Developing new transport stream formats

Notice that the operation of transports and transport stream formats is rather complex. Thus this document will probably need to be read over and over again in combination with studying the kbmMW source code to give the full insight into how the transport stream format and transports operate together.

One of the forces of kbmMW is the very flexible infrastructure which allows for adding new features. One of these features is called a 'transport stream format'.

A transport stream format has the responsibility to - as the name says - format and interpret the data being send between a client and a server via any transport. A transport is for example the TkbmMWTCPIPIndyTransport.

Since communication between a client and the server is request/response based, there are two parts of a transport stream format, a request formatter/interpreter based on *TkbmMWCustomRequestTransportStream* and a response formatter/interpreter based on *TkbmMWCustomResponseTransportStream*. Both inherit from *TkbmMWCustomTransportStream* and are defined in the file **kbmMWCustomTransport.pas**.

TkbmMWCustomTransportStream defines many methods that can be overridden by other child classes to tailor the stream format. By default the main class contains streaming facilities for binary format.

The most used methods will be listed here with samples of how the STANDARD streaming protocol has been implemented.

Methods that must always be overridden	
`class function GetIdentifier:string; virtual;`	Should return the unique identifier of the streamformat. For example 'STANDARD'. No other registered transport streamformats should use same identifier.
`class function GetVersion:string; virtual;`	Should return the identifier of the version of the streamformat. For example '100' which means v. 1.00.

Low level data type methods.

Methods that handles basic data types. These methods can optionally be overridden. Its usually a good idea to override all low level data type methods instead of only one to have a consistent format. Standard format used is binary. These data type methods are used by higher level methods and thus its often only needed to create your own implementation of the low level data type methods to implement a new streamformat. All operations should store or load from the DataStream stream after making sure that the stream is ok via the CheckStream method. The name parameter passed to all methods can optionally be used as part of the streaming process. This is interesting for example in XML streaming situations where attributes usually are given a name.

In all examples the TYourTransportStream refers to your custom RequestTransportStream and ReponseTransportStream. Notice that custom streamformat classes go in pairs.

Description	Example

WriteString stores a string in the DataStream. Override it to store the string in a format of your own.
 	
`procedure WriteString(`
`  const AName:string;`
`  const AString:string); virtual;`
	`procedure TYourTransportStream.WriteString(`
	`  const AName:string;`
	`  const AString:string);`
	`var`
	`  l:integer;`
	`begin`
	`  CheckStream;`
	`  l:=length(AString);`
	`  WriteCount(AName+'_SIZE',l);`
	`  DataStream.Write(PChar(AString)^,l);`
	`end;`

WriteInteger stores an integer in the DataStream. Override it to store the integer in a format of your own.
 	
`procedure WriteInteger(`
`  const AName:string;`
`  const AInteger:integer); virtual;`
	`procedure TYourTransportStream.WriteInteger(`
	`  const AName:string;`
	`  const AInteger:integer);`
	`var`
	`  buf:array [0..3] of byte;`
	`begin`
	`  CheckStream;`
	`  buf[0]:=byte(AInteger and $FF);`
	`  buf[1]:=byte((AInteger shr 8) and $FF);`
	`  buf[2]:=byte((AInteger shr 16) and $FF);`
	`  buf[3]:=byte((AInteger shr 24) and $FF);`
	`  DataStream.Write(Buf,4);`
	`end;`

Variants are represented by several write methods. The most low level one, used by the other methods is the WriteVariantByteArray. Override this to store all variants in a format of your own. Notice that variant streaming routines also use WriteString and WriteInteger.

```
procedure WriteVariantByteArray(
   const AName:string;
   const AVariant:Variant); virtual;

procedure WriteVariant(
   const AName:string;
   const AVariant:variant); virtual;

procedure WriteVariantArray(
   const AName:string;
   const AVariant:Variant); virtual;
```

```
procedure TYourTransportStream.WriteVariantByteArray(
  const AName:string;
  const AVariant:variant);
var
   l,h:integer;
   p:PChar;
begin
     CheckStream;

     // Check if only one dimension. If not complain.
     if VarArrayDimCount(AVariant)<>1 then
        kbmMWRaiseServerException(
          'WriteVariantByteArray writes arrays of one
dimension only.');
     WriteCount(AName+'_DIM',1);

     // Write dimension boundary.
     l:=VarArrayLowBound(AVariant,1);
     h:=VarArrayHighBound(AVariant,1);
     WriteRange(AName,l,h);

     // Create variant array.
     p:=VarArrayLock(AVariant);
     try
        DataStream.WriteBuffer(p^,h-l+1);
     finally
        VarArrayUnlock(AVariant);
     end;
end;
```

