framework_unified

目次 *[Table of contents]*

        - シーケンス [Sequence]
            - メッセージキューオープン/クローズ /message quene open/close
- 同期通信 /sync communication
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 同期通信 /sync communication
- 非同期通信 /asynchronous communication
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 非同期通信 /asynchronous communication
- レスポンスデータ送信 /send response data
    - 概要 [Overview]
    - シーケンス [Sequence]
        - レスポンスデータ送信 /send response data
- メッセージデータ受信 /receive message data
    - 概要 [Overview]
    - シーケンス [Sequence]
        - メッセージデータ受信 /receive message data
- ゼロコピー通信 /zero copy communication
    - 概要 [Overview]
    - シーケンス [Sequence]
        - ゼロコピー通信 /zero copy communication
- 非ブロッキングメッセージ受信 /none-blocked message receive
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 非ブロッキングメッセージ受信 /none-blocked message receive
- 受信メッセージ破棄 /delete received message
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 受信メッセージ破棄 /delete received message
- 子スレッド起動_ループなし /start sub thread(no loop)
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 子スレッド起動_ループなし /start sub thread(no loop)
- 子スレッド起動_ループあり(子スレッド間通信) /start sub thread_has loop(sub thread communication)
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 子スレッド起動_ループあり(子スレッド間通信)/start sub thread_has loop(sub thread communication)
- タイマー（コールバック） /timer(callback)
    - 概要 [Overview]
    - シーケンス [Sequence]
        - タイマー（コールバック） /timer(callback)
- タイマー（API） /Timer(API)
    - 概要 [Overview]
    - シーケンス [Sequence]
        - タイマー（API） /Timer(API)
- タイマー（NSTimerクラス） /timer(NSTimer class)
    - 概要 [Overview]
    - シーケンス [Sequence]
        - タイマー（NSTimerクラス） /timer(NSTimer class)
- RingBufferの読込み/書込み /read/write ringBuffer
    - 概要 [Overview]
    - シーケンス [Sequence]
        - RingBufferの読込み/書込み /read/write ringBuffer
- Configuration Fileの読込み/書込み /read/write configuration file
    - 概要 [Overview]
    - シーケンス [Sequence]
        - Configuration Fileの読込み/書込み /read/write configuration file
- 初期化 /Initialize
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 初期化 /Initialize
- ログ出力 /log output
    - 概要 [Overview]
    - シーケンス [Sequence]
        - ログ出力(_CWORD78_LOG) /log output(_CWORD78_LOG)
        - ログ出力(NsLogData) /log output(NsLogData)
        - ログ出力(NsLog) /log output(NsLog)
        - ログ出力(NsLogTime) /log output(NsLogTime)
        - ログ出力(NSLogPrintPerformanceLog) /log output(NSLogPrintPerformanceLog)
        - ログ出力(NsLog0) /log output(NsLog0)
- 強制クリア / force clear
    - 概要 [Overview]
    - シーケンス [Sequence]

