ADD A LINKING SERVICE TO ANOTHER SERVICE

A linking service can be used to create and store dynamic linking information for its parent service. It maintains the link paths, weights and references. Before you can create dynamic links, you need to add a linking service to your main service, as a utility or child service. The linking service should be created with the same passwords as the parent service, so that access to it is automatically known. To update the link structure, you then typically make method calls on the linking service itself. You should also add the service with the default 'UUID' and 'Type' of `ServiceConst.LINKER`, so that it can always be found. For example, the 'Service.dynamicLinksToXml()' method needs to automatically find it. The path to invoke a method on the linking service might therefore be:

```
<Handle><U>http://123.4.5.6:8888/</U><S>S1</S><S>Linker</S></Handle>
```

After adding the linking service, you still need to initialise the linking structure, where the required information includes the threshold values for each level, number of allowed entries at each level, weight update values, etc. The licas documentation gives complete details on this, where the following code describes how to create and initialise a linking service.

ADD A LINKING SERVICE

You can add a linking service to any other service, including all of the services running on a server. To add it to all services, you need to retrieve all of the service names and then call each one in turn to add a linking service to it. To add to a specific service, you can then just invoke one service instead. The following code describes how this can be done:

```
//some variables that might be required
int i;
boolean isOK;              //true if call is OK
String nextServiceName;    //next service name
String serverPassword;     //server password
Vector serviceNames;       //list of service names
Vector params;             //list of parameters
Element serviceUri;        //service uri
MethodInfo methodInfo;     //used to store the method information
CallObject callObject;     //used to make the remote method call


//initialise the calling mechanism
callObject = new CallObject();

//retrieve the server url from somewhere. The server uri should have a default service name that is the same as Const.HTTPSERVER. As calling the server, this is also the service uri.
serviceUri = <Handle><U>http://???:port/</U><S>HttpServer</S></Handle>
```

```
//passwords can be stored under the full uri address or just the uuid name
serverPassword = passwordHandler.getPassword(serverUri)

//get the names of all base services on the server - need to make a call to the server itself
methodInfo = MethodFactory.createMethodCall(MethodConst.GETSERVICENAMES,
TypeConst.VECTOR, serverUri, new Vector(), passwordHandler);

//invoke the server method to get service names
serviceNames = (Vector)callObject.call(methodInfo);

//call each service on the server in turn and add a linking service to it
for (i = 0; i < serviceNames.size(); i++)
{
    try
    {
        //can create the path to each service from the server url,
        //by retrieving the ip address part and then adding the service uuid to it.
        //Each service is stored using this uri format,
        //with child services only requiring further nesting
        nextServiceName = (String)serviceNames.elementAt(i);
        serviceUri = Handle.createNewUrlHandle(Handle.getURI(serverUri));
        serviceUri = Handle.addToHandle(serviceUri,
                        Handle.asHandleElement(nextServiceName));

        //new service constructor parameters
        //this uses the default password and service admin key values of 'anon'
        params = new Vector();
        params.add(Const.ANON);
        params.add(Const.ANON);

        //make a method call to add a new linking service – can also construct one manually
        methodInfo = new MethodInfo();
        methodInfo.setName(MethodConst.ADDSERVICE);     //add service method name
        methodInfo.setRtnType(TypeConst.BOOLEAN);       //method return type
        methodInfo.setServiceURI(serviceUri);
        methodInfo.setServerPassword(serverPassword);
        methodInfo.setPassword(passwordHandler.getPassword(nextServiceName));
        methodInfo.addParam(passwordHandler.getPassword(nextServiceName));
        methodInfo.addParam(ServiceConst.LINKER);       //linking service uuid
        methodInfo.addParam(ServiceConst.LINKER);       //linking service type (anything)
        methodInfo.addParam(new Vector());          //jar files, can be empty if nothing new
        methodInfo.addParam(Link_M.class.getName());    //linking service class description
        methodInfo.addParam(false);                     //start thread running
        methodInfo.addParam(params);                    //constructor parameters
        isOK = ((Boolean)callObject.call(methodInfo)).booleanValue();
    }
    catch (Exception ex) {}
}
```

INITIALISE THE LINKING SERVICE

The linking service then needs to be initialised with threshold and weight values. It uses these to update the values for each link that it stores. The following parameters are required:

- *float:* `linkThreshold` - this is the threshold value that must be passed for a link to reach the link level.
- *float:* `monitorThreshold` - this is the threshold value that must be passed for a link to reach the monitor level.
- *int:* `linkNumber` - this is the maximum allowed number of references at the top link level.
- *int:* `monitorNumber` - this is the maximum allowed number of references at the middle monitor level.
- *int:* `possibleNumber` - this is the maximum allowed number of references at the lowest possible links level.
- *float:* `increment` - this is the amount to increment an existing link by if it is used again. This is also the initial link value.
- *float:* `weight` - this weights the amount to decrement the link value by if it is not used but related references are. The decrement value is the increment value times this value.
- *String:* `thisLinkMethod` - this is a list of linking features that the linking mechanism should use. The LinkConfig class constants can help to construct this, for example `linkMethod = LinkConfig.addLinkMethod(linkMethod, LinkConfig.MEMORY)`. They should be in the form of a String that looks something like 'Link_Full:Memory:Stats:View'.
- *String :* `thisActivFunc` - a new option to try a different activation function. Probably only AiHeuristicConst.FUNCTIONLINEAR or FUNCTIONSIGMOID. Linear is the default, if nothing is entered.

This information is now passed in the form of a `LinkSpec` object, which includes all of the required values.

```
//you can make this sort of call on any linking service to configure its linking structure
//the service url is the parent service url plus the ServiceConst.LINKER uuid,
//which should be used for a linking service. It could be constructed by something like:

//get the server URL first
serviceUri = Handle.createNewUrlHandle(serverUrl);

//then add to it the service and default linker service names
serviceUri = Handle.addToHandle(serviceUri, Handle.asHandleElement(serviceName));
serviceUri = Handle.addToHandle(serviceUri,
                                        Handle.asHandleElement(ServiceConst.LINKER));

//set the initialisation parameters
linkSpec = new LinkSpec();
linkSpec.linkThreshold = linkThreshold;
linkSpec.monitorThreshold = monitorThreshold;
linkSpec.linkNumber = linkNumber;
```

```
linkSpec.monitorNumber = monitorNumber;
linkSpec.possibleNumber = possibleNumber;
linkSpec.linkIncrement = increment;
linkSpec.linkWeightDec = weight;
linkSpec.linkMethod = thisLinkMethod;
linkSpec.functionActivate = thisActivFunc;

//this call then traverses up to the linking service and invokes the setThresholds method
methodInfo = new MethodInfo();
methodInfo.setName(MethodConst.SETTHRESHOLDS);
methodInfo.setRtnType(TypeConst.VOID);
methodInfo.setServiceURI(serviceUri);
methodInfo.setServerPassword(serverPassword);
methodInfo.setPassword(servicePassword);
methodInfo.addParam(linkSpec);
call.call(methodInfo);
```