

Using kbmMW as a file server

for kbmMW v. 1.00rc3b+

kbmMW includes a custom service, `TkbmMWFileService`, which combined with a component `TkbmMWFilePool` makes kbmMW able to efficiently operate as a file server for kbmMW based clients.

Even without the `TkbmMWFileService`, it has been possible to copy files from a client to the server and back via the `RequestStream` or `ResultStream` memory streams, but the problem with this approach is that files may be pretty large and since the complete file has to reside in memory to be in a `RequestStream` or `ResultStream`, and because compression, encryption etc. require additional copies in memory, the memory usage can really fast become much to big.

A file of 30MB would easily take up 90MB of memory while being transferred, both on client and server side. This would fast make the server and possibly the client really slow and might end up crashing the application due to the machine its running on, running out of virtual memory.

Thus what is needed is a way to copy the files in sizable chunks.

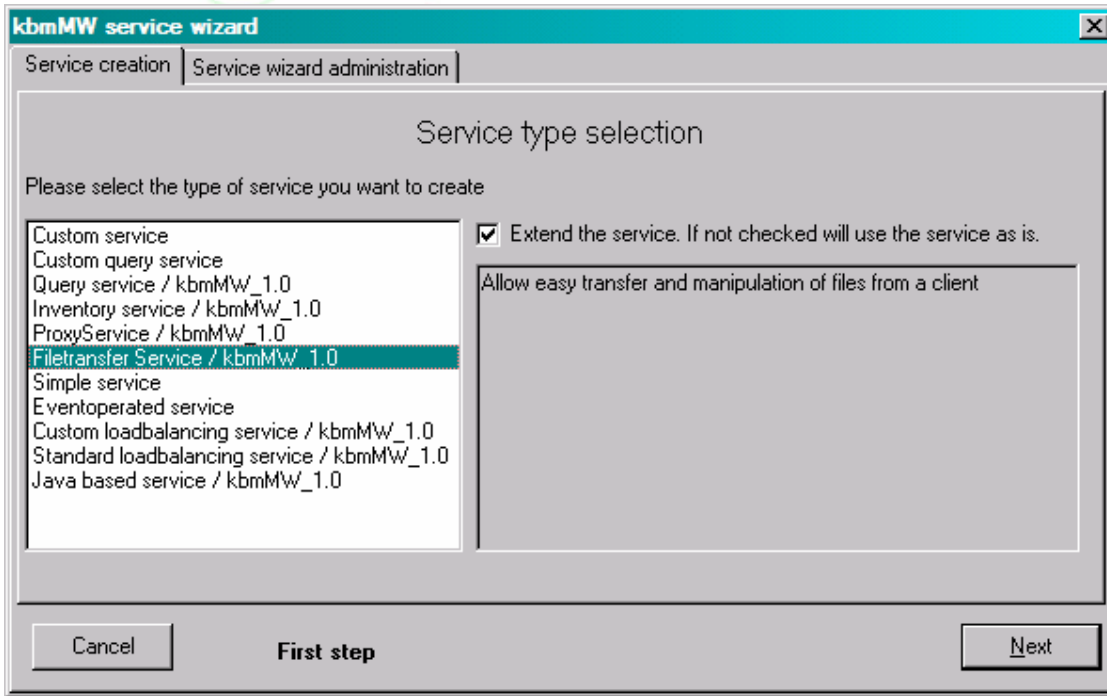
This is the job of the file service and the file client component.

The problem about copying in chunks is that some kind of state information is needed... how much have been copied already, and what is missing. Further the server would have to repeatedly access the same file each time the client want to get another chunk. This would require the server to open, send chunk and close the file each time the client makes a request.

