**sdk**

## SDK - Setup()

```
heroes-service / blockchain / setup.go
// Initialize the SDK with the configuration file
sdk := fabsdk.New(config.FromFile(setup.ConfigFile))

// The resource management client is responsible for
// managing channels (create/update channel)
resourceManagerClientContext :=
sdk.Context(fabsdk.WithUser("admin"),
fabsdk.WithOrg("org1"))

resMgmtClient :=
resmgmt.New(resourceManagerClientContext)

sdk.admin = resMgmtClient

mspClient := mspclient.New(sdk.Context(),
mspclient.WithOrg("org1"))

adminIdentity := mspClient.GetSigningIdentity("admin")

req := resmgmt.SaveChannelRequest{ChannelID:
"mychannel", ChannelConfigPath: "channel.tx",
SigningIdentities: []msp.SigningIdentity{adminIdentity}}

txID := sdk.admin.SaveChannel(req,
resmgmt.WithOrdererEndpoint("orderer-id"))

// Make admin user join the previously created channel
sdk.admin.JoinChannel("channel",
resmgmt.WithRetry(retry.DefaultResMgmtOpts),
resmgmt.WithOrdererEndpoint("orderer-id"))
```

## CONFIG - FromFile()

```
fabric-sdk-go / pkg / core / config / config.go
// FromFile reads from named config file
// and returns a function to parse config file
func FromFile(filename, options) core.ConfigProvider {
    return func() ([]core.ConfigBackend, error) {
        backend := newBackend(opts...)

        // create new viper
        backend.configViper.SetConfigFile(name)

        // If a config file is found, read it in.
        backend.configViper.MergeInConfig()
    }
}
```

## CONFIG - ConfigProvider()

```
fabric-sdk-go / pkg / common / providers / core / provider.go
//ConfigProvider provides config backend for SDK
type ConfigProvider func() ([]ConfigBackend, error)

//ConfigBackend backend for all config types in SDK
type ConfigBackend interface {
    Lookup(key string) (interface{}, bool)
}
```

## CONFIG - newBackend()

```
fabric-sdk-go / pkg / core / config / config.go
func newBackend(opts) (*defConfigBackend) {
    o := options { envPrefix: cmdRoot, }
    for _, option := range opts { err := option(&o) }
    v := newViper(o.envPrefix)

    //default backend for config
    backend := &defConfigBackend{
        configViper: v,
        opts:       o,
    }
    err := backend.loadTemplateConfig()
    return backend, nil
}
```

## fabsdk - New()

```
fabric-sdk-go / pkg / fabsdk / fabsdk.go
// New initializes the SDK based on the options provided.
// ConfigOptions provides the application configuration.
func New(configProvider, opts) (*FabricSDK) {
    pkgSuite := defPkgSuite{}
    return fromPkgSuite(configProvider, &pkgSuite, opts...)
}
```

## fabsdk - FabricSDK {}

```
fabric-sdk-go / pkg / fabsdk /
fabsdk.go
type FabricSDK struct {
    opts        options
    provider   *context.Provider
    cryptoSuite core.CryptoSuite
}
```

## context - Provider {}

```
fabric-sdk-go / pkg / context / context.go
//Provider implementation of Providers interface
type Provider struct {
    cryptoSuiteConfig      core.CryptoSuiteConfig
    endpointConfig         fab.EndpointConfig
    identityConfig         msp.IdentityConfig
    userStore              msp.UserStore
    cryptoSuite            core.CryptoSuite
    localDiscoveryProvider fab.LocalDiscoveryProvider
    signingManager         core.SigningManager
    idMgmtProvider         msp.IdentityManagerProvider
    infraProvider          fab.InfraProvider
    channelProvider        fab.ChannelProvider
}
```

## fabsdk - fromPkgSuite()

```
fabric-sdk-go / pkg / fabsdk / fabsdk.go
// fromPkgSuite creates an SDK based on the
// implementations in the provided pkg suite.
func fromPkgSuite(configProvider, pkgSuite, opts)
(*FabricSDK) {
    sdk := FabricSDK {
        opts: options {
            Core:    pkgSuite.Core(),
            MSP:     pkgSuite.MSP(),
            Service: pkgSuite.Service(),
            Logger:  pkgSuite.Logger(),
        },
    }
    initSDK(&sdk, configProvider, opts)
    return &sdk
}
```

