Unique numbers/Sequencers

for kbmMW v. 2.00+

kbmMW v2 contains new functionality for manage unique numbers.
Whats the point of that? Well unique numbers are one of the most used things in database applications. You need unique numbers for identifying for example a specific person record, an order item record etc.

Traditionally different forms of auto increment fields was used in which the database automatically assigned a number each time a record was inserted. The problem with that scheme is that its not always possible to know exactly what number was assigned. Different databases/datastores handle that situation in different ways, and some don't handle it at all.

Why would you like to know the number after inserting the record? Well for one your client may be interested in knowing the number to be able to reference to the record later on or to handle master/detail setups reliably and easily.

A sequence is in kbmMW identified with a name and optionally an initial value (start value for counting).

Each time one draws a number from a named sequence, the sequence will automatically be updated with the next number in line.

kbmMW bundles two types of sequence components, **TkbmMWTemporarySequencer** and **TkbmMWDatastoreSequencer**.

The temporary sequencer store the sequence values in memory, and thus the values are not easily shareable between multiple instances of the application server. The values in the temporary sequencer can be stored to a file or stream and reloaded later on.

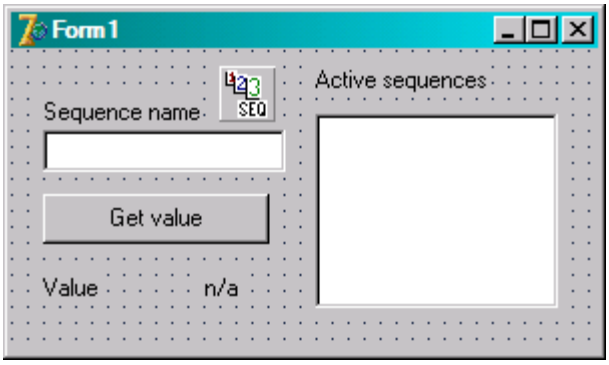
The datastore sequencer store its sequence values in a datastore/database instead. How the values/sequences are stored depends on the capabilities of the datastore/database and on which metadata component have been chosen to represent the database (read more about metadata components in the 'Metadata with kbmMW' whitepaper.

For example on Oracle databases, using the Oracle metadata component, all kbmMW sequences accessed through the same connection pool, will automatically be represented in the database as Oracle SEQUENCE objects.

On Interbase/Firebird, using the Interbase metadata component, all kbmMW sequences accessed through the same connection pool, will automatically be represented in the database as Interbase/Firebird GENERATOR objects.

Using TkbmMWTemporarySequencer

Either put a TkbmMWTemporarySequencer component on the form/datamodule, or create one at runtime.



In the OnClick event of the 'Get value' button we can add the following sample code:

```
….
begin
     Label1.Caption:=inttostr(kbmMWTemporarySequencer1.Value[Edit1.Text]);
     ListBox1.Items.Assign(kbmMWTemporarySequencer1.Values);
end;
```

What happens is that next numeric value is fetched from the temporary sequencer for the sequence which name is given in the edit box. This value is displayed in a label.
Then the list of known sequence names is displayed in the listbox.

What we see here is that sequences are automatically created starting from 0 if they don't exist when a value is fetched from them. This can be prevented by setting the property **AutoCreateSequence** to false (default true).

Its possible to specifically create, reset and delete specific named sequences using the functions **CreateSequence**, **ResetSequence** and **DeleteSequence**.

CreateSequence require a sequence name and an initial value (starting point for the counter).
ResetSequence alters the next value for an existing sequence to a new value.
DeleteSequence obviously delete a named sequence.

Example: CreateSequence

```
var
   n:integer;
begin
  kbmMWTemporarySequencer1.CreateSequence('SEQ1',20000);
  n:=kbmMWTemporarySequencer1.Value['SEQ1'];
end;
```

n will have the value of 20000 or 20001 (depending on the way the sequencer schedules numbers) first time.

Example: ResetSequence

```
var
   n:integer;
begin
  kbmMWTemporarySequencer1.ResetSequence('SEQ1',20000);
  n:=kbmMWTemporarySequencer1.Value['SEQ1'];
end;
```

Regardless of what the original value of the named sequence SEQ1 had before, n will now be 20000 or 20001 on each call.