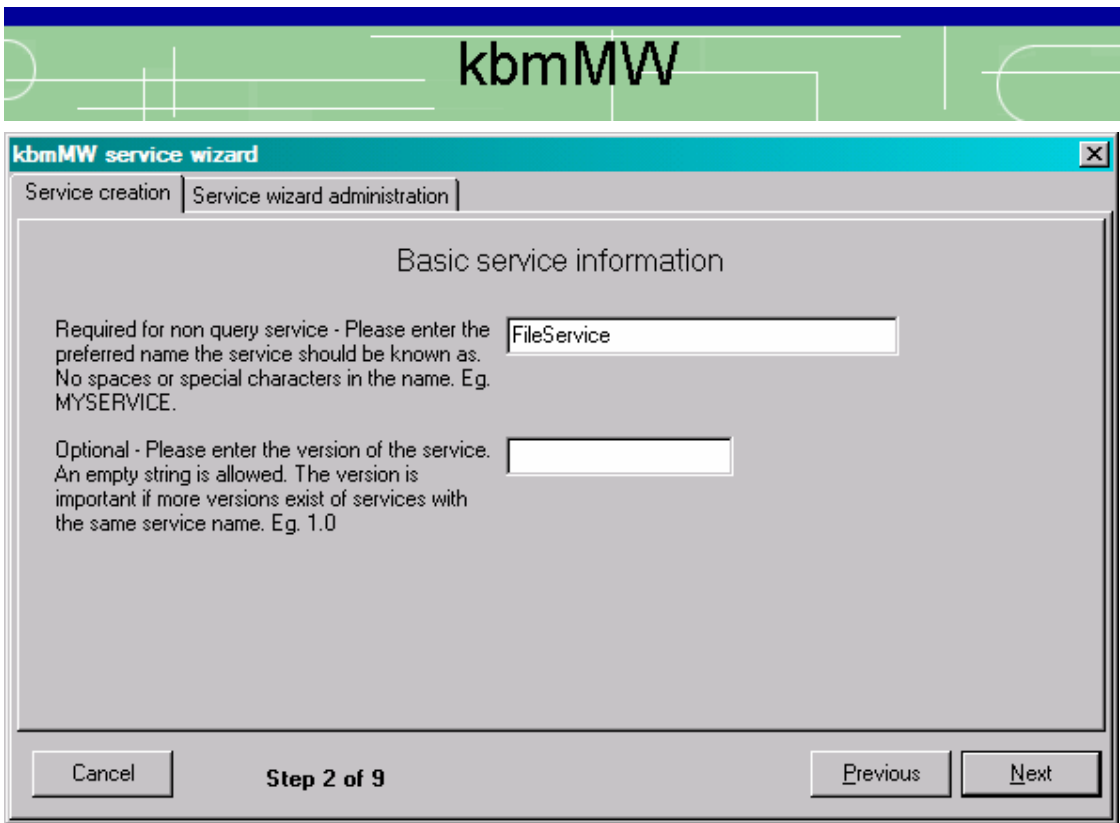
This is not very efficient, and therefore kbmMW includes the `TkbmMWFilePool` which takes care of caching file handles and makes sure not to allow one client to write to a file being read by another etc.

Creating the File service

First create new File service using the kbmMW service wizard:

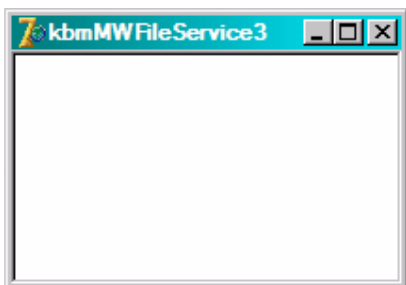


As with any other custom service, give it a preferred name and fill in any other optional service informations.

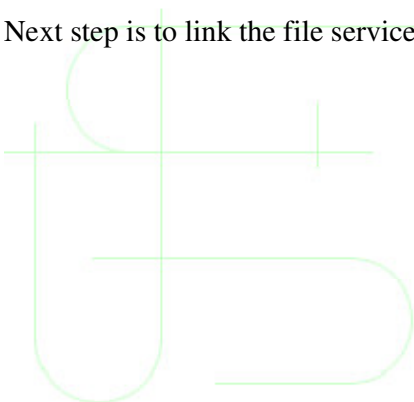


You don't need to add any service functions to the service, since that functionality is already in place in the `TkbmMWFileService` base class. You can however as usual choose to add your own extra native functionality in the same service if you want to by overriding `ProcessRequest` and remember to call inherited for functions not covered by your custom native code.

Finish the service creation and you will end up with an empty service datamodule as usual.

