AKKA.NET CHEAT SHEET

NUGET PACKAGES

Main: Akka - Akka.Remote - Akka.Persistence - Akka.Cluster

Logging: Akka.Logger.NLog - Akka.Logger.Serilog - Akka.Logger.slf4net

DI: Akka.DI.Ninject - Akka.DI.Unity - Akka.DI.StructureMap - Akka.DI.AutoFac - Akka.DI.CastleWindsor

DEFINING ACTORS

UNTYPED

```
class MyUntypedActor : UntypedActor
{
    protected override void OnReceive(object message)
    {
        if (message is Order) { /*...*/ }
    }
}
```

RECEIVE ACTOR

```
class MyReceiveActor : ReceiveActor
{
    public MyReceiveActor()
    {
        Receive<Order>(message => { /*...*/ });
        // Predicate message filters
        Receive<Order>(handleIf => handleIf.IsVip ,message => { /*...*/ });
        Receive<Order>(message => { /*...*/ }, handleIf => handleIf.IsVip);
    }
}
```

CREATION

```
ActorSystem system = ActorSystem.Create("MyUriFriendlyActorSystemName");

// Top level actors
IActorRef myActor1 = system.ActorOf<MyReceiveActor>("NameOfMyActor1");

Props props = Props.Create<MyReceiveActor>();
IActorRef myActor2 = system.ActorOf(props, "NameOfMyActor2");

Props props2 = Props.Create(() => new MyReceiveActor());
IActorRef myActor3 = system.ActorOf(props2, "NameOfMyActor3");

SupervisorStrategy strat = null; // = new AllForOneStrategy(...);
Props props3 = Props.Create(() => new MyReceiveActor(), strat);
IActorRef myActor4 = system.ActorOf(props3, "NameOfMyActor4");
```

CHILD ACTORS

```
class MyReceiveActorWithChildren : ReceiveActor
{
    public MyReceiveActorWithChildren()
    {
        Context.ActorOf(Props.Create<Child>(), "AChild");
    }
}
```

SWITCHABLE BEHAVIOURS

```
class MyBehaviorActor : ReceiveActor
{
    public MyBehaviorActor()
    {
        // Initial behavior
        Happy();
    }

    private void Happy()
    {
        Receive<MeetingRequest>(message => Become(Sad));
    }

    private void Sad()
    {
        Receive<MeetingCancelled>(message => Become(Happy));
    }
}
```

```
    // Stacked behaviour API
    Receive<MeetingRequest>(message => BecomeStacked(Sad));
    Receive<MeetingRequest>(message => UnbecomeStacked());
```

MESSAGE STASHING

```
class MyStashingActor: ReceiveActor, IWithUnboundedStash
{
    public IStash Stash { get; set; }

    public MyStashingActor()
    {
        Receive<string>(message => Stash.Stash());
        Receive<int>(message => Stash.Unstash());
        Receive<char>(message => Stash.UnstashAll());
    }
}
```

LOGGING

```
class MyLoggingActor : ReceiveActor
{
    private readonly ILoggingAdapter _log = Context.GetLogger();

    public MyLoggingActor()
    {
        _log.Debug("...");
        _log.Info("...");
        _log.Warning("...");
        _log.Error("...");
    }
}
```

LIFECYCLE EVENTS

```
class MyLifecycleActor : ReceiveActor
{
    protected override void PreStart()
    {
        base.PreStart();
    }

    protected override void PreRestart(Exception reason, object message)
    {
        base.PreRestart(reason, message);
    }

    protected override void PostRestart(Exception reason)
    {
        base.PostRestart(reason);
    }

    protected override void PostStop()
    {
        base.PostStop();
    }
}
```

PLURALSIGHT COURSES

http://bit.ly/psjasonroberts

BLOG POSTS

http://dontcodetired.com/blog/?tag=/akka.net

PROJECT DOCUMENTATION

http://getakka.net/docs/