The Proxyservice

for kbmMW v. 1.03+

One of the bundled services, the proxy service, is very useful for proxying requests from clients to one or more other application servers who does the real work.

An example for a very useful usage is in ISAPI based kbmMW application servers that are hosted by an ISAPI capable web server like IIS. The proxy service will allow controlled access from the outside world, via the web server to internal application servers through firewalls etc. without needing to reconfigure network hardware.

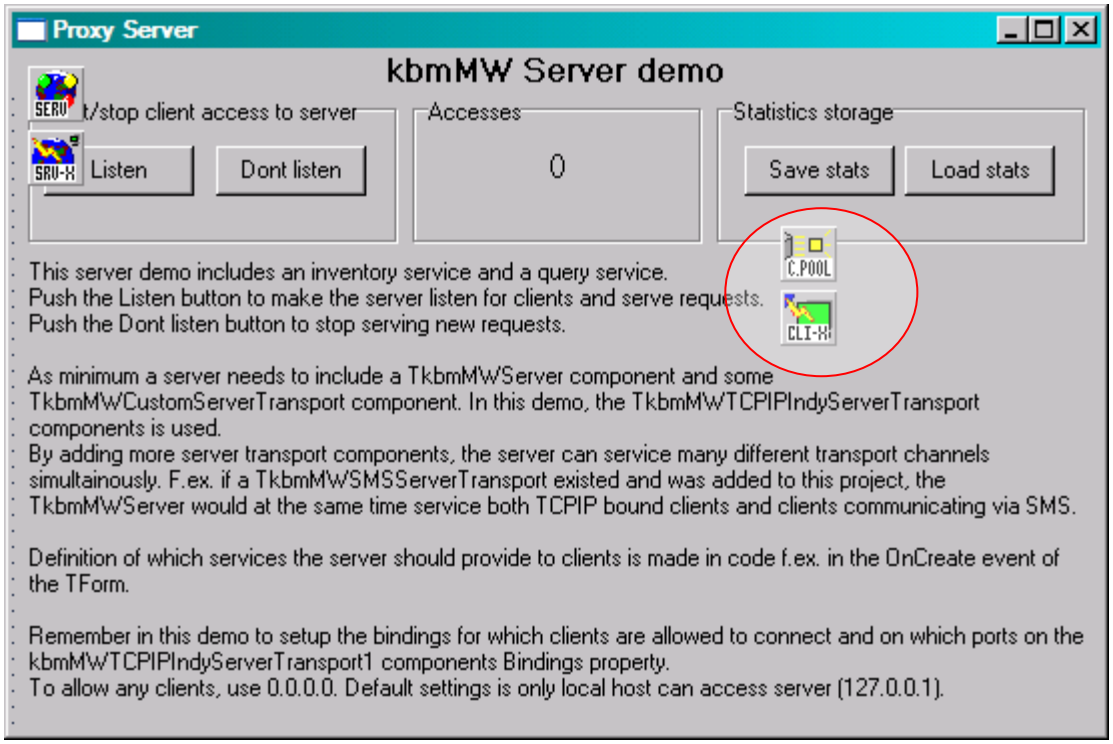
This document will explain how to use the proxy service.

The prerequisite for following the documentation is to have an empty application server main form setup already. It can be ISAPI based or standalone, you decide, but it must have the TkbmMWServer component on it.