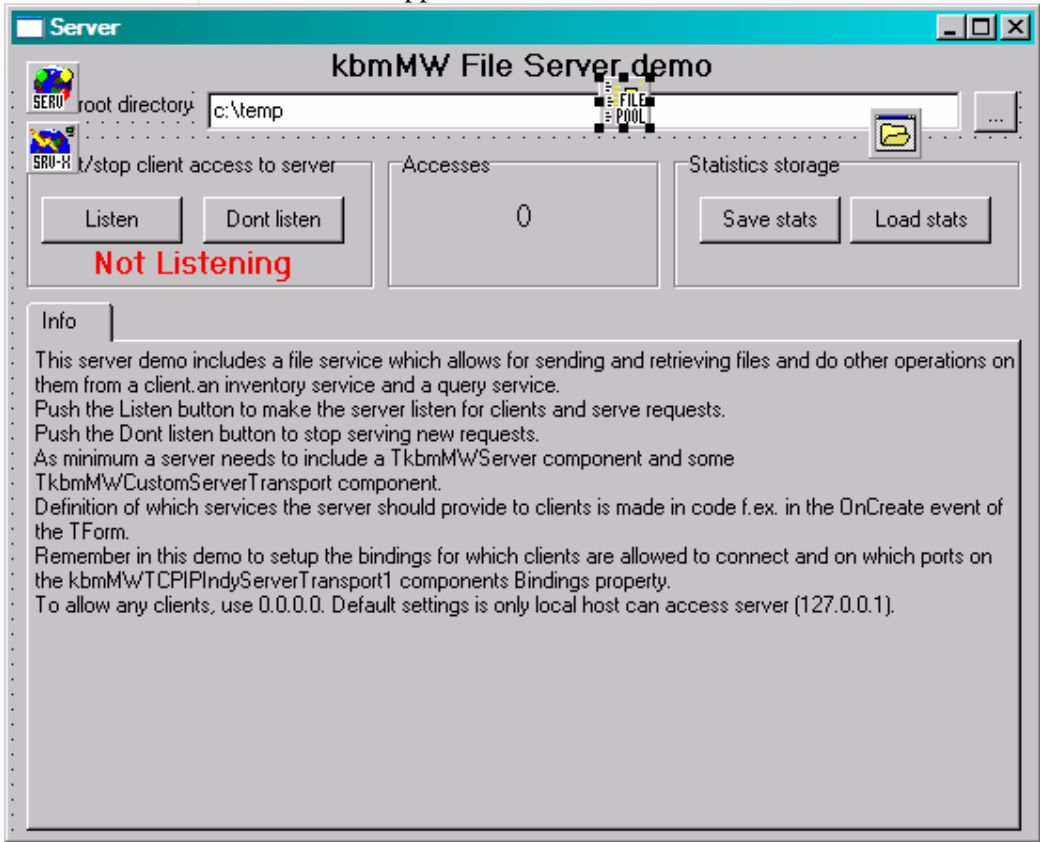


Next step is to link the file service to a file pool.





Put a `TkbmMWFilePool` on the application servers main form.



Optionally set its properties:

AutoCreateDirectories: Set to true if the client should be allowed to upload files which contain directory information even if the directories do not exist already on the server.

GarbageCollection: Usually leave as true to allow the server to clean up in its file handle pool.

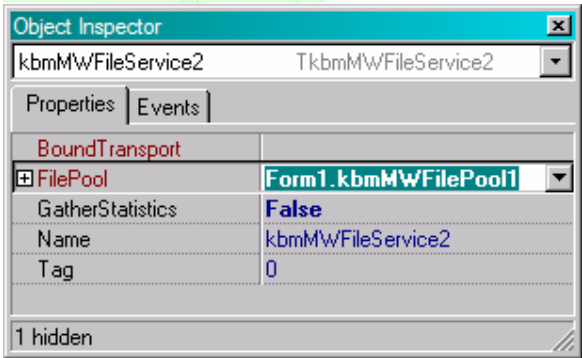
GarbageInterval: The number of seconds between each garbage collection sweep. Usually leave on default.

MaxAge: The number of seconds a file handle is allowed to be idle (not used by any client requests) before its marked as ready for being garbagecollected. Usually leave on default.

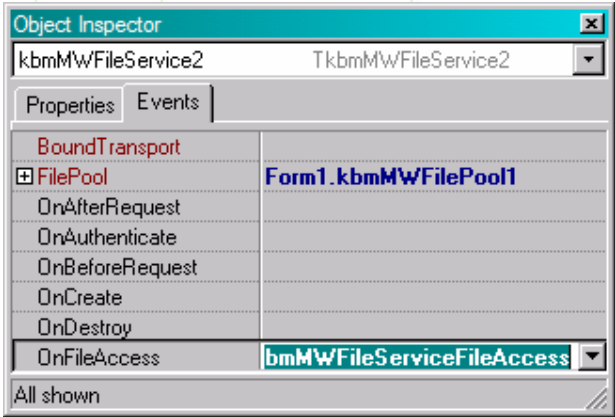
Optionally add an event handler for its **OnGarbageCollect** event which is called on each garbage collection sweep.



Then on the file service datamodule make sure to 'Use' the server form (File/Use) and set the FilePool property of the file service datamodule to the newly added TkbmMWFilePool.



