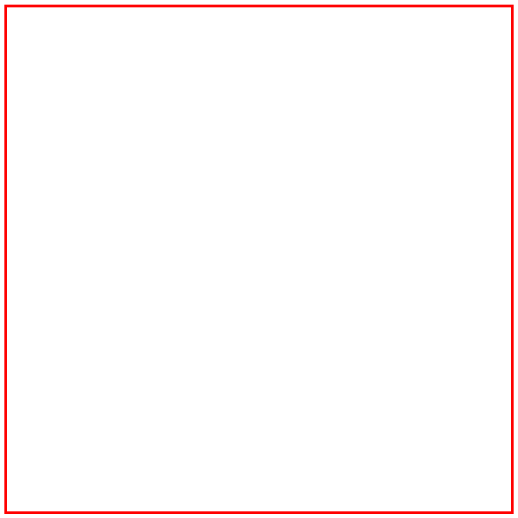
pfBaseLine Options

Style	Value	Example
Roman	0	
Hanging	1	
Centered	2	



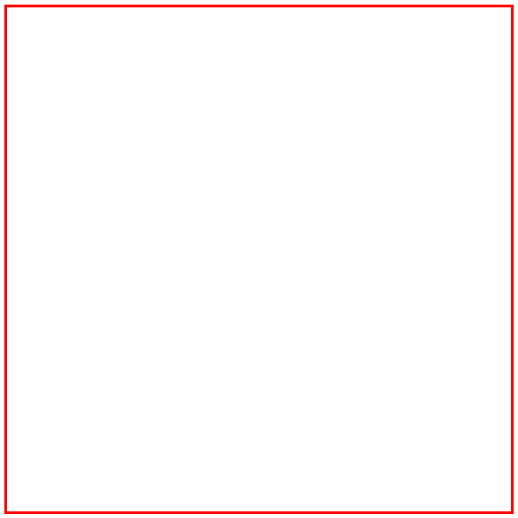
PersistPtrFullBlock

Complete list of persist's for this version. (For more information, see UserEditAtom Element Descriptions)



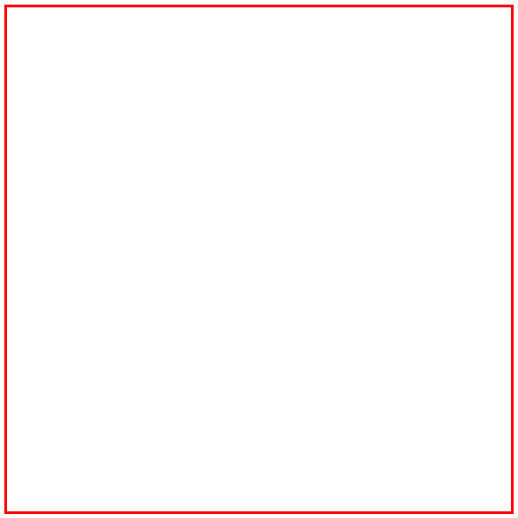
PersistPtrIncrementalBlock

Incremental diffs on persists. (For more information, see UserEditAtom Element Descriptions)



SoundCollAtom (2021)

ObjectIdSeed: next unique identifier for external objects



Sound (2022)

Sound consists of:

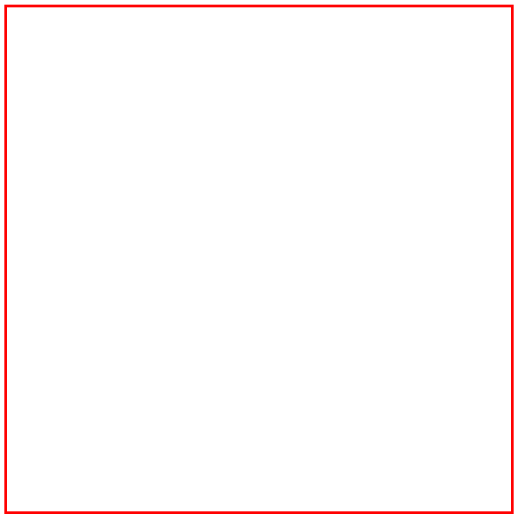
PST_CString (instance 0): name of sound (e.g. "crash")