Creating the Proxy service

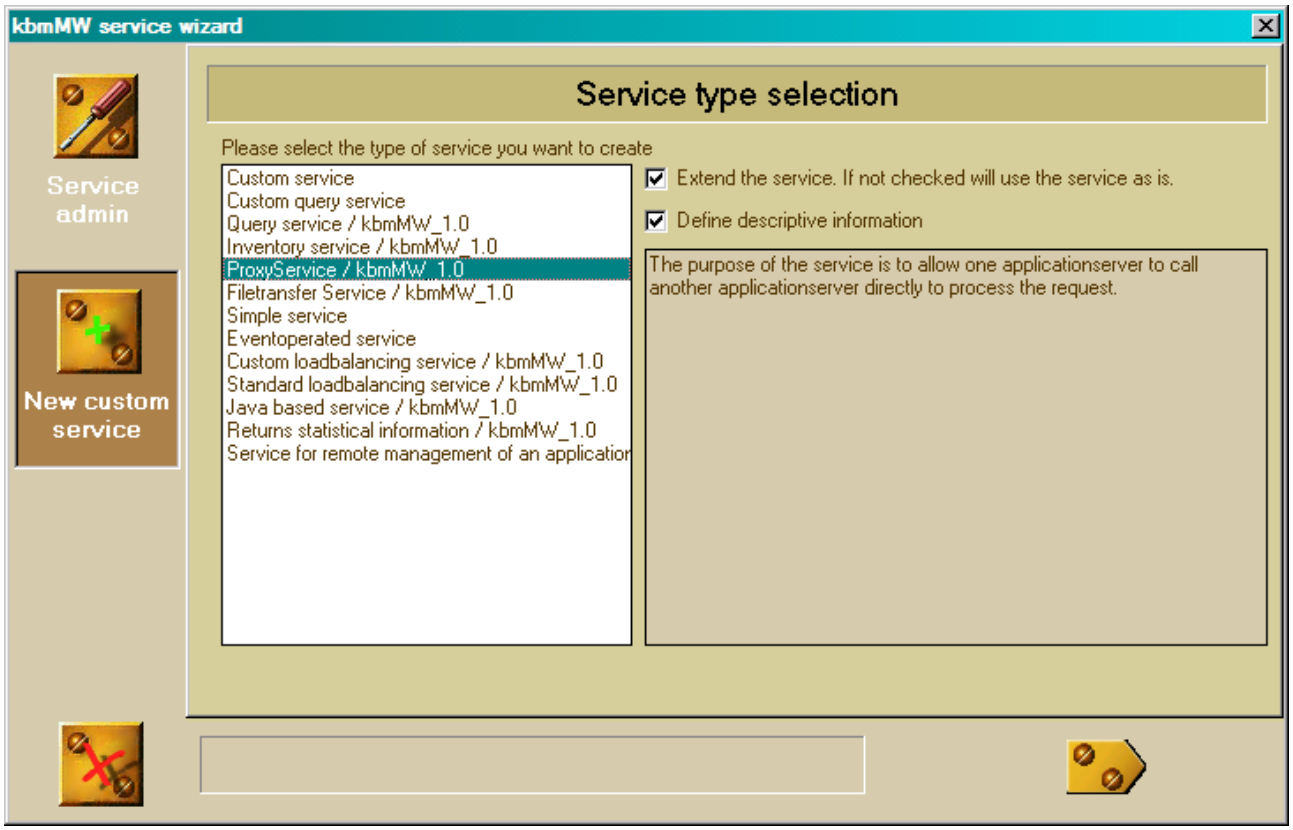
First you need to decide how the application server containing the proxy functionality (now called the proxy server) should communicate with other business application servers. If the business application servers can be reached by TCPIP, then you should add functionality to the proxy server to be able to act as a TCPIP client to the business application servers.

Thus add a TkbmMWClientConnectionPool and a TkbmMWTCPIPClientTransport to the proxy servers main form/data module (the one containing TkbmMWServer).

