

Object transfer using kbmMW

for kbmMW v. 1.00+

Datatypes

kbmMW contains direct support for moving datasets and normal variants from client to server and back. But its little known that kbmMW also handles moving objects.

Looking at a typical client request using a `TkbmMWSimpleClient` component we see that what is sent to the server must be of some variant type and what is returned also is of a variant type.

Since objects are not compatible with variants, then what to do?

There are several ways. If you are using Delphi 6 or newer, you can benefit of some advanced features which was introduced in kbmMW v. 1.00 gold 1 – object variants. If not, you can still get going although somewhat less elegant. Check the object conversion section.

Object variants

Object variants is a custom variant type implemented by kbmMW. This feature is only available starting with Delphi 6.

If you can design your objects from scratch, you should base the objects on TkbmMWObject. Creating your new objects you are required to overwrite two public methods: Reader and Writer. These two methods are used by kbmMW to stream the contents of an object.

An example on a simple object:

```

unit myclass;

interface

uses kbmMWCustomTransport, kbmMWGlobal;

type

  TMyClass = class(TkbmMWObject)
  private
    FValue1:integer;
    FValue2:string;
    FValue3:double;
  public
    procedure Reader(AObject: TObject; ATransportStream: TkbmMWCustomTransportStream); override;
    procedure Writer(AObject: TObject; ATransportStream: TkbmMWCustomTransportStream); override;
    property Value1:integer read FValue1 write FValue1;
    property Value2:string read FValue2 write FValue2;
    property Value3:double read FValue3 write FValue3;
  end;

implementation

procedure TMyClass.Reader(AObject: TObject; ATransportStream: TkbmMWCustomTransportStream);
begin
  with TMyClass(AObject) do
  begin
    FValue1:=ATransportStream.ReadInteger('VALUE1');
    FValue2:=ATransportStream.ReadString('VALUE2');
    FValue3:=ATransportStream.ReadFloat('VALUE3');
  end;
end;

procedure TMyClass.Writer(AObject: TObject; ATransportStream: TkbmMWCustomTransportStream);
begin
  with TMyClass(AObject) do
  begin
    ATransportStream.WriteInteger('VALUE1',FValue1);
    ATransportStream.WriteString('VALUE2',FValue2);
    ATransportStream.WriteFloat('VALUE3',FValue3);
  end;
end;

initialization
  kbmMWRegisterStreamableObject(TMyClass);

end.

```

What we have here is a simple object with 3 properties or values, Value1, Value2 and Value3. But in addition to that there are also two methods, Reader and Writer.

These two methods must be there in any class to be streamed automatically by kbmMW using the object variants syntax.

The methods should contain the code to respectively store and load whatever values from the object that you have the need for streaming. Storage and loading is done via methods of the `ATransportStream` object. That means that the data stored and loaded follows whatever transport stream format that have been selected on the transport component. Also notice the `kbmMWRegisterStreamableObject` call in the initialization part of the unit.

If you already have a class which you want to make streamable, there is a slightly different way to make it work.

```
TMyClass = class(TObject)
private
  FValue1:integer;
  FValue2:string;
  FValue3:double;
public
  property Value1:integer read FValue1 write FValue1;
  property Value2:string read FValue2 write FValue2;
  property Value3:double read FValue3 write FValue3;
end;
```

Then you need to create a subclass which implements the `IkbmMWObject` and the `IInterface` interfaces:

```
TStreamableMyClass = class(TMyClass, IInterface, IkbmMWObject)
protected
  function QueryInterface(const IID:TGUID; out Obj):HResult; stdcall;
  function _AddRef:integer; stdcall;
  function _Release:integer; stdcall;
public
  procedure Writer(AObject:TObject; ATransportStream:TkbmMWCustomTransportStream);
  procedure Reader(AObject:TObject; ATransportStream:TkbmMWCustomTransportStream);
end;
```

The `IInterface` part is implemented by adding the following code (must be like this every time):

```
function TStreamableMyClass.QueryInterface(const IID:TGUID; out Obj):HResult;
begin
  if GetInterface(IID,Obj) then
    Result:=0
  else
    Result:=E_NOINTERFACE;
end;

function TStreamableMyClass._AddRef:integer;
begin
  Result:=-1;
end;

function TStreamableMyClass._Release:integer;
begin
  Result:=-1;
end;
```

And the IkbmMWObject part is implemented just like in the previous example except you access the public properties instead of the private variables (if in same unit you can still use the private variables):

```

procedure TStreamableMyClass.Reader(AObject: TObject;
    ATransportStream: TkbmMWCustomTransportStream);
begin
    with TMyClass(AObject) do
    begin
        Value1:=ATransportStream.ReadInteger('VALUE1');
        Value2:=ATransportStream.ReadString('VALUE2');
        Value3:=ATransportStream.ReadFloat('VALUE3');
    end;
end;

procedure TStreamableMyClass.Writer(AObject: TObject;
    ATransportStream: TkbmMWCustomTransportStream);
begin
    with TMyClass(AObject) do
    begin
        ATransportStream.WriteInteger('VALUE1',Value1);
        ATransportStream.WriteString('VALUE2',Value2);
        ATransportStream.WriteFloat('VALUE3',Value3);
    end;
end;

```

The how to use it?