PST_CString (instance 1): type of sound (e.g. ".wav")

PST_CString (instance 2): reference id of sound in sound collection

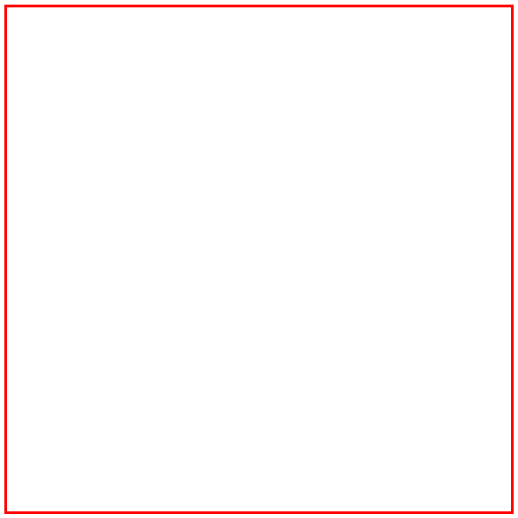
PST_CString (instance 3): built-in id of sound, for sounds we ship. This is the id that's in the reg file.

PST_SoundData



SoundData (2023)

Variable number of bytes. This is the sound file.



TextHeaderAtom (3999)

Appears in the beginning of a series of atoms belonging to the same text.

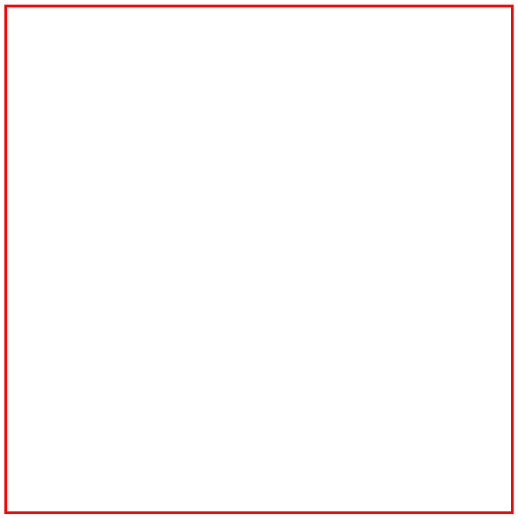
TextHeaderAtom Fields

Offset	Type	Name	Contents
0	uint4	TxType	Type of text. *See the Text Type table below.*

Text Types

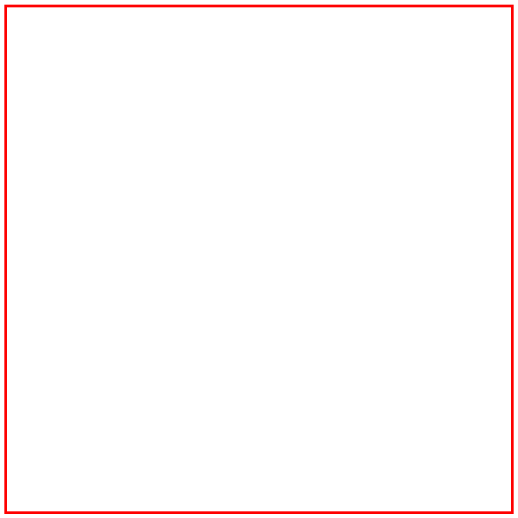
Flag	Meaning
0	Title
1	Body
2	Notes
3	Not Used
4	Other (Text in a shape)
5	Center body (subtitle in title slide)