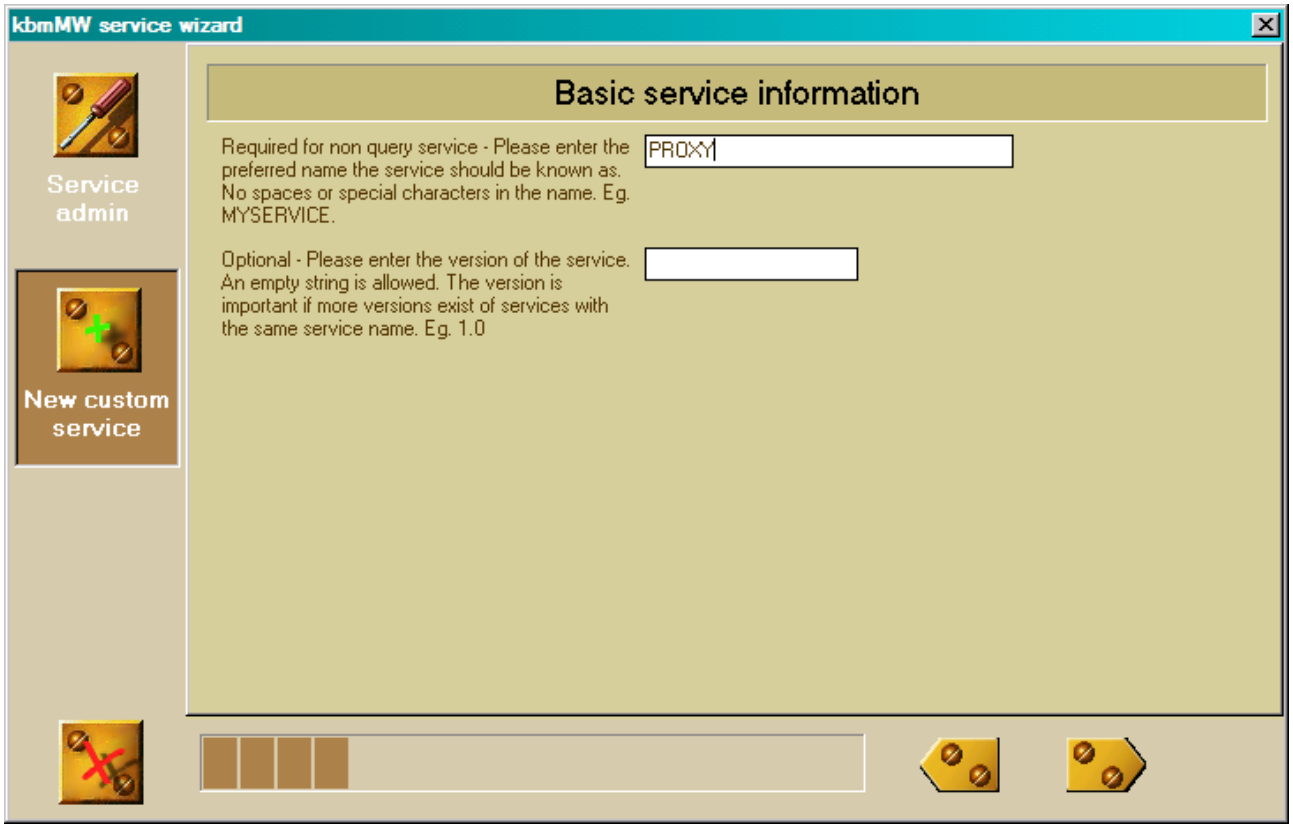


Hook up the client connection pool to the client transport, and configure the client transport to be able to access the business application server.

Then create a new proxy service using the kbmMW service wizard:

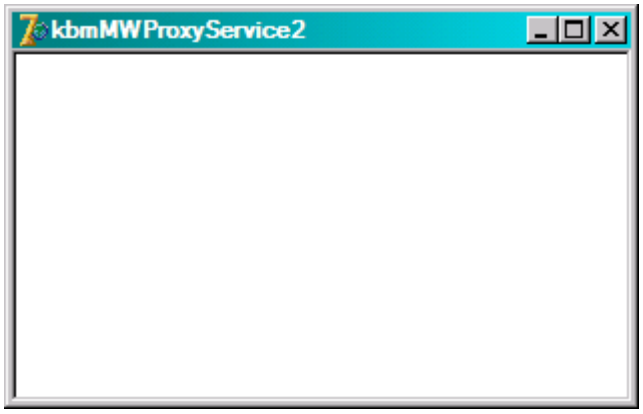


As with any other custom service, give it a preferred name and fill in any other optional service information.



You don't need to add any service functions to the service, since the required functionality is already in place in the proxy service's 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 data module as usual.



Next step is to make the proxy service reference the client connection pool, so it knows how to get to the business application server for which this proxy is targeted.
Set the ConnectionPool property of the proxy service data module to point on the client connection pool on the main form.

Registering the service

The proxy service module must now be registered in the TkbmMWServer.
The options you have in registering the proxy service makes provides you with great flexibility in if your proxy server should be a simple pass-through to one internal business application server, or if it should be accessing multiple internal business application servers.

