# A simple idea becomes powerfull

### SIMPLE IDEA

Let's make a library that abstracts primary service providers with an interface.

#### BENEFITS

* Single library and process for all services, instead of knowing each vendors process.
* Technical debt for upgrades moved from your application to library owner.
* Reduced Architecture debt. Easily move a single portion of code between local caching, to network caching if something needs to scale.
* Reduced library impedance. Information should freely flow between your different versions independent service providers.
* Increased ramp up. The library should pass the benefits of learning the native provider off to you. With increased time to use. And configurable overrides. The speed to get a service bus up and running for someone who has not used one before should be quick.

Create generic interfaces for the primary service providers with implementations for the most common. And allow the generic interface to get to its base native types for exceptional requirements. Note: using the base type will incur some technical debt back to your application.

The library should support multiple registrations. You may need to register multiple different cache providers, or have the same provider registered more than once with different configurations.

Additional benefits from a pure abstraction:

* Being able to switch from local caching like we cache to shared caching like memcache with one registration change.
* Transparent service provider upgrades. Move to the latest version of a service with less or no change. BCL has already incurred the technical debt of upgrading.
* Single interfaces assembly needed in dependent assemblies. All services available without a bunch of registrations, dependent on if U use the base types.

Library should handle less-foreseen dependency impedances, and incur this technical debt.

Sometimes you have a dependency impedance where two libraries uses different versions of a shared service like a logger. Which can typically be handled easily with an assembly redirection. Using an abstracted interface removes this impedance. But in and higher impedance problem, you may have two libraries using different IoC containers, this happens a lot in the more complex space of enterprise service busses. The service abstraction implementation should unify all of these services, removing the impedance.

### MAKE IT LAZY

Then the next easy step, is making these providers lazy load so they can be defined without the usage overhead till you need it.

Go ahead and centrally register every service you might use, only the ones you end up using get loaded. Note: If you tell your service to register with service locator [Provider.RegisterWithServiceLocator()] they will load at startup to place in your container.

### ENHANCE SERVICES

Once you have this frame work. Service enhancements can easily be applied.

To things like:

* Service locator type registration by scan. Which is available in some providers. But now you can do for all, with a consistent, and transparent pattern. Show what the scan is matching and skipping and why.
* Look-aside caching pattern similar to the application block.

Network or system cache dependencies for IPC data synchronization.

Or with wrapping interfaces:

Subdividing a caching space by a string key Scoped changing of behaviors, for instance a service locator registration from transitional to static.

### A SINGLE ASSEMBLY

Finally, a suite of services can be bundled into a single assembly since they all rely on a single set of interfaces. And optionally be used in your project with one reference addition, providing you numerous services from different providers.

### LIBRARIES

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Service Type** | **Name** | **Package ID** | **.net35** | **.net40** | | **.net45** |
| Event Source | EventStore | EventStore | [3.0.11326.44] | 3.2.0.28 | | |
|  | MongoDB | mongocsharpdriver | 1.8.3 | | | |
|  | MSSql | MSSql | System | | | |
|  | NEventStore | NEventStore-Signed | N/A | 4.1.0.10 | | |
|  | SQLite | System.Data.SQLite | 1.0.91.3 | | | |
| Service Bus | MTServiceBus | MassTransit | 2.9.8 | | | |
|  | NServiceBus | NServiceBus | [2.6.0.1511] | 4.4.2 | | |
|  | RabbitMq | RabbitMQ.Client | 3.3.5 | | | |
|  | RhinoServiceBus | Rhino.ServiceBus | 3.1.1.0 | | | |
|  | ZeroMQ | clrzmq/clrzmq-x64 | 2.2.5 | | | |
|  | \*AppServiceBus |  | System | | | |
| Service Cache | Memcached | EnyimMemcached | 2.12 | | | |
|  | ServerAppFabric | ServerAppFabric.Client | 1.1.2106.32 | | | |
|  | Redis | StackExchange.Redis | N/A | | 1.0.394 | |
|  | \*StaticServiceCache |  | System | | | |
|  | \*SystemServiceCache |  | System | | | |
| Service Locator | Autofac | Autofac | [2.6.3.862] | 3.3.0 | | |
|  | CastleWindsor | Castle.Windsor | 3.2.0 | | | |
|  | Hiro | Hiro | 1.0.3 | | | |
|  | Munq | Munq.IocContainer | N/A | 3.1.6 | | |
|  | Ninject | Ninject | 3.0.1.10 | | | |
|  | Spring | Spring.Core | 1.3.2 | | | |
|  | StructureMap | structuremap | [2.6.3] | 2.6.4.1 | | |
|  | Unity | Unity | [2.1.505.2] | [2.1.505.2] | | 3.0.1304.1 |
|  | \*MicroServiceLocator |  | System | | | |
| Service Log | CommonLogging | Common.Logging | 2.3.1 | | | |
|  | Log4Net | log4net | 2.0.3 | | | |
|  | NLog | Nlog | 3.1.0.0 | | | |
|  | \*ConsoleServiceLog |  | System | | | |
|  | \*EventLogServiceLog |  | System | | | |
|  | \*StreamServiceLog |  | System | | | |
|  | \*TraceSourceServiceLog |  | System | | | |
| Service Map | AutoMapper | AutoMapper | 1.1.0.118 | 3.2.1 | | |