6	Center title (title in title slide)
7	Half body (body in two-column slide)
8	Quarter body (body in four-body slide)



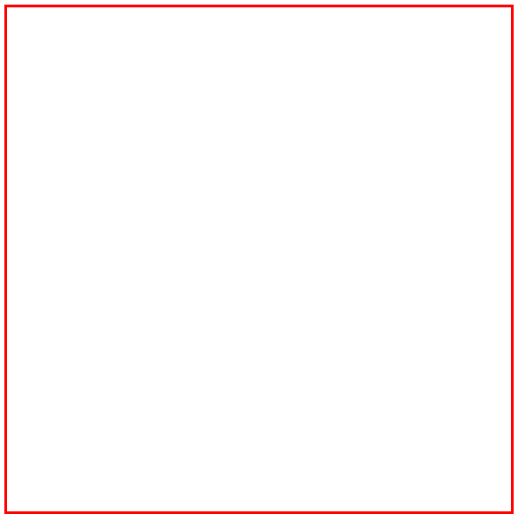
TextCharsAtom (4000)

Unicode characters of the text without trailing return character. The byte size of the atom is double the length of the text, minus one for the trailing return character.



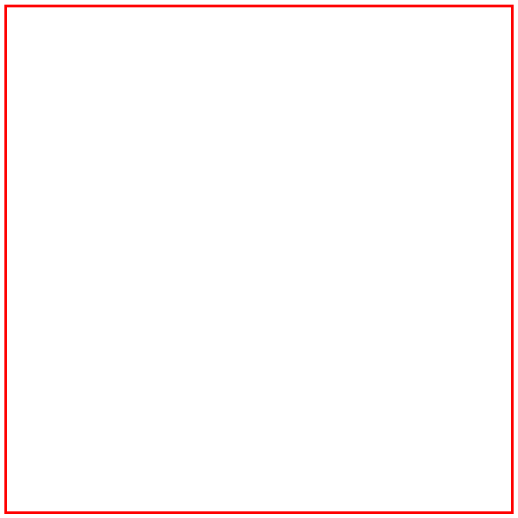
StyleTextPropAtom (4001)

The character properties (such as bold, italic, font, color etc.) and paragraphs properties (alignment, line spacing etc.). Organized in two run lists (one for character and paragraph properties) specifying differences to the style. Special parsing code is needed to parse content of this atom.



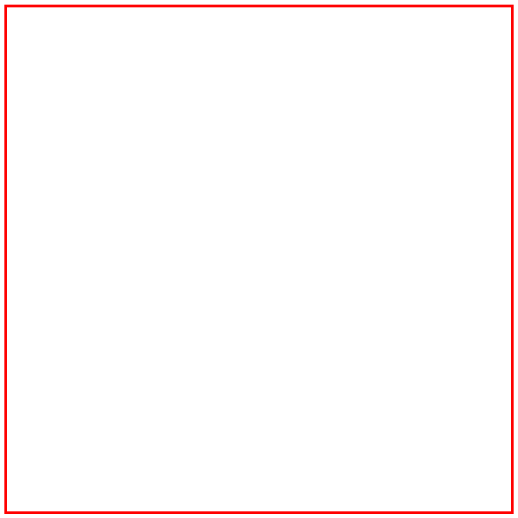
TxMasterStyleAtom (4003)

PowerPoint text styles. The atom instance value is the text type and is encoded like the *txstyle* field in PST_TextHeaderAtom. The text styles are located in the PST_MainMaster container, except for the "other" style, which is in the PST_Environment container. Special parsing code is needed to parse content of this atom.



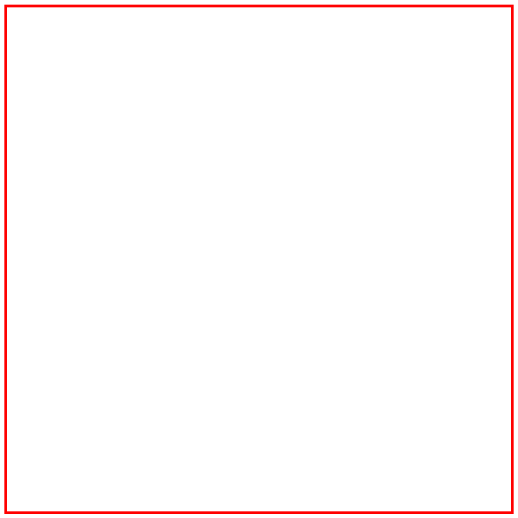
TxCFStyleAtom (4004)

Single character property container. Storing differences to a style. Used only within PST_Environment container the to store text default character properties for new texts. Special parsing code is needed to parse content of this atom.