The options are:

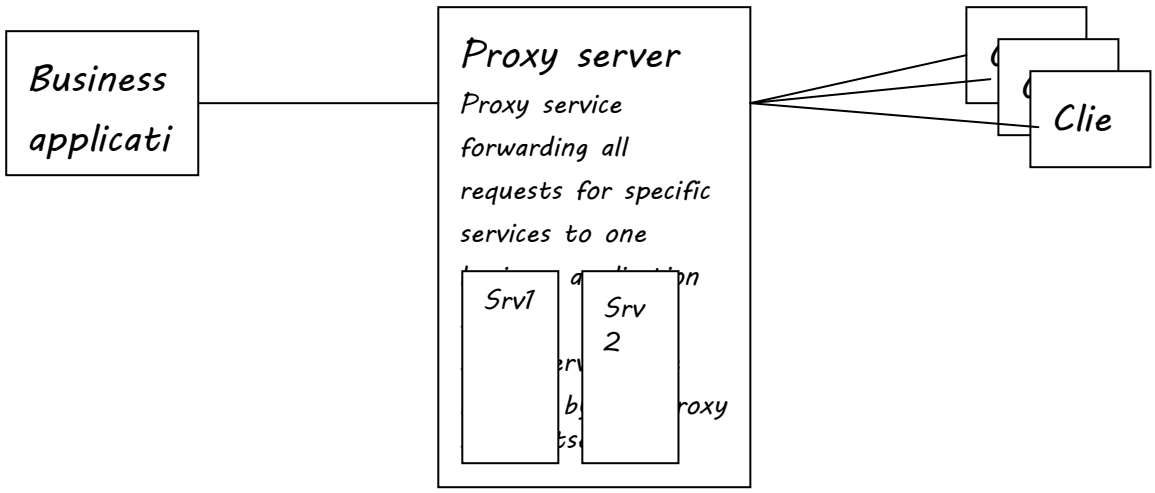
1) Clients thru proxy to one business application server.

This setup is handled by registering the service like this:

```
var
  sd:TkbmMWCustomServiceDefinition;
begin
     sd:=kbmMWServer1.RegisterService(TMyProxyService1,true);
end;
```

The keyword here is 'true' in the 2nd argument of the register service call. It means that the service TMyProxyService1 is the default service for all client requests. That in turn means that the proxy service will auto forward all client requests to the business application server pointed to by the client transport on the main module.

2) Clients thru proxy for specific services to one business application server. Some services are provided by the proxy server itself.



This setup is handled by registering the service like this:

```
var
  sd:TkbmMWCustomServiceDefinition;
begin
     sd:=kbmMWServer1.RegisterService(TMyInternalService1,false);
     sd:=kbmMWServer1.RegisterService(TMyInternalService2,false);
     sd:=kbmMWServer1.RegisterService(TMyProxyService1,true);
end;
```

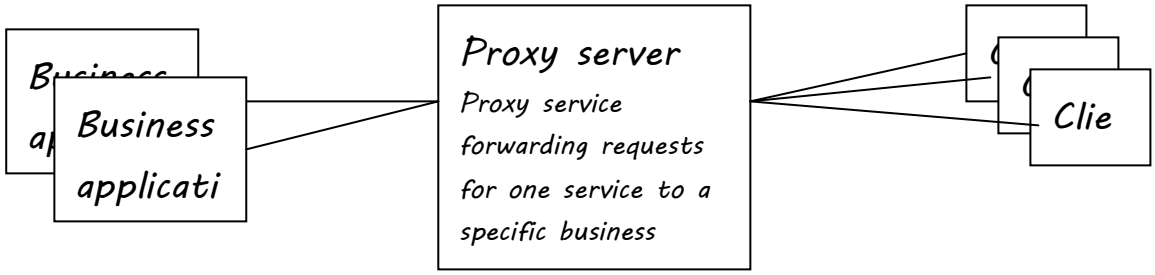
This way, the proxy server handles all requests from clients to internal service 1 and 2, while all others are proxied to one business application server.

Alternatively:

```
var
  sd:TkbmMWCustomServiceDefinition;
begin
     sd:=kbmMWServer1.RegisterService(TMyInternalService1,false);
     sd:=kbmMWServer1.RegisterService(TMyInternalService2,false);
     sd:=kbmMWServer1.RegisterServiceByName('SOMESERVICE1',TMyProxyService1,
           false);
     sd:=kbmMWServer1.RegisterServiceByName('SOMESERVICE2',TMyProxyService1,
           false);
end;
```

This way, the proxy server handles all requests from clients to internal service 1 and 2, proxies off requests for services with name SOMESERVICE1 or SOMESERVICE2 to another business application server, and if client is asking for any other service, the client will right away, receive an exception that the service is not available.

3) Client requests thru proxy who selects appropriate business application server for the job.