## fabsdk - withUser()

```
fabric-sdk-go / pkg / fabsdk / context.go
// WithUser uses the named user to load the identity
func WithUser(username string) ContextOption {
    return func(o *identityOptions) error {
        o.username = username
        return nil
    }
}
```

## fabsdk - Context()

```
fabric-sdk-go / pkg / fabsdk / fabsdk.go
//Context creates and returns context client
// which has all the necessary providers
func (sdk *FabricSDK) Context(options
...ContextOption) ClientProvider {
    clientProvider := func() (Client, error) {
        identity := sdk.newIdentity(options...)
        return &Client{Providers: sdk.provider,
SigningIdentity: identity}
    }
    return clientProvider
}
```

## resmgmt - New()

```
fabric-sdk-go / pkg / client / resmgmt / resmgmt.go
// New returns a resource management client instance.
func New(ctxProvider context.ClientProvider, opts)(*Client) {
    ctx := ctxProvider()

    resourceClient := &Client{
        ctx: ctx,
    }
    for _, opt := range opts { opt(resourceClient) }

    if resourceClient.localCtxProvider == nil {
        resourceClient.localCtxProvider = func()(context.Local) {
            return contextImpl.NewLocal(
                func() (context.Client) {
                    return resourceClient.ctx, nil
                },
            )
        }
    }
    return resourceClient, nil
}
```

## fabsdk - initSDK()

```
fabric-sdk-go / pkg / fabsdk / fabsdk.go
func initSDK(sdk *FabricSDK, configProvider, opts) error { //nolint
    for _, option := range opts { option(&sdk.opts) }

    //Initialize configs if not passed through options
    cfg := sdk.loadConfigs(configProvider)

    // Initialize rand (TODO: should probably be optional)
    rand.Seed(time.Now().UnixNano())

    // Initialize state store
    userStore := sdk.opts.MSP.CreateUserStore(cfg.identityConfig)

    // Initialize Signing Manager
    signingManager := sdk.opts.Core.CreateSigningManager(sdk.cryptoSuite)

    // Initialize IdentityManagerProvider
    identityManagerProvider :=
sdk.opts.MSP.CreateIdentityManagerProvider(cfg.endpointConfig, sdk.cryptoSuite,
userStore)

    // Initialize Fabric provider
    infraProvider := sdk.opts.Core.CreateInfraProvider(cfg.endpointConfig)

    // Initialize local discovery provider
    localDiscoveryProvider :=
sdk.opts.Service.CreateLocalDiscoveryProvider(cfg.endpointConfig)

    channelProvider :=
sdk.opts.Service.CreateChannelProvider(cfg.endpointConfig)

    //update sdk providers list since all required providers are initialized
    sdk.provider =
context.NewProvider(context.WithCryptoSuiteConfig(cfg.cryptoSuiteConfig),
        context.WithEndpointConfig(cfg.endpointConfig),
        context.WithIdentityConfig(cfg.identityConfig),
        context.WithCryptoSuite(sdk.cryptoSuite),
        context.WithSigningManager(signingManager),
        context.WithUserStore(userStore),
        context.WithLocalDiscoveryProvider(localDiscoveryProvider),
        context.WithIdentityManagerProvider(identityManagerProvider),
        context.WithInfraProvider(infraProvider),
        context.WithChannelProvider(channelProvider))

    //initialize
    if pi := infraProvider.(providerInit) {
        pi.Initialize(sdk.provider)
    }
    if pi := localDiscoveryProvider.(providerInit) {
        pi.Initialize(sdk.provider)
    }
    if pi := channelProvider.(providerInit) {
        pi.Initialize(sdk.provider)
    }
}
```

## fabsdk - newIdentity()

```
fabric-sdk-go / pkg / fabsdk / context.go
func (sdk *FabricSDK) newIdentity(options)
(SigningIdentity) {
    opts := identityOptions {
        orgName:
sdk.provider.IdentityConfig().Client().Organization,
    }
    for _, option := range options { option(&opts) }

    // Checks for missing required options (org,user,etc.)
    mgr := sdk.provider.IdentityManager(opts.orgName)
    user := mgr.GetSigningIdentity(opts.username)
    return user
}
```