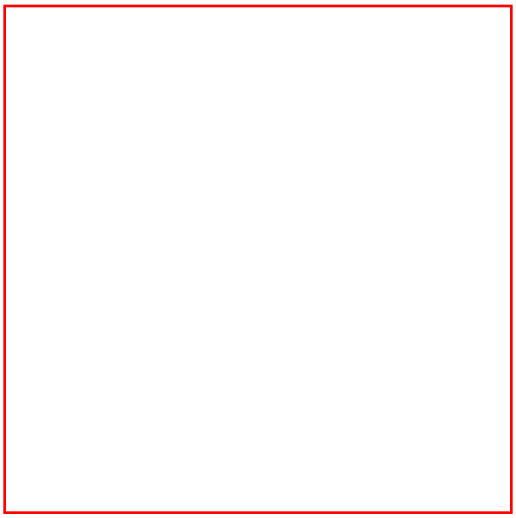
TxPFStyleAtom (4005)

Single paragraph property container. Storing differences to a style. Used only within PST_Environment container the to store text default paragraph properties for new texts. Special parsing code is needed to parse content of this atom.



TextRulerAtom (4006)

Ruler of a text if it differs from the style's ruler settings. Special parsing code is needed to parse content of this atom.

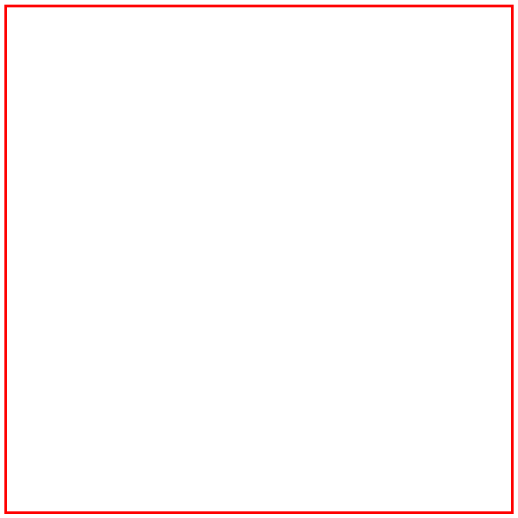


TextBookmarkAtom (4007)

Bookmark ("property") within text

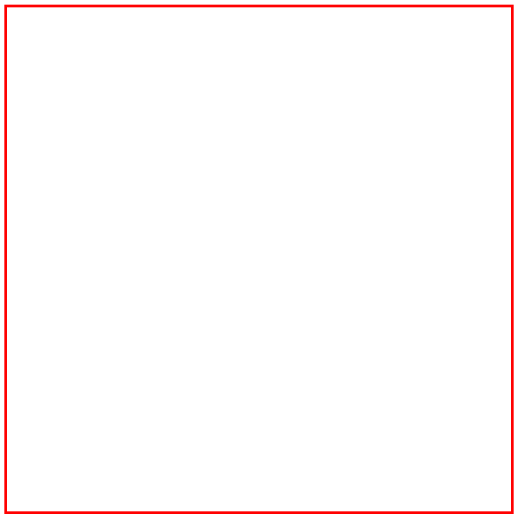
TextBookmarkAtom Fields

Offset	Type	Name	Contents
0	uint4	Begin	Beginning character position
4	uint4	End	End character position
8	uint4	bookmarkID	Bookmark ID



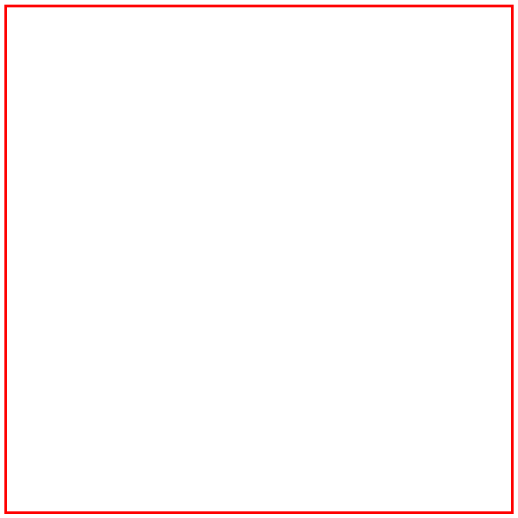
TextBytesAtom (4008)

Byte values of characters in the text if all characters (without trailing return character) of the text are < 256. The byte size of the atom is equal the length of the text, minus one for the trailing return character.