Optionally put some code in the OnFileAccess event of the file service datamodule.



```

procedure TkbmMWFileService4.kbmMWFileServiceFileAccess(Sender: TObject;
  Path: String; var AccessPermissions: TkbmMWFileAccessPermissions);
begin
  // Optionally put code in this event to determine what permissions a client should have
  // on a specific file.
  // Default permissions is given in AccessPermissions. Override those with the ones you need
  // for the file
  // specified with Path.
end;

```

AccessPermissions is a set of TkbmMWFileAccessPermission which can be:

mwfapList: If set on entry means that the client wants to get a list of file names from the server on the Path specified. By conditionally clearing this value, you can deny the client access to get a list of file names.

mwfapGet: If set on entry means that the client wants to receive a chunk of the file specified by Path. By conditionally clearing this value, you can deny the client to receive a chunk of the file (or in reality the complete file).

mwfapPut: If set on entry means that the client wants to deliver a chunk of a file to the place specified by *Path*. By conditionally clearing this value, you can deny the client to deliver the file chunk (or in reality the complete file).

mwfapDelete: If set on entry means that the client wants to delete the file specified with *Path*. By conditionally clearing this value, you can deny the client the right to delete the file.

mwfapOverwrite: Used in combination with **mwfapPut**. If this value is not set (default), overwriting existing files are not allowed.

mwfapSubDirs: Used in combination with **mwfapList**. If this value is not set (default), the client will not be allowed to get a list of file names from subdirectories.

mwfapStatus: If set on entry means that the client wants to obtain information about the specific file given by *Path* (size, attributes etc.). If you conditionally clear this value, the client will not be allowed to obtain the information.

Registering the File service

Just like with any other kbnMW services, the File service needs to be registered to the TkbmMWServer. At the same time several File service specific options can (and should) be set:

```
var
  sd:TkbmMWCustomServiceDefinition;
begin
  sd:=kbnMWServer1.RegisterService(TFileService4, false);
  with sd as TkbmMWFileServiceDefinition do
  begin
    RootPath:='c:\temp'; // Default value is .
    ListAttributes:= faReadOnly+faDirectory+faArchive; // Default values.
    BlockSize:=2048; // Default value 8192.
  end;
end;
```

Here you can see that the custom service definition returned by RegisterService can be casted to a File specific version of it. This gives access to several File specific settings that usually should be set.

RootPath should point to the directory which will be seen as the ‘virtual root’ directory from which the client will be able to obtain files from and deliver files to. All client file operations are relative to this path.

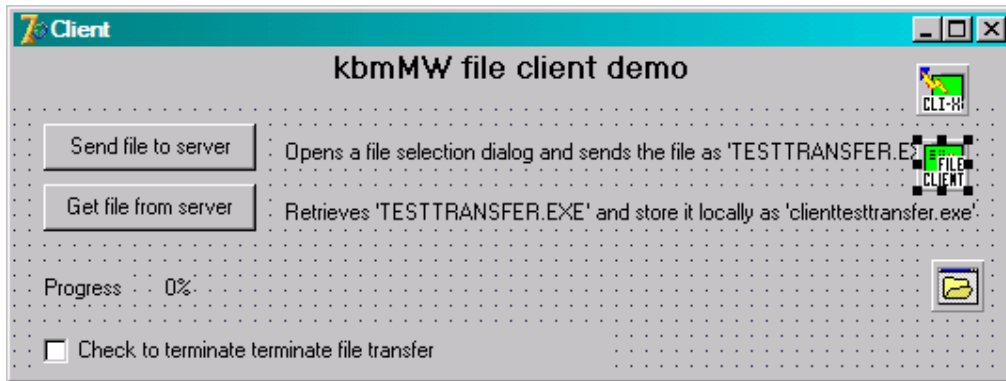
ListAttributes should be set to the attributes of files the client is allowed to list. Only files matching these attributes are listed on a client list request.

BlockSize is the size of the chunks delivered by the server to the client. The client can choose to send with a different block size when uploading files. Default its 8192 bytes, but a better value is often 2048.