## fabsdk - loadConfigs()

```
fabric-sdk-go / pkg / fabsdk / fabsdk.go
//loadConfigs load config from config backend when configs
are not provided through opts
func (sdk *FabricSDK) loadConfigs(configProvider)
core.ConfigProvider) (*configs, error) {
    c := &configs{
        identityConfig:    sdk.opts.IdentityConfig,
        endpointConfig:    sdk.opts.endpointConfig,
        cryptoSuiteConfig: sdk.opts.CryptoSuiteConfig,
    }
    var configBackend []core.ConfigBackend
    if configProvider != nil {
        configBackend, err = configProvider()
    }
    //configs passed through opts take priority over default
ones
    // load crypto suite config
    c.cryptoSuiteConfig, err =
sdk.loadCryptoConfig(configBackend...)

    //Initialize cryptosuite once crypto Suite config is available
    err = sdk.initializeCryptoSuite(c.cryptoSuiteConfig)

    // load endpoint config
    c.endpointConfig, err =
sdk.loadEndpointConfig(configBackend...)

    // load identity config
    c.identityConfig, err =
sdk.loadIdentityConfig(configBackend...)
    sdk.opts.ConfigBackend = configBackend
    return c, nil
}
```

## defmsp - CreateUserStore()

```
fabric-sdk-go / pkg / fabsdk / factory / defmsp / mspfactory.go
type ProviderFactory struct {}
// CreateUserStore creates a UserStore using the SDK's default implementation
func (f *ProviderFactory) CreateUserStore(config msp.IdentityConfig)
(msp.UserStore, error) {
    stateStorePath := config.Client().CredentialStore.Path
    stateStore, err := kvs.New(&kvs.FileKeyValueStoreOptions{Path:
stateStorePath})
    userStore, err := mspimpl.NewCertFileUserStore1(stateStore)
    return userStore, nil
}
```

## fabsdk - options {}

```
fabric-sdk-go / pkg / fabsdk / fabsdk.go
type options struct {
    Core              sdkApi.CoreProviderFactory
    MSP               sdkApi.MSPProviderFactory
    Service           sdkApi.ServiceProviderFactory
    Logger            api.LoggerProvider
    CryptoSuiteConfig core.CryptoSuiteConfig
    endpointConfig    fab.EndpointConfig
    IdentityConfig    msp.IdentityConfig
    ConfigBackend     []core.ConfigBackend
}
```

## msp - UserStore {}

```
fabric-sdk-go / pkg / common / providers / msp / store.go
// UserStore is responsible for UserData persistence
type UserStore interface {
    Store(*UserData) error
    Load(IdentityIdentifier) (*UserData, error)
}
```

## msp - NewCertFileUserStore1 {}

```
fabric-sdk-go / /pkg / msp / certfileuserstore.go
// NewCertFileUserStore1 creates a new instance
of CertFileUserStore
func NewCertFileUserStore1(store core.KVStore)
(*CertFileUserStore, error) {
    return &CertFileUserStore{
        store: store,
    }, nil
}
```

## fabsdk - CoreProviderFactory {}

```
fabric-sdk-go / /pkg / fabsdk / api / factory.go
// CoreProviderFactory allows overriding of primitives and
the fabric core object provider
type CoreProviderFactory interface {
    CreateCryptoSuiteProvider(config
core.CryptoSuiteConfig) (core.CryptoSuite, error)
    CreateSigningManager(cryptoProvider
core.CryptoSuite) (core.SigningManager, error)
    CreateInfraProvider(config fab.EndpointConfig)
(fab.InfraProvider, error)
}
```

## msp - IdentityConfig {}

```
fabric-sdk-go / pkg / common / providers / msp / provider.go
//IdentityConfig contains identity configurations
type IdentityConfig interface {
    Client() *ClientConfig
    CAConfig(org string) (*CAConfig, bool)
```

