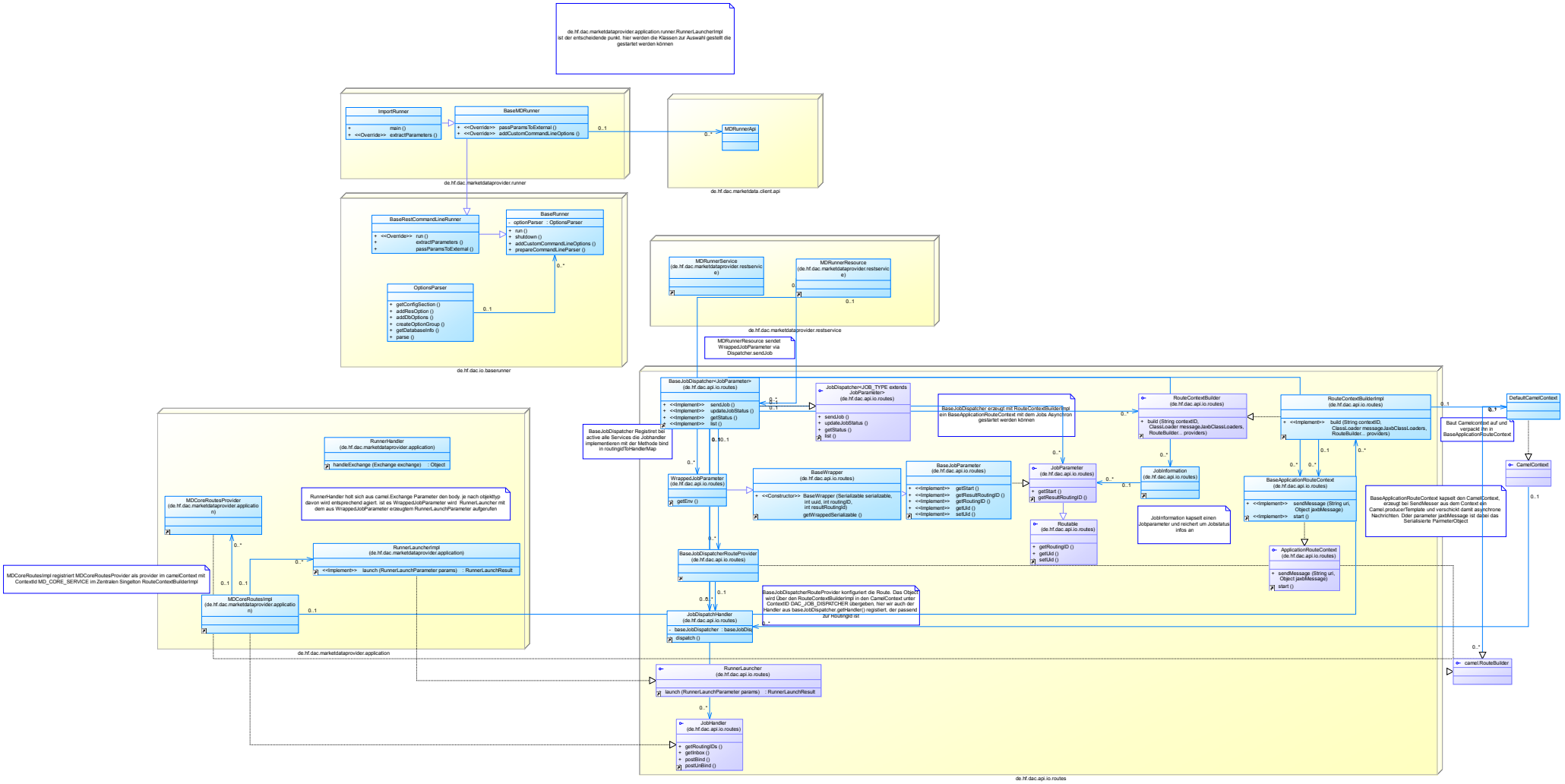
de.hf.dac.marketdataprovider.application.runner.RunnerLauncherImpl ist der entscheidende punkt. hier werden die Klassen zur Auswahl gestellt die gestartet werden können
ImportRunner
+ main ()
+ <<Override>> extractParameters ()
BaseMDRunner
- <<Override>> passParamsToExternal ()
+ <<Override>> addCustomCommandLineOptions ()
0..1
0..*
MDRunnerApi
de.hf.dac.marketdataprovider.runner
de.hf.dac.marketdata.client.api
BaseRestCommandLineRunner
+ <<Override>> run ()
+ extractParameters ()
+ passParamsToExternal ()
BaseRunner
- optionParser : OptionsParser
+ run ()
+ shutdown ()
+ addCustomCommandLineOptions ()
+ prepareCommandLineParser ()
0..*
OptionsParser
+ getConfigSection ()
+ addResOption ()
+ addDbOptions ()
+ createOptionGroup ()
+ getDatabaseInfo ()
+ parse ()
0..1
de.hf.dac.io.baserunner
MDRunnerService (de.hf.dac.marketdataprovider.restservice)
MDRunnerResource (de.hf.dac.marketdataprovider.restservice)
0
0..1
de.hf.dac.marketdataprovider.restservice
MDRunnerResource sendet WrappedJobParameter via Dispatcher.sendJob
BaseJobDispatcher<JobParameter> (de.hf.dac.api.io.routes)
+ <<Implement>> sendJob ()
+ <<Implement>> updateJobStatus ()
+ <<Implement>> getStatus ()
+ <<Implement>> list ()
0..1
0..1
JobDispatcher<JOB_TYPE extends JobParameter> (de.hf.dac.api.io.routes)
+ sendJob ()
+ updateJobStatus ()
+ getStatus ()
+ list ()
BaseJobDispatcher erzeugt mit RouteContextBuilderImpl ein BaseApplicationRouteContext mit dem Jobs Asynchron gestartet werden können
0..*
RouteContextBuilder (de.hf.dac.api.io.routes)
+ build (String contextID, ClassLoader message.JaxbClassLoaders, RouteBuilder... providers)
RouteContextBuilderImpl (de.hf.dac.api.io.routes)
+ <<Implement>> build (String contextID, ClassLoader message.JaxbClassLoaders, RouteBuilder... providers)
0..1
0..1
DefaultCamelContext
Baut CamelcontXt auf und verpackt ihn in BaseApplicationRouteContext
BaseJobDispatcher Registriret bei active alle Services die Jobhandler implementieren mit der Methode bind in routingIdToHandlerMap
0..*
0..1
0..1
0..*
0..*
0..1
0..*
0..*
0..*
0..*
0..1
0..1
0..*
CamelContext
0..1
WrappedJobParameter (de.hf.dac.api.io.routes)
getEnv ()
BaseWrapper (de.hf.dac.api.io.routes)
+ <<Constructor>> BaseWrapper (Serializable serializable, int uuid, int routingID, int resultRoutingID)
getWrappedSerializable ()
BaseJobParameter (de.hf.dac.api.io.routes)