This setup is handled by creating multiple proxy services, each pointing on their own client connection pool, that in turn points on each their own business application server.

```
var
  sd:TkbmMWCustomServiceDefinition;
begin
     sd:=kbmMWServer1.RegisterServiceByName('SOMESERVICE1',TMyProxyService1,
           false);
     sd:=kbmMWServer1.RegisterServiceByName('SOMESERVICE2',TMyProxyService2,
           false);
end;
```

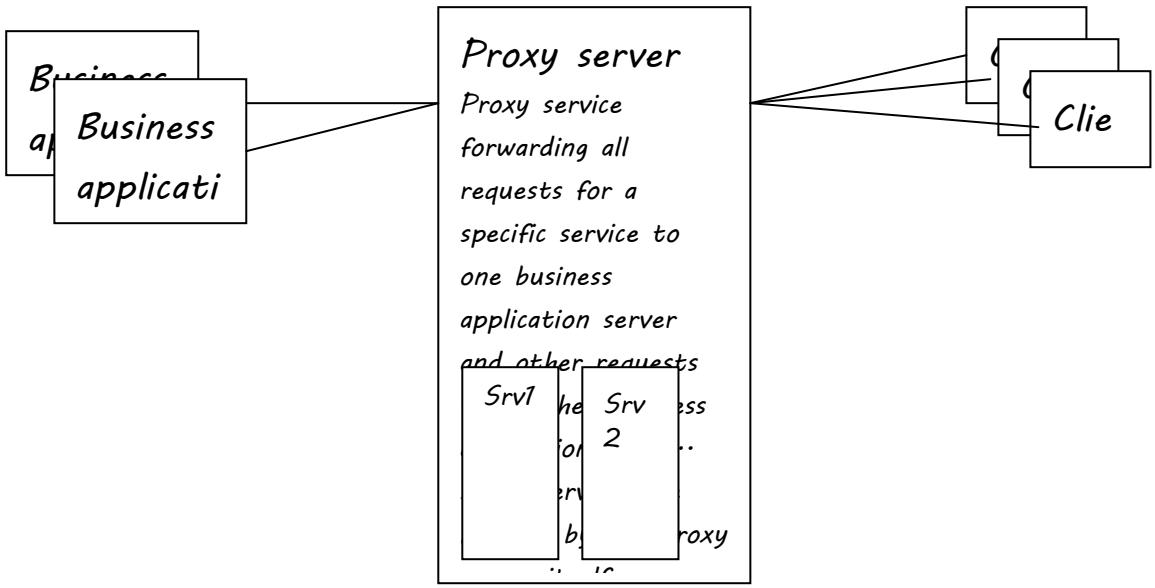
If the client request service 'SOMESERVICE1' the request will be handled by the server pointed on by proxy service 1 etc. If the client ask for another service than SOMESERVICE1 or 2, the client will receive an exception about service not available straight away.

Alternatively:

```
var
  sd:TkbmMWCustomServiceDefinition;
begin
     sd:=kbmMWServer1.RegisterServiceByName('SOMESERVICE1',TMyProxyService1,
           false);
     sd:=kbmMWServer1.RegisterService(TMyProxyService2,true);
end;
```

If the client request service 'SOMESERVICE1' the request will be handled by the server pointed on by proxy service 1. For all other service requests, the call will be proxied to the business application server pointed to by proxy service 2.

4) Client request thru proxy who selects appropriate business application server, or its own internal services for the job.



This is simply a combination of the previous options.

Additional information

Proxying is a very handy functionality, but it also comes with a price.
Since the proxy server will first get and interpret the request, to be able to decide what to do with it, and then will create the request a new to the business application server, there are some overhead both in memory usage, and in the time taken to process a request.

However unless you try to transfer multi megabytes of data, you probably wont notice it.

Proxying is completely transparent in the sense that responses or exceptions thrown on a business application server will be proxied back to the client making the original request and thus the client will never know that if it has been proxied or not.

On ISAPI based application servers, I usually recommend only defining a default proxy server, simply because it this way will be possible to add new functionality in the business application servers without having to down and up the webserver hosting the proxy.

Also notice that its possible to define fail over capabilities and loadbalancing capabilities in the proxy the same way as you do on a client, simply because the proxy acts as a full fledged client towards the internal business application servers.

This completes ‘The Proxyservice’ whitepaper.

Kim Madsen
Components4Developers