WriteStream store a stream into the DataStream. Override it to store the stream in your own format.

```
procedure WriteStream(
   const AName:string;
   AStream:TStream;
   ALength:integer); virtual;
```

```
procedure TYourTransportStream.WriteStream(
   const AName:string;
   AStream:TStream;
   ALength:integer);
begin
     CheckStream;
     WriteCount(AName+'_SIZE',ALength);
     DataStream.CopyFrom(AStream,ALength);
end;
```

ReadString returns a string from the DataStream. Override it to read the string in your own format.

```
function ReadString(
   const AName:string):string; virtual;
```

```
function TYourTransportStream.ReadString(
   const AName:string):string;
var
   l:integer;
begin
     CheckStream;
     l:=ReadInteger(AName+'_SIZE');
     SetLength(Result,l);
     DataStream.Read(PChar(Result)^,l);
end;
```

ReadInteger returns an integer value from the DataStream. Override it to read the integer in your own format.

```
function ReadInteger(
   const AName:string):integer; virtual;
```

```
function TYourTransportStream.ReadInteger(
   const AName:string):integer;
var
   buf:array [0..3] of char;
begin
     CheckStream;
     DataStream.Read(buf,4);
     Result:=byte(buf[0])
          + (byte(buf[1]) shl 8)
          + (byte(buf[2]) shl 16)
          + (byte(buf[3]) shl 24);
end;
```

Variants are represented by several read methods. The most low level one, used by the other methods is the ReadVariantByteArray. Override this to read all variants in a format of your own. Notice that variant streaming routines also use ReadString and ReadInteger.

```
function ReadVariantByteArray(
   const AName:string):variant; virtual;

function ReadVariant(
   const AName:string):variant; virtual;

function ReadVariantArray(
   const AName:string):variant; virtual;
```

```
function TYourTransportStream.ReadVariantByteArray(
   const AName:string):variant;
var
   d,l,h:integer;
   p:PChar;
begin
     CheckStream;
     d:=ReadCount(AName+'_DIM');
     if d<>1 then
       kbmMWRaiseServerException(
          'ReadVariantByteArray only supports one
dimension.');

     // Read dimension layout.
     ReadRange(AName,l,h);

     // Create variant array.
     Result:=VarArrayCreate([l,h],varByte);
     p:=VarArrayLock(Result);
     try
        DataStream.ReadBuffer(p^,h-l+1);
     finally
        VarArrayUnlock(Result);
     end;
end;
```

ReadStream returns a stream from the DataStream. Override it to read the stream in your own format.

```
procedure ReadStream(
   const AName:string;
   AStream:TStream;
   var ALength:integer); virtual;
```

```
procedure TYourTransportStream.ReadStream(
   const AName:string; AStream:TStream;
   var ALength:integer);
begin
     CheckStream;
     ALength:=ReadCount(AName+'_SIZE');
     AStream.CopyFrom(DataStream,ALength);
end;
```

Medium level data type methods. These methods are used for streaming more specialized types of information, like counts, ranges etc. Default they simply call lowlevel methods for the actual streaming. Often you will not need to override these methods.	
Description	Example

WriteCount store some count type of information in the stream. Override it to write store the numeric count value in the stream in your own format.
 	
procedure WriteCount(
const AName:string;
const ACount:integer); virtual;
	procedure TYourTransportStream.WriteCount(
	const AName:string;
	const ACount:integer);
	begin
	WriteInteger(AName,ACount);
	end;