+ <<Implement>> getStart ()
+ <<Implement>> getResultRoutingID ()
+ <<Implement>> getRoutingID ()
+ <<Implement>> getUId ()
+ <<Implement>> setUId ()
JobParameter (de.hf.dac.api.io.routes)
+ getStart ()
+ getResultRoutingID ()
0..*
0..1
JobInformation (de.hf.dac.api.io.routes)
JobInformation kapselt einen Jobparameter und reichert um Jobstatus infos an
BaseApplicationRouteContext (de.hf.dac.api.io.routes)
+ <<Implement>> sendMessage (String uri, Object jaxbMessage)
+ <<Implement>> start ()
BaseApplicationRouteContext kapselt den CamelContext, erzeugt bei SendMesser aus dem Context ein Camel.producerTemplate und verschickt damit asynchrone Nachrichten. Dder parameter jaxbMessage ist dabei das Serialisierte ParmeterObject
Routable (de.hf.dac.api.io.routes)
+ getRoutingID ()
+ getUId ()
+ setUId ()
ApplicationRouteContext (de.hf.dac.api.io.routes)
+ sendMessage (String uri, Object jaxbMessage)
start ()
BaseJobDispatcherRouteProvider (de.hf.dac.api.io.routes)
0..1
0..*
0..1
BaseJobDispatcherRouteProvider konfiguriert die Route. Das Object wird Über den RouteContextBuilderImpl in den CamelContext unter ContextID DAC_JOB_DISPRTCHER Übergeben. Her wir auch der Handler aus baseJobDispatcher.getHandler() registiert, der passend zur routingID ist
0..*
JobDispatchHandler (de.hf.dac.api.io.routes)
- baseJobDispatcher : baseJobDis
dispatch ()
RunnerLauncher (de.hf.dac.api.io.routes)
launch (RunnerLauncherParameter params) : RunnerLauncherResult
0..*
JobHandler (de.hf.dac.api.io.routes)
+ getRoutingIDs ()
+ getInbox ()
+ postBind ()
+ postUnBind ()
0..*
camel.RouteBuilder
de.hf.dac.api.io.routes
RunnerHandler (de.hf.dac.marketdataprovider.application)
handleExchange (Exchange exchange) : Object
MDCoreRoutesProvider (de.hf.dac.marketdataprovider.application)
RunnerHandler holt sich aus camel.Exchange Parameter den body. je nach objekttyp davon wird entsprechend agiert. ist es WrappedJobParameter wird RunnerLauncher mit dem aus WrappedJobParameter erzeugtem RunnerLaunchParameter aufgerufen
0..*
0..*
RunnerLauncherImpl (de.hf.dac.marketdataprovider.application)
<<Implement>> launch (RunnerLauncherParameter params) : RunnerLauncherResult
MDCoreRoutesImpl registriert MDCoreRoutesProvider als provider im camelContext mit ContextId MD_CORE_SERVICE im Zentralen Singelton RouteContextBuilderImpl
0..1
0..1
MDCoreRoutesImpl (de.hf.dac.marketdataprovider.application)
0..1
de.hf.dac.marketdataprovider.application