## msp - CertFileUserStore {}

## msp - CertFileUserStore {}

fabric-sdk-go / /pkg / msp / certfileuserstore.go
// CertFileUserStore stores each user in a separate file.
// Only user's enrollment cert is stored, in pem format.
// File naming is <user>@<org>-cert.pem
type **CertFileUserStore** struct {
    store core.KVStore
}

## fabsdk - MSPProviderFactory {}

fabric-sdk-go / /pkg / fabsdk / api / factory.go
// MSPProviderFactory allows overriding providers of MSP services
type **MSPProviderFactory** interface {
    **CreateUserStore**(config msp.IdentityConfig) (msp.UserStore, error)
    CreateIdentityManagerProvider(config fab.EndpointConfig, cryptoProvider core.CryptoSuite, userStore msp.UserStore) (msp.IdentityManagerProvider, error)
}

## fabsdk - ServiceProviderFactory {}

fabric-sdk-go / /pkg / fabsdk / api / factory.go
// ServiceProviderFactory allows overriding default service providers (such as peer discovery)
type **ServiceProviderFactory** interface {
    CreateLocalDiscoveryProvider(config fab.EndpointConfig) (fab.LocalDiscoveryProvider, error)
    CreateChannelProvider(config fab.EndpointConfig) (fab.ChannelProvider, error)
}

core.CryptoSuite) (core.SigningManager, error)
    CreateInfraProvider(config fab.EndpointConfig) (fab.InfraProvider, error)
}

## keyvaluestore - New()

fabric-sdk-go / pkg / fab / keyvaluestore / filekeyvaluestore.go
// New creates a new instance of FileKeyValueStore using provided options
func **New**(opts *FileKeyValueStoreOptions) (*FileKeyValueStore, error) {
    if opts.KeySerializer == nil {
        // Default key serializer
        opts.KeySerializer = func(key interface{}) (string, error) {
            keyString, ok := key.(string)
            return path.Join(opts.Path, keyString), nil
        }
    }
    if opts.Marshaller == nil {
        opts.Marshaller = defaultMarshaller
    }
    if opts.Unmarshaller == nil {
        opts.Unmarshaller = defaultUnmarshaller
    }
    return &FileKeyValueStore{
        path:      opts.Path,
        keySerializer: opts.KeySerializer,
        marshaller:   opts.Marshaller,
        unmarshaller:  opts.Unmarshaller,
    }, nil
}

//IdentityConfig contains identity configurations
type **IdentityConfig** interface {
    Client() *ClientConfig
    CAConfig(org string) (*CAConfig, bool)
    CAServerCerts(org string) ([][]byte, bool)
    CAClientKey(org string) ([]byte, bool)
    CAClientCert(org string) ([]byte, bool)
    CAKeyStorePath() string
    CredentialStorePath() string
}

## msp - ClientConfig {}

fabric-sdk-go / pkg / common / providers / msp / provider.go
// ClientConfig provides the definition of the client configuration
type **ClientConfig** struct {
    Organization   string
    Logging       logApi.LoggingType
    CryptoConfig   **CCType**
    TLSKey        []byte
    TLSCert      []byte
    CredentialStore **CredentialStoreType**
}

## msp - CCType {}

fabric-sdk-go / pkg / common / providers / msp / provider.go
// CCType defines the path to crypto keys and certs
type **CCType** struct {
    Path string
}

## msp - CredentialStoreType {}

fabric-sdk-go / pkg / common / providers / msp / provider.go
// CredentialStoreType defines pluggable KV store properties
type **CredentialStoreType** struct {
    Path       string
    CryptoStore struct {  Path string  }
}

## keyvaluestore - FileKeyValueStoreOptions()

fabric-sdk-go / pkg / fab / keyvaluestore / filekeyvaluestore.go
// FileKeyValueStoreOptions allow overriding store defaults
type **FileKeyValueStoreOptions** struct {
    // Store path, mandatory
    Path string
    // Optional. If not provided, default key serializer is used.
    KeySerializer KeySerializer
    // Optional. If not provided, default Marshaller is used.
    Marshaller Marshaller
    // Optional. If not provided, default Unmarshaller is used.
    Unmarshaller Unmarshaller
}