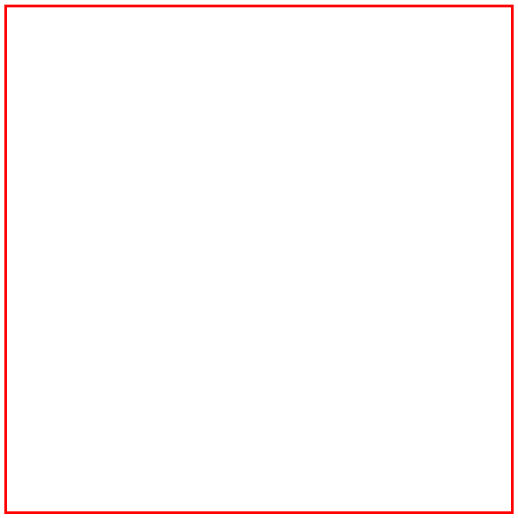
TxSIStyleAtom (4009)

Single special info container. Used only within PST_Environment container to store default special info (see PST_TextSpecInfoAtom for a definition of "special info") for new texts. Special parsing code is needed to parse content of this atom.



TextSpecInfoAtom (4010)

The special info runs contained in this text. "Special infos" are character properties that don't follow styles, such as background spelling info or language ID. Special parsing code is needed to parse content of this atom.



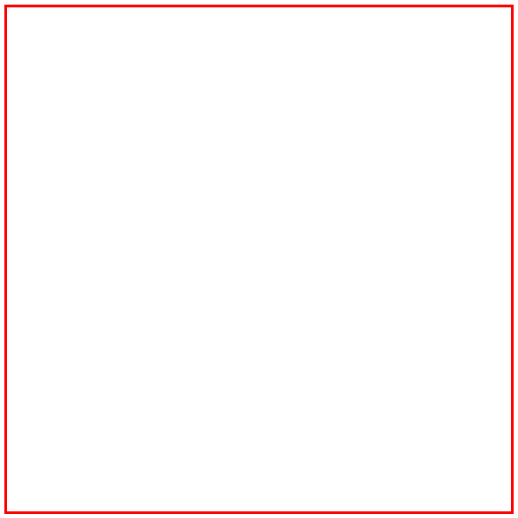
SrKinsoku (4040)

A container for the Japanese word wrap feature. It contains:

A SrKinsokuAtom that contains the level of kinsoku.

A CString and Instance Leading (4) that keeps the punctuation that is forbidden at the end of the line.

A CString and Instance Following (5) that keeps the punctuation that is forbidden at the beginning of the line.



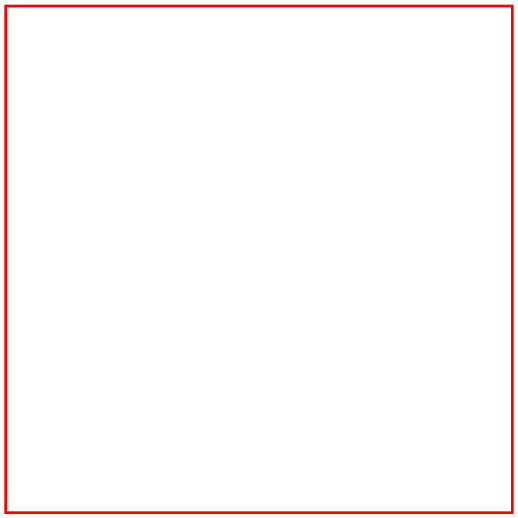
SrKinsokuAtom (4050)

Atom that keeps in a four-byte signed integer the Kinsoku level that is displayed on the Kinsoku dialog. The level can be:

Level 1: value of 0

Level 2: value of 1

Custom Level: value of 2



PrintOptions (6000)

PrintOptions is a record that stores default settings for how the PowerPoint presentation should be printed.

