# AvailabilityContract

## Availability Contract : What is that ?

In a static, single-JVM application it is assumed that the structure of the application itself doesn't change during the course of execution. Objects are linked to other objects by the JVM itself, and things is rigid and safe.

In more dynamic environments, such as many J2EE containers, Servlet containers and network/distributed applications, such a solution is either not desirable (perhaps due to reloading requirements) or not possible (network can be temporary down).

Avalon should support dynamic environment more than the current Avalon 4 Framework does. And it is required that the Framework has explicit support for it, so the proper contract between the container and the various components can be established.

Availability Contract is all about that one component is INFORMED when another component becomes available and unavailable for access. The contract DOES NOT concern itself about the loading/unloading of components and all the related issues around that.

### Availability in Avalon 4

In Avalon 4, the ServiceManager is closely related to the Availability Contract. The current contract is, when

```
void service( ServiceManager manager ) throws ServiceException
```

is called, all serviced components ARE AVAILABLE for lookup. Furthermore, the Avalon 4 Framework also stipulates that components are not removed or disappear during execution.

So what would be needed?

First of all we need the AvailablityAware interface, which is both the marker interface for that the component is capable of handling components that disappears and reappears. But the interface may as well contain the callback methods for the Availabilty notifications. So we end up with something like this;

```
public interface AvailabilityAware
{
    void available( String lookupKey, Object instance );

    void unavailable( String lookupKey, Object instance );
}
```

Secondly, we need a mechanism to ***subscribe*** to the component. This should be done with the exact same query mechanism being used for the lookup(), which in Avalon 4's ServiceManager is just a String, potentially meaning anything. Any more advanced query or query template is beyond this contract and should be addressed elsewhere.

But how do we inform what we are interested in?

Well, we can't add methods to the ServcieManager interface, as this would break compatibility. So we can either extend the ServiceManager, and downcast in the service() method, or we could pass the AvailabilityManager in a separate method call to the component. Such call could then be bundled into the above AvailabilityAware interface, but one could imagine that you have listeners that doesn't receives the Manager. I leave this choice out of the discussion. What is important is that the Manager in question has two methods;

```
void registerAvailibiltyObserver( AvailabilityAware observer );

void unregisterAvailibiltyObserver( AvailabilityAware observer );
```

And of course the names can be discussed, as well as whether the JavaBean Event pattern should be used instead, and many other picky details. I am just trying to describe the principle.

What compatibility effects would the introduction of this contract have?

- First of all, no container and no component support it today.
- Secondly, if a non-AvailabilityAware component is used in a capable container, there are no issues what so ever.
- Thirdly, if a new Availability-aware component is used in a non-capable container, then it is required that the container is updated with the new avalon-framework JAR, so that the component can be loaded, but otherwise no issues.

Conclusion: It would be possible to introduce the Availability Contract in Avalon 4 Framework without much impact on existing containers and no impact on existing components, and slowly upgrade the supported containers.

### Availability in Avalon 5

Instead of doing an extension of Avalon 4's ServiceManager, there are several ways an Availability Contract can be implemented in a *fresh* framework. One should probably address this hand-in-hand with ComponentQuery contract, since they are closely related. That includes registering a ServiceTemplate with the ServiceManager of what is of interest, which is the same template used for lookups, which may return more than one component.

Another related issue in a new framework would be that the Availability contract becomes the center of all dependency resolution, and the ServiceManager is modified accordingly.

## Container Issues

It is not straight forward for the containers to support the Availability Contract, especially if non-AvailabilityAware components are allowed to depend on components that may be removed. In such case, some form of interceptor between the non-capable component and the removable component has to be put in place. For Merlin, such functionality would go into an existing Proxy that typically wraps all components today. Furthermore, the Availability Contract would only be needed if components can be added and removed on-the-fly. Such process has its own pitfalls and tricky bits, which needs to be addressed elsewhere. Merlin has in February 2004 got the ability to modify the components used during runtime, but at the time of this writing, it still has a long way to go before it is a solid functionality.