Well firstly make sure that the object is registered in using the kbmMWRegisterStreamableObject global method (use the one with only one argument for this type of object).

procedure kbmMWRegisterStreamableObject(AObject:TObject)

Then you'll need to get your object wrapped into a special type of variant a object variant. This is done using the global function kbmMWVarObjectCreate.

function kbmMWVarObjectCreate(AObject:TObject):variant

Notice that it does not make a copy of your object, it use it directly as is. Therefore do not free your object until after kbmMW has streamed it, but do remember to free it then. If not you will have a severe memory leak. On the server it is often not possible to free the server side instance of an object after it has been streamed simply because the object is the result of some server side function. In that case you can instead use the global function kbmMWVarObjectCreateAutoDispose. Notice that assigning null to an object variant do not free the object it holds.

function kbmMWVarObjectCreateAutoDispose(AObject:TObject):variant

If a object is put into an object variant, it will automatically be freed by kbmMW when it has been streamed.

To get the object reference back from an object variant, use the global function kbmMWObjectFromVar.

function kbmMWObjectFromVar(AVariant:variant):TObject

To make a call using a TkbmMWSimpleClient you could do something like this:

```

procedure TForm1.Button1Click(Sender: TObject);
var
  o,o2:TMyClass; // or TStreamableMyClass depending on which way you created your object.
  v:variant;
  i:integer;
begin
  // Create an instance of the object to be streamed. Initialize it with some values.
  o:=TMyClass.Create; // or TStreamableMyClass.Create...
  try
    o.Value1:=10;
    o.Value2:='TEST 10';
    o.Value3:=10.00;

    // Call a server function (PUTOBJECT) in the service
    // named OBJECTSERVICE (some custom service created by you).
    // Notice the special kbmMWVarObjectCreate call.
    v:=kbmMWSimpleClient1.Request('OBJECTSERVICE','', 'PUTOBJECT', [kbmMWVarObjectCreate(o)]);

    // The server returned in our example another object variant.
    // A new object instance was automatically created for us and stored in the object
    // variant. To get to the object use the following call:
    o2:=TMyClass(kbmMWObjectFromVar(v)); // or cast to TStreamableMyClass...

  try
    // Show contents of returned object.
    SomeLabel.Caption:='Value1='+inttostr(o2.Value1)+' Value2='+o2.Value2+
      ' Value3='+floattostr(o2.Value3);
  finally
    // Its our responsibility to free the returned object after use.
    o2.Free;
  end;
  finally
    // Now we free our original object.
    o.Free;
  end;
end;
end;

```

On the server side you could have a custom service with a function looking like this:

```
function TkbmMWCustomService1.PerformPUTOBJECT(ClientIdent:TkbmMWClientIdentity;
const Args:array of Variant):Variant;
var
  o:TMyClass;
begin
  // Fetch the object. Notice that this server side object was automatically
  // created during loading the client request by kbmMW.
  // Thus its your responsibility to destroy it.
  o:=TMyClass(kbmMWObjectFromVar(Args[0]));

  // Do some calculations on it.
  o.Value1:=o.Value1+1;
  o.Value2:=o.Value2+' TESTED';
  o.Value3:=o.Value3*2;

  // Return the object as result to the client. Let kbmMW destroy the object
  // instance after it has been streamed.
  Result:=kbmMWVarObjectCreateAutoDispose(o);
end;
```

The great thing about object variants is that you are 100% in charge of whats streamed and whats not. You can choose to stream nested objects if you wish in the Reader and Writer methods. Further the objects are streamed using whatever transport stream format that has been selected on the transport. Thus your object could automatically be shown as XML or EDI or whatever if a stream format of that type is available. The transport stream also contains more functions for easy streaming of lots of different data types. Finally the object is more memory conserving than with object conversion which is shown in the next part of this whitepaper.

Remember that since the variant only contains a reference to your object, and not the object itself, you have to take special care if you assign one variant to another not use any of them if the object has been destroyed. Destruction of the object do not automatically empty your variants.

Thus a good rule of thumb is to assign null to your variants or otherwise empty them just before destroying the object.

Object conversion

For quite some time, kbmMW have had some pretty smart global functions which could convert an object to a variant and back.

```
function kbmMWObject2Variant(AObject:TObject):variant  
function kbmMWVariant2Object(AVariant:variant):TObject
```

Objects are converted to a specially formatted variant array which - in case of an object based on TComponent - contains the value of all published properties.

The stored data is in the same format as how Delphi store its form data. It's Delphi that are in control of doing the actual streaming.