PrintOptions Fields

Offset	Type	Name	Contents
0	ubyte1	PrintWhat	What to print by default when printing the presentation. Valid values are from 0-6. See PrintWhat field values table below for valid values.
1	ubyte1	ColorMode	Default color mode to use when printing the presentation. Valid values are from 0-2. See ColorMode field values table below for valid values.
2	bool1	PrintHidden	True if hidden slides should be printed by default when printing the presentation.
3	bool1	ScaleToFitPaper	True if presentation should be scaled to fit paper when printing, by default.
4	bool1	FrameSlides	True if slides should be framed by default when printing the presentation.

PrintWhat Field Values

Value Meaning

0 slides (without animations, if any are present)

1 slides (with animations, if any are present)

2 handouts (2 slides per page)

3 handouts (3 slides per page)

4 handouts (6 slides per page)

5 notes pages

6 outline view

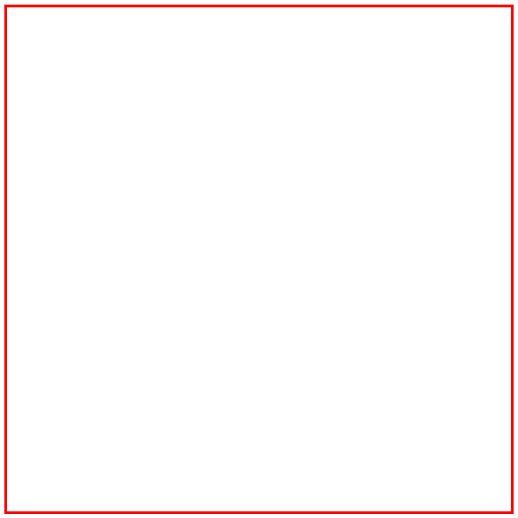
ColorMode Field Values

Value Meaning

0 black and white

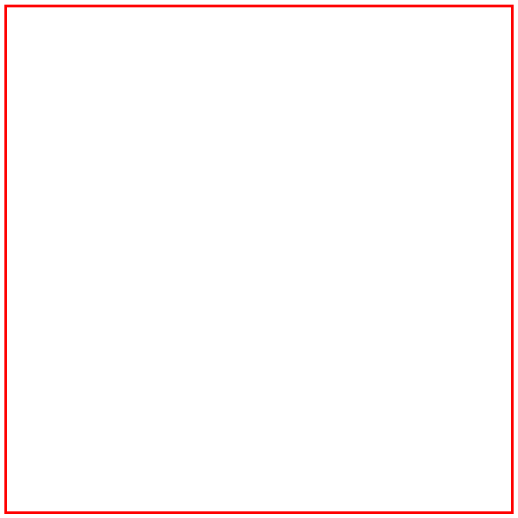
1 gray-scale

2 color



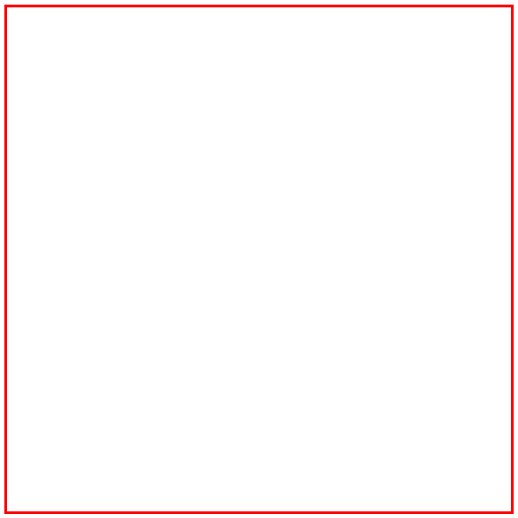
ProgBinaryTag

A name/value pair within a given content object. The next things in the file are the tag name (INS_TagName--a CString) followed by a PST_BinaryTagData.