        - 概要 [Overview]
        - シーケンス [Sequence]
            - SharedMemoryWriter
- 共有メモリオブジェクトのオープン状態を確認する(SharedMemoryWriter) /Check the open state of the shared memory object (SharedMemoryWriter).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemoryWriter
- 共有メモリオブジェクトをクローズする(SharedMemoryWriter) /Close shared memory object (SharedMemoryWriter).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemoryWriter
- 共有メモリオブジェクトにデータを書き込む /Write the data to shared memory object.
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemoryWriter
- 共有メモリバッファーをクリアする /Clear the shared memory buffer.
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemoryWriter
- Operation of Shared Memory /Operation of Shared Memory
    - 概要 [Overview]
    - シーケンス [Sequence]
        - DeferMessage /DeferMessage
- 共有メモリオブジェクトをオープンする(SharedMemory) / Open shared memory object(SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリオブジェクトのオープン状態を確認する(SharedMemory) / Check the open state of the shared memory object (SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリオブジェクトをクローズする(SharedMemory) / Close shared memory object (SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリオブジェクトからデータ読み込み(SharedMemory) / Read data from shared memory object (ShardMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリオブジェクトのデータをファイルへ書き出し(SharedMemory) / Write the data of the shared memory object to a file(SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリオブジェクトのデータサイズを取得(ShraedMemory) / Get the data size of the shared memory object (SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリオブジェクトのデータ読み出しポインタを初期化(SharedMemory) / Initializes the data read pointer of the shared memory object(SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリオブジェクトにデータを書き込む(SharedMemory) / Write the data to shared memory object (SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- 共有メモリバッファーをクリアする(SharedMemory) /Clear the shared memory buffer(SharedMemory).
    - 概要 [Overview]
    - シーケンス [Sequence]
        - SharedMemory
- メイン処理 /Main process
    - 概要 [Overview]
    - シーケンス [Sequence]
        - 参照先シーケンス [Referring sequence]
        - 参照元シーケンス [Referred sequence]
- シーン別電源の状態遷移 /State transition of Situation based power management
    - 概要 [Overview]

図表目次 *[Table of figures]*

framework_unified *[framework_unified]*

機能概要 *[Functional overview]*

表.ユニット概要

ユニット名*[Unit Name]*	コンポーネント名*[Component Name]*	ユニット概要*[Description]*	オーナーディレクトリ*[Owner Dir]*
framework_unified	NativeService	framework_unifiedは以下の機能を提供している。 *framework_unified provide functions as follows.* 1. HANDLE：インスタンスの内部構造を隠ぺいするために用いる型を提供。 2. Dispatcher：メッセージ受信とコールバック関数登録の仕組みを提供。 3. 1 対 1 のプロセス間通信：任意のプロセスと通信を行う仕組みを提供。 4. 1対多のプロセス間通信：多数のプロセスと通信を行う仕組みを提供。 5. Session：通信したいサービスとのセッション(通信経路)を確立する仕組みを提供。 6. Abnormal Monitor ：framework_unifiedにおける異常監視する機能を提供。 *1.HANDLER:provide variable type to hide the internal construction of instance* *2.Dispatcher:provide mechanism for receiving message and registering callback function* *3.one-to-one communication between process: provide mechanism for communicating with any process* *4.one-to-many communication between process: proved mechanism for communicating with many process*	framework_unified

		5.Session: provide mechanism for build session with communication target *6.Abnormal Monitor: provide mechanism of abnormality monitoring*	

機能詳細 *[Function detail]*

HANDLE

HANDLEはインスタンスを特定するための変数であり、インスタンスの内部構造を隠ぺいするために用いる型である。(C言語の実装でvoid * である。)

例えば、framework_unifiedが提供する_CWORD78_OpenService(pServiceName)は、指定されたサービスと通信するため通信リソースを作成し、 HANDLEを呼び出し元に返却する。この時、作成した通信リソース情報の構造の先頭アドレスをHANDLEとして返却する。

こうする事で使用者は、通信リソース情報の構造を意識する必要がなくなる。

HANDLE is a variable to specify instance and is also a variable type to hide the internal construction of instance.

For example: _CWORD78_OpenService(pServiceName) which is provided by framework_unified, is a IF to create communication resource for communicating with specified service, and return HANDLE to the caller.

At this case,start address of communication resource information will be returned as a HANDLE.

By this way, users need not to know the construction of communication resource.

Dispatcher

概要*[overview]*

Dispatcherはframework_unifiedの根幹をなすモジュールである。

Dispatcherは、外部からメッセージを受信し、そのメッセージに応じて、登録されたサービスの関数を呼び出す仕組みを提供する。

よって、サービスが他のモジュールからメッセージを受けるためには、Dispatcherを使用する必要がある。また、Dispatcherは、MainLoopを内包する。

Dispatcher is a fundamental module of framework_unified.