Example: DeleteSequence

```
var
   n:integer;
begin
  kbmMWTemporarySequencer1.DeleteSequence('SEQ1');
end;
```

Finally its possible to save the contents of all the sequences in a file or stream or another string list via the Values property.

Example: Saving and loading the sequence values to/from a file.

```
begin
  kbmMWTemporarySequencer1.Values.SaveToFile('SomeFile.txt');
  ...
  kbmMWTemporarySequencer1.Values.LoadFromFile('SomeFile.txt');
end;
```

Using TkbmMWDatastoreSequencer

Its used exactly the same way as the **TkbmMWTemporarySequencer** except that the sequence names/values cannot be stored to or loaded from a file or stream, but instead are stored in a datastore/database.

The **TkbmMWDatastoreSequencer** have an additional property **ConnectionPool** which must be set to point on a connection pool which represents the database/datastore that will host the sequences from this sequencer.

Also its important to set the **MetaData** property of the connection pool to the type of meta data component that match the database/datastore, or match the way that sequences should be stored. The **TkbmMWGenericSQLMetaData** component will always create a pivot table which will contain the sequence names and values, while the **TkbmMWOracleSQLMetaData** component instead will make kbmMW use Oracle's SEQUENCE objects etc.

If you choose to use the TkbmMWGenericSQLMetaData component its important once to call the MetaInitialize function specifying the mwmdtSequence type as the argument. Eg.

```
connectionpool.MetaInitialize(mwmdtSequence);
```

This ensures that a pivot table is automatically created. The name of the pivot table is default 'KBMMW_SEQUENCES'. The name can be altered in the property **SequenceTableName** of the **TkbmMWGenericSQLMetaData** component before initializing the sequences.

Again its possible to set **AutoCreateSequence** to false to fail when a sequence hasnt been explicitly created instead of silently create one.

CreateSequence, **ResetSequence** and **DeleteSequence** operates the same as with the **TkbmMWTemporarySequencer** except on the datastore/database.

Please notice that some datastores (like Oracle) dont support resetting a sequence to a new value. Instead kbmMW will transparently automatically delete and recreate the sequence with the new starting offset.

Creating your own number generator

If you want to create your own number generator, you need to override a couple of methods:

```
interface

TMySequencer = class(TkbmMWCustomSequencer)
private
    FValueList:TStringList;
    FIncrements:integer;
protected
    function GetValue(AIdent:string):longint; override;
public
    constructor Create(AOwner:TComponent); override;
    destructor Destroy; override;

    function CreateSequence(AIdent:string; AValue:longint):boolean; override;
    function ResetSequence(AIdent:string; AValue:longint):boolean; override;
    function DeleteSequence(AIdent:string):boolean; override;
published
    property Increments:integer read FIncrements write FIncrements;
end;

implementation

constructor TMySequencer.Create(AOwner:TComponent);
begin
   inherited;
   FValueList:=TStringList.Create;
   FIncrements:=10;
end;

destructor TMySequencer.Destroy;
begin
   FValueList.Free;
   inherited;
end;

function TMySequencer.GetValue(AIdent:string):longint;
var
   i,v:integer;
begin
    i:=FValueList.IndexOfName(AIdent);
    if i<0 then kbmMWRaiseException(AIdent+' not found.');
    Result:=integer(FValueList.Objects[i];
    integer(FValueList.Objects[i]):=Result+FIncrement;
end;


function TMySequencer.CreateSequence(AIdent:string; AValue:longint):boolean;
begin
    if FValueList.IndexOfName(AIdent)>=0 then
       kbmMWRaiseException(AIdent+' already exists.');
    FValueList.AddObject(AIdent,pointer(AValue));
end;
```

```
function TMySequencer.ResetSequence(AIdent:string; AValue:longint):boolean;
var
   i:integer;
begin
    i:=FValueList.IndexOfName(AIdent);
    if i<0 then
       kbmMWRaiseException(AIdent+' not found.');
    FValueList.Objects[i]:=pointer(AValue);
end;

function TMySequencer.DeleteSequence(AIdent:string):boolean;
var
   i:integer;
begin
    i:=FValueList.IndexOfName(AIdent);
    if i<0 then
       kbmMWRaiseException(AIdent+' not found.');
    FValueList.Delete(i);
end;
```

This concludes the whitepaper about the unique numbers and sequences.

Kim Madsen
Components4Developers