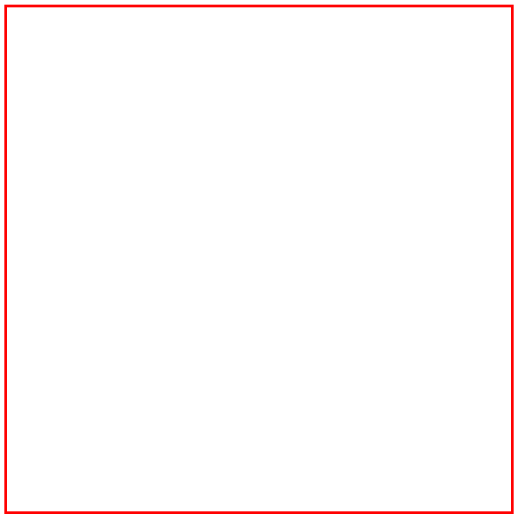
ProgStringTag

A name/value pair within a given content object. It indicates that the next two things in the file are strings--first the tag name (INS_TagName), followed by the tag value (INS_TagValue). These are both CStrings.



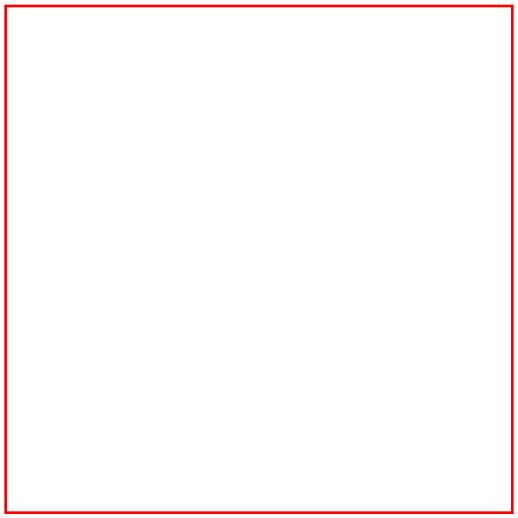
ProgTags (5000)

Information used internally by PowerPoint.



RecolorInfoAtom (4071)

This atom keeps the recoloring information used internally by PowerPoint to recolor pictures.

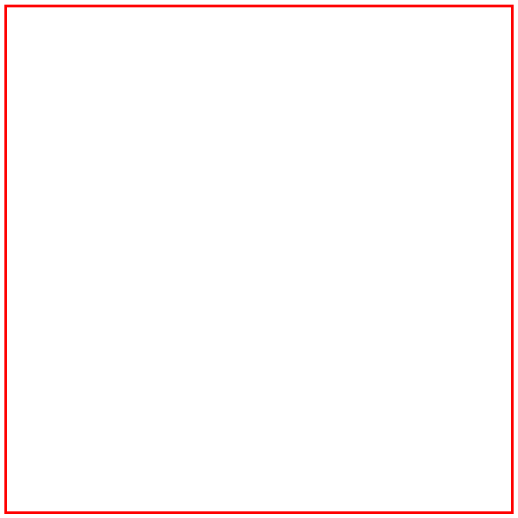


RTFDateTimeMCAtom (4117)

RTFDateTimeMCAtom is a 64 uint2 long string for storing a Word-type field string that describes a date or time. For more information about Word's field strings, consult the Word Technical Reference. The field string is parsed and turned into a metacharacter like SlideNumberMCAtom.