This module receives message from outside and call function of register service in response of message.

So if a service wants to receive message from other module,Dispatcher is in need.By the way Dispatcher includes MainLoop

MainLoop とは、外部からのイベント受信待ちを行うためのLoop関数である。使用方法として、Dispatcherに内包するMainLoopを使用する方法と、 サービスでMainLoopを実装する方法があるが、詳細は後述する。

Dispatcherが受信したメッセージを取得するためには、Callback関数を_CWORD78_AttachCallbackToDispatcherなどの関数を用いてframework_unifiedに設定する必要がある。

MainLoop is a loop function to control event message from outside.*User can use MainLoop function that is included in Dispatcher or implement MainLoop by service.*

To get message received by Dispatcher,user should use callback function of _CWORD78_AttachCallbackToDispatcher etc. and set it to framework_unified.

サービスが_CWORD78_Dispatcher()を呼び出すことでDispatcherは生成される。

framework_unifiedはDispatcherの作成要求を受けると、Dispatcherインスタンスの作成、 _CWORD78_OnInitializationのCallbackを行い、その後、メッセージ受信待ちになる。

サービスは作成されたDispatcherインスタンスをアプリケーションHANDLE(以降hApp)として、_CWORD78_OnInitialization の引数で取得する事ができる。

Call _CWORD78_Dispatcher() to create Dispatcher.

When framework_unified gets the request to create Dispatcher, Dispatcher instance will be created and conduct callback of _CWORD78_OnInitialization.After all these process,wait for message.

Service will treat Dispatcher instance as application HANDLER(hApp), and this instance can be get by calling _CWORD78_OnInitialization.

framework_unifiedが提供するAPIの多くは、hAppの指定が必要である。framework_unifiedはhAppにより、どのコンテキストからAPICALLされたかを判断する。

Dispatcherは１スレッドに対して1つのインスタンスを持つ。サービスは_CWORD78_CreateChildThread()を用いてサブスレッドを作成する事により、 Dispatcherインスタンスを追加する事が出来る。

LinuxPFではプロセス間のIFはframework_unified(Dispatcher同士の通信)を用いて通信を行う方針となっている。

よって、framework_unifiedを使用した他のプロセスと通信を行うスレッドは、Dispatcherを使用する必要があるため、サブスレッドが他プロセスと通信を行う場合は_CWORD78_CreateChildThread()を用いてスレッド起動を行い、Dispatcherインスタンスを持つ必要がある。

There are mane APIs in framework_unified unit.When using these APIs,user should specify hApp and framework_unified will judge context by hApp.