WriteType store some kind of numeric type information in the stream. Override it to store the type information in the stream in your own format.
 	
procedure WriteType(
const AName:string;
const AType:integer); virtual;
	procedure TYourTransportStream.WriteType(
	const AName:string;
	const AType:integer);
	begin
	WriteInteger(AName,AType);
	end;

WriteRange store a numeric range in the stream (start/end values). Override it to store the numeric range start/end value in the stream in your own format.
 	
procedure WriteRange(
const AName:string;
const ALow,AHigh:integer); virtual;
	procedure TYourTransportStream.WriteRange(
	const AName:string;
	const ALow,AHigh:integer);
	begin
	WriteInteger(AName+'_LOW',ALow);
	WriteInteger(AName+'_HIGH',AHigh);
	end;

WriteIdentifier store a numeric identifier in the stream. Override it to store the numeric identifier in the stream in your own format.
 	
procedure WriteIdentifier(
const AName:string;
const AIdentifier:integer); virtual;
	procedure TYourTransportStream.WriteIdentifier(
	const AName:string;
	const AIdentifier:integer);
	begin
	WriteInteger(AName,AIdentifier);
	end;

WriteVersion store a numeric version value in the stream. Override it to store the numeric version value in the stream in your own format.
 	
procedure WriteVersion(
const AName:string;
const AVersion:integer); virtual;
	procedure TYourTransportStream.WriteVersion(
	const AName:string;
	const AVersion:integer);
	begin
	WriteInteger(AName,AVersion);
	end;

ReadCount reads a numeric count from the stream. Override it to read the numeric count from the stream in your own format.
 	
function ReadCount(
const AName:string):integer; virtual;
	function TYourTransportStream.ReadCount(
	const AName:string):integer;
	begin
	Result:=ReadInteger(AName);
	end;

ReadType reads a numeric type information from the stream. Override it to read the numeric type information from the stream in your own format.
 	
function ReadType(
const AName:string):integer; virtual;
	function TYourTransportStream.ReadType(
	const AName:string):integer;
	begin
	Result:=ReadInteger(AName);
	end;

ReadRange reads a numeric range (start/end) value from the stream. Override it to read the numeric range value from the stream in your own format.
 	
procedure ReadRange(
const AName:string;
var ALow,AHigh:integer); virtual;
	procedure TYourTransportStream.ReadRange(
	const AName:string;
	var ALow,AHigh:integer);
	begin
	ALow:=ReadInteger(AName+'_LOW');
	AHigh:=ReadInteger(AName+'_HIGH');
	end;

ReadIdentifier reads a numeric identifier value from the stream. Override it to read the numeric identifier value from the stream in your own format.
 	
function ReadIdentifier(
const AName:string):integer; virtual;
	function TYourTransportStream.ReadIdentifier(
	const AName:string):integer;
	begin
	Result:=ReadInteger(AName);
	end;

ReadVersion reads a numeric version value from the stream. Override it to read the numeric version value from the stream in your own format.
 	
function ReadVersion(
const AName:string):integer; virtual;
	function TYourTransportStream.ReadVersion(
	const AName:string):integer;
	begin
	Result:=ReadInteger(AName);
	end;

High level data type methods These methods are used for streaming high level information like headers. The methods use medium and lowlevel methods to do the actual streaming. You will usually not need to override these methods.	
Description	Example

These methods handles writing a header to the stream. The basic header contains information about type and version of the stream format.
In addition to that, the child classes - request and response - adds more information to the header. Generally these headers methods are not needed to be overriden unless you want to add some more information in the header in addition to whats there already.
 	
BeforeWriteHeader is guaranteed to be called before writing the header and AfterWriteHeader is guaranteed to be called immediately after the header has been written to the stream.
 	
`procedure BeforeWriteHeader; virtual;`
 	
`procedure WriteHeader(`
`  const AName:string); virtual;`
 	
`procedure AfterWriteHeader; virtual;`
	`procedure TYourTransportStream.WriteHeader(`
	`  const AName:string);`
	`begin`
	`  inherited;`
	`  WriteString('MYVALUE',FMyValue);`
	`end;`
	
	Will store one additional header value. Note that this require the client and the server to both accept and use the new transportstream format.