RTFDateTimeMCAtom fields

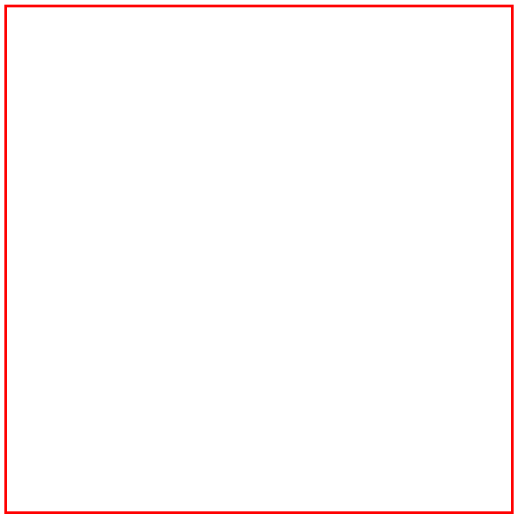
Offset	Type	Name	Content
0	sint4	position	The position of the character in a text.
4	uint2[64]	format	The field string



SlideListWithText

Container that contains the title and body texts of all the slides in the presentation. The actual placeholder shapes in the slides can then use a PST_OutlineTextRefAtom to reference these texts instead of containing them again.

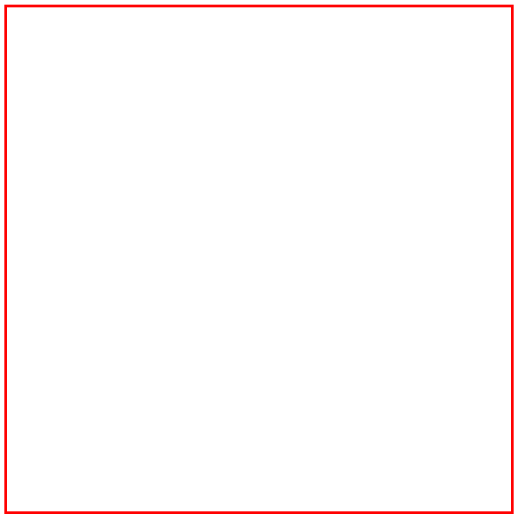
Each slide within the PST_SlideListWithText starts with a PST_SlidePersistAtom, and each text within these slides with a PST_TextHeaderAtom.



SlideNumberMCAtom (4056)

SlideNumberMCAtom is a record that stores the position of the slide number meta character in the text. The slide number meta character is replaced by the current slide number during PowerPoint's runtime.

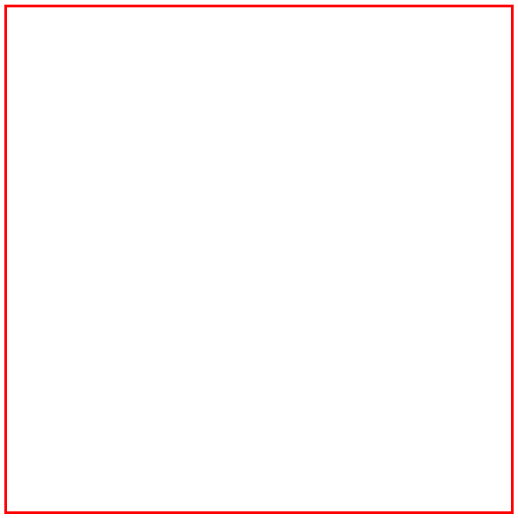
Offset	Type	Name	Content
0	sint4	position	The position of the character in a text.



TxInteractiveInfoAtom (4063)

An interactive info in a text. These atoms always follow a corresponding PST_InteractiveInfo atom containing the actual interactive info data.

Offset	Type	Name	Contents
0	uint4	begin	Beginning character position
4	uint4	end	Ending character position



UserEditAtom (4085)

See UserEditAtom in "Current User Stream" section.

Offset	Type	Name	Contents
0	sint4	LastSlideID	Id of slide currently selected in view
4	uint4	Version	Major and minor app version that did the save
8	Uint4	OffsetLastEdit	File offset of UsereditAtom of the previous incremental save. 0 after a full save
12	Uint4	OffsetPersistDirectory	File offset to persist pointers for this save operation
16	Uint4	DocumentRef	Persist reference to the document persist object
20	Uint4	MaxPersistWritten	Seed value for persist object id management
24	Sint2	LastViewType	View type see table below

View Types

Flag	Meaning

0	None
1	SlideView
2	OutlineView
3	Notes