E.g.

```
TMyClass = class(TComponent)
private
    FValue1:integer;
    FValue2:string;
    FValue3:double;
published
    property Value1:integer read FValue1 write FValue1;
    property Value2:string read FValue2 write FValue2;
    property Value3:double read FValue3 write FValue3;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    v:variant;
    o,o2:TMyClass;
begin
    o:=TMyClass.Create(nil);
    try
        o.Value1:=10;
        o.Value2:='TEST';
        o.Value3:=10.00;
        v:=kbmMWObject2Variant(o);
        v:=kbmMWSimpleClient1.Request('OBJECTSERVICE2','','PUTOBJECT',[v]);
        o2:=kbmMWVariant2Object(v);
        try
            // Show contents of returned object.
            SomeLabel.Caption:='Value1='+inttostr(o2.Value1)+' Value2='+o2.Value2+
                ' Value3='+floattostr(o2.Value3);
        finally
            o2.Free;
        end;
    finally
        o.Free;
    end;
end;
```

If the object is not based on TComponent, you have to register the object and two methods which takes care of streaming the data.

```
procedure kbmMWRegisterStreamableObject(AObjectClass:TClass;
AObjectReader:TkbmMWObjectReader;
AObjectWriter:TkbmMWObjectWriter);
```

```
TkbmMWObjectWriter = procedure (AObject:TObject; AStream:TStream);
TkbmMWObjectReader = function (AStream:TStream):TObject;
```

E.g.

```
TMyClass = class(TObject)
private
  FValue1:integer;
  FValue2:string;
  FValue3:double;
public
  property Value1:integer read FValue1 write FValue1;
  property Value2:string read FValue2 write FValue2;
  property Value3:double read FValue3 write FValue3;
end;

function MyClassObjectReader (AStream:TStream):TObject;
var
  i:integer;
  f:double;
  s:string;
  sz:integer;
  o:TMyClass;
begin
  // Load data from stream.
  AStream.Read(i, sizeof(Integer));
  AStream.Read(f, sizeof(Double));
  AStream.Read(sz, sizeof(Integer));
  SetLength(s, sz);
  AStream.Read(s, sz);

  // Create object based on loaded data.
  o:=TMyClass.Create;
  o.Value1:=i;
  o.Value2:=s;
  o.Value3:=f;
  Result:=o;
end;
```

```

procedure MyClassObjectWriter(AObject:TObject; AStream:TStream);
var
  i:integer;
  f:double;
  s:string;
  sz:integer;
  o:TMyClass;
begin
  o:=TMyClass(AObject);
  i:=o.Value1;
  s:=o.Value2;
  f:=o.Value3;
  sz:=length(s);
  AStream.Write(i, sizeof(Integer));
  AStream.Write(f, sizeof(Double));
  AStream.Write(sz, sizeof(Integer));
  AStream.Write(s, sz);
end;

initialization
kbmMWRegisterStreamableObject(TMyClass, @MyClassObjectReader, @MyClassObjectWriter);

```

The object conversion syntax is the same as in previous example basing the object on TComponent.

```

procedure TForm1.Button1Click(Sender: TObject);
var
  v:variant;
  o, o2:TMyClass;
begin
  o:=TMyClass.Create(nil);
  try
    o.Value1:=10;
    o.Value2:='TEST';
    o.Value3:=10.00;
    v:=kbmMWObject2Variant(o);
    v:=kbmMWSimpleClient1.Request('OBJECTSERVICE2', '', 'PUTOBJECT', [v]);
    o2:=kbmMWVariant2Object(v);
    try
      // Show contents of returned object.
      SomeLabel.Caption:='Value1='+inttostr(o2.Value1)+' Value2='+o2.Value2+
        ' Value3='+floattostr(o2.Value3);
    finally
      o2.Free;
    end;
  finally
    o.Free;
  end;
end;

```

Similar kbmMWObject2Variant and kbmMWVariant2Object conversions are needed on the server side too, to get hold of the object sent from the client and to return an object back to the client. E.g.

```
function TkbmMWCustomService1.PerformPUTOBJECT(ClientIdent:TkbmMWClientIdentity;
        const Args:array of Variant):Variant;
var
    o:TMyClass;
begin
    // Fetch the object.
    // Notice that the variant conversion automatically creates an object which you must destroy
    // when its not needed anymore.
    o:=TMyClass(kbmMWVariant2Object(Args[0]));
    try
        // Do some calculations on it.
        o.Value1:=o.Value1+1;
        o.Value2:=o.Value2+' TESTED';
        o.Value3:=o.Value3*2;

        // Convert the object to a variant.
        Result:=kbmMWObject2Variant(o);
    finally
        o.Free;
    end;
end;
```

Then which method to use - Object variants or Object conversion?

Its recommended that whenever possible to use Object variants instead of Object conversion since it gives a more clean implementation, saves memory and allows for the object being transferred to follow the transport stream format.

Never mix the two methods. If one end use one method to send an object, the other end must use the same method to receive it.

This concludes the object transfer whitepaper.

Kim Madsen
Components4Developers