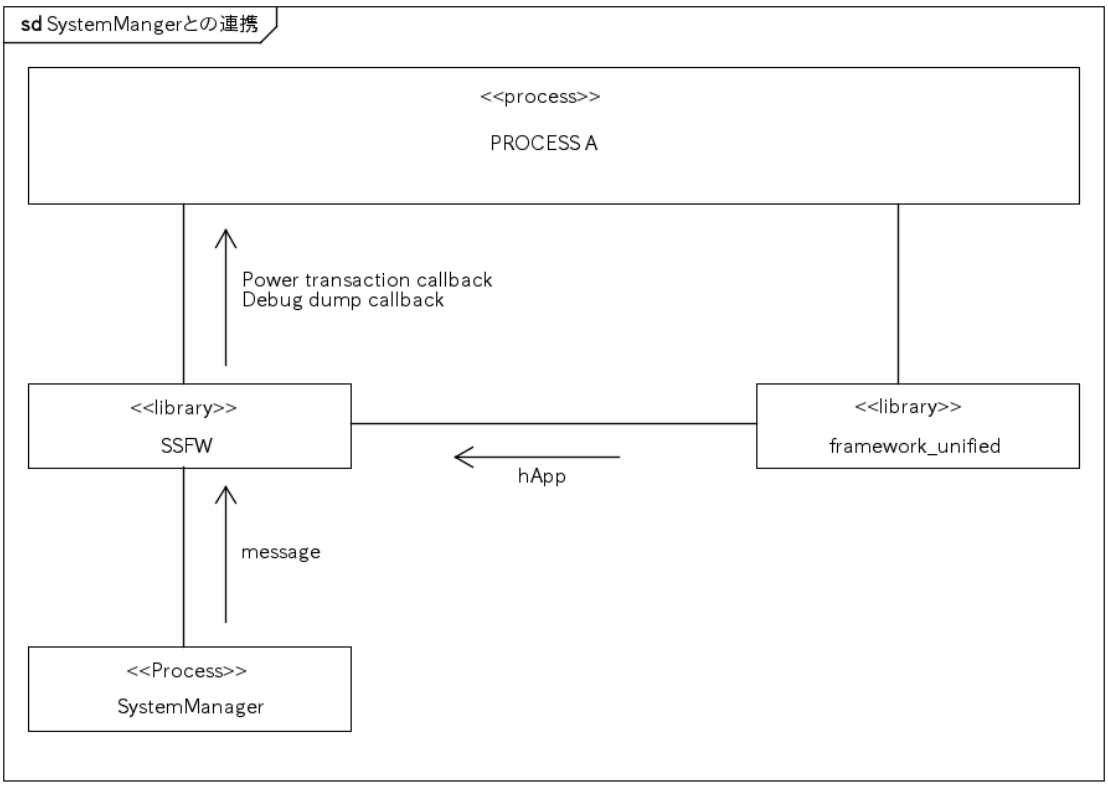
Dispatcher only has one instance for one thread.Service can create sub thread by calling _CWORD78_CreateChildThread().By this way Dispathcer instance can be added.

User should use framework_unified(communication between DIspatcher) to communicate between process in Linux platform.

So,if user wants to communicate with other process which uses framework_unified, Dispatcher is in need.

when Sub thread communicates with other process, start sub thread by calling _CWORD78_CreateChildThread() and Dispatcher instance is in need.

SystemMangerとの連携*[work with SystemManager]*



framework_unifiedはメインスレッドのDispatcherインスタンスが作成された際に、system_manager(SM)とのセッションを自動的に確立する。

その際SMとのセッション情報は、SSFW内に隠ぺいして管理する。SMはこのセッションを用いてSSFWを介し、シーン別電源の状態遷移やデバッグダンプ要求など のCallbackをProcess Aに対して行う。(_CWORD78_OnStartや_CWORD78_OnDebugDumpのCallbackが該当する)

SSFWとは、SMなどのSystemServiceと通信するためのライブラリであり、SMから起動させる常駐プロセスはリンクする必要がある。

なお、SMとのセッションを自動的に確立するのは、常駐サービスのみであり、非常駐サービスに関しては同等の機能を別の方法で提供する。

When framework_unified creates main thread of Dispatcher instance,the session with system_manager(SM) will be build automatically.

At this time,session information is hided and managed in SSFW. SM uses this session and through SSFW Process A will handle callback of state transition of Situation based power management and debug dump request . (callback of _CWORD78_OnStart and _CWORD78_OnDebugDump is applicable)

SSFW is a library to communicate SystemService with SM etc. Resident process which is started by SM should link this library.

Only resident service can build the session with SM automatically.none-resident service can use other method to implement this function.

SystemManagerとの接続処理を実行する関数_CWORD78_SSFrameworkInterfaceを提供する。

アプリケーションが非常駐サービスの場合は、以下名称の関数を空関数で定義すること。

E_CWORD78_Status _CWORD78_SSFrameworkInterface(HANDLE hApp))。

また、Discpatcher作成時のパラメータによっては、SMとのセッションを確立しない事もできる。(この場合_CWORD78_OnStartは呼ばれない)

User can connect with SystemManager by calling _CWORD78_SSFrameworkInterface().

If an application is not a resident service,the function as follows should define as an empty function

E_CWORD78