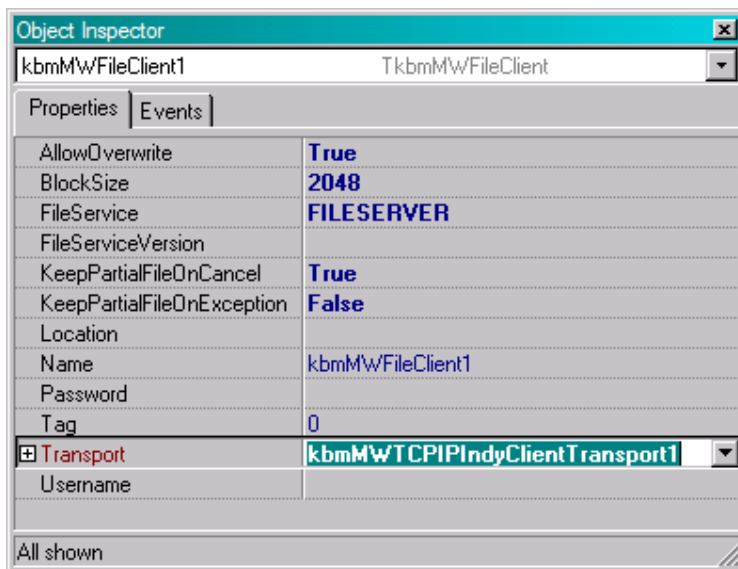
Multiple file services can be registered in the same application server. They must however each have a unique ServiceName/ServiceVersion identifier.

Client side

On the client, a special client side file component is used to facilitate the communication with the file service.



As usual a client side transport is available along with the `TkbmMWFileClient` component. Hook the `Transport` property of the file client component up to the client transport.



Next set the **FileService** and **FileServiceVersion** properties to the selected name and version string of the file service on the server side.

Optionally fill in **Username**, **Password** and **Location** which functions like on the `TkbmMWSimpleClient` component.

Set **BlockSize** to a reasonable value (default 8192 bytes, but 2048 is found to be a good size). It controls the size of file chunks delivered by the client to the server.

KeepPartialFileOnCancel can be set to true if a partial download of a file should be saved when the user decides to abort the transfer. (see the description of the OnProgress event). Default partial transfers are deleted.

KeepPartialFileOnException can be set to true if a partial download of a file should be saved when an exception occur during the transfer. Default partial transfers are deleted.

The **OnProgress** event can be used for two purposes: to keep track of progress, and to allow the user to cancel a file transfer.

```
procedure TForm1.kbmMWFileClient1Progress(Sender: TObject; Pct: Integer;  
    Receiving: Boolean; var Terminate: Boolean);  
begin  
    lPct.Caption:=inttostr(Pct)+'%';  
    Terminate:=chbTerminate.Checked;  
end;
```

Pct contains a number from 0 to 100 specifying the percentage of completion of the transfer.

Receiving is true if the transfer is inbound (from server to client).

Terminate can optionally be set to true to abort the transfer.

Sending a file to the server

```
kbmMWFileClient1.PutFile(OpenDialog1.FileName, 'TESTTRANSFER.EXE');
```

The 1st argument specifies the name of the file to send on the client. The 2nd the name of the file on the server (optionally including a path). The file will be placed relative to the RootPath that has been set on the server.

Receiving a file from the server

```
kbmMWFileClient1.GetFile('clienttesttransfer.exe','TESTTRANSFER.EXE');
```

The 1st argument specifies the name to store the file as on the client, and the 2nd the name of the file on the server.

Getting a list of file names on the server

```
kbmMWFileClient1.ListFiles('.',true);
```

1st argument is the path relative to the RootPath on the server to obtain the file name list from. The 2nd argument can be either true or false depending on if the client would like to recurse subdirectories (the server default forbid that).

Deleting files on the server

```
kbmMWFileClient1.DeleteFile('TESTTRANSFER.EXE');
```

The argument is the name of the file to delete on the server.

Obtaining status information of a file on the server

This needs a special call to the server:

```
var
  v:variant;
begin
  v:=kbmMWFileClient1.Request(kbmMWFileClient1.FileService,
                              kbmMWFileClient1.FileServiceVersion,
                              'STATUS', ['TESTTRANSFER.EXE']);

  ShowMessage('Filesize='+inttostr(v[0]));
  ShowMessage('FileCreated='+DateTimeToString(v[1]));
  ShowMessage('FileAccessed='+DateTimeToString(v[2]));
  ShowMessage('FileModified='+DateTimeToString(v[3]));
end;
```

Advanced use

In addition to these methods, there are existing similar ones with the tag Ex put on the name. Eg. DeleteFileEx etc.

This is an extended method which allow the developer to send additional arguments to the file service. They only have a use if you have overridden the *ProcessRequest* method on the service and need additional information sent from the client.

For List requests, the user defined arguments start as Args[2] (3rd argument)

For Status requests, the user defined arguments start as Args[2] (3rd argument)

For Get requests, the user defined arguments start as Args[3] (4th argument)

For Put requests, the user defined arguments start as Args[3] (4th argument)

For Delete request, the user defined arguments start as Args[2] (2nd argument)

That concludes the whitepaper about using kbmMW as a file server.

Kim Madsen
Components4Developers