These methods handles reading a header from the stream. The basic ReadHeader method verifies that the header is valid and set the FVersion class variable to a combination of the streamformat type and version. The child request/response classes takes care of reading additional request and response related header information. BeforeReadHeader is guaranteed to be called before ReadHeader and AfterReadHeader is guaranteed to be called after ReadHeader.
 	
`procedure BeforeReadHeader; virtual;`
 	
`procedure ReadHeader(`
`  const AName:string); virtual;`
 	
`procedure AfterReadHeader; virtual;`
	`procedure TYourTransportStream.ReadHeader(`
	`  const AName:string);`
	`begin`
	`  inherited;`
	`  FMyValue:=ReadString('MYVALUE');`
	`end;`
	
	Will read one additional header value. Note that this require the client and the server to both accept and use the new transportstream format.

Flow of operation

The following pages contain the flow of the operations happening when making requests and receiving responses transport and transport stream wise.

Read the diagrams top, down one line at a time which is the chronological order in which the operations occur.

Flow for request from client to the transport media

Custom client	Request transport stream format	Custom client transport
SendRequest		
	BeforeWrite	
	BeforeWriteHeader	
	WriteHeader	
	AfterWriteHeader	
	WriteRequest	
	AfterWrite	
	BeforeEncrypt	
	Encrypt	
	AfterEncrypt	
	BeforeCompress	
	Compress	
	AfterCompress	
		Not Connected? Connect
		BeginTransmit
	BeforeSigning	
	Sign	
	AfterSigning	
	BeforeDeliver	
	Deliver	
		TransmitTransmissionHeader
		TransmitStream
	AfterDeliver	
		EndTransmit

Flow for request from transport media to server components and response back to transport media

Custom server request transport stream format	Custom server transport	Custom server response transport stream format
	BeginReceive	
BeforeRead		
		BeforeWrite
BeforeFetch		
Fetch		
	ReceiveTransmissionHeader	
	ReceiveStream	
AfterFetch		
BeforeVerify		
Verify		
AfterVerify		
BeforeDecompress		
Decompress		
AfterDecompress		
BeforeDecrypt		
Decrypt		
AfterDecrypt		
BeforeReadHeader		
ReadHeader		
AfterReadHeader		
ReadRequest		
Find requested service and process request.		
		BeforeWriteHeader
		WriteHeader
		AfterWriteHeader
		WriteResponse
AfterRead		
		AfterWrite
		BeforeEncrypt
		Encrypt
		AfterEncrypt
		BeforeCompress
		Compress
		AfterCompress
	BeginTransmit	
		BeforeSigning
		Sign
		AfterSigning
		BeforeDeliver
		Deliver
	TransmitTransmissionHeader	
	TransmitStream	
		AfterDeliver
	EndTransmit	
	EndReceive	

Flow for response from transport media to client

Custom client transport	Custom response transport stream format	Custom client
BeginReceive		
	BeforeFetch	
	Fetch	
ReceiveTransmissionHeader		
ReceiveStream		
	AfterFetch	
EndReceive		
	BeforeRead	
	BeforeVerify	
	Verify	
	AfterVerify	
	BeforeDecompress	
	Decompress	
	AfterDecompress	
	BeforeDecrypt	
	Decrypt	
	AfterDecrypt	
	BeforeReadHeader	
	ReadHeader	
	AfterReadHeader	
Check if response was a server exception. If so, throw exception and discard ReadResponse		
	ReadResponse	
	AfterRead	

Registration of transport stream formats

When a new complete transport stream format has been developed, it needs to be registered in the servers and the clients where its to be used,

The easiest way is to add a call to kbmMWRegisterTransportStream to the initializes section of the unit containing the transport stream code.

```
initialization

 kbmMWRegisterTransportStream(
           TkbmMW…RequestTransportStream,
           TkbmMW…ResponseTransportStream);
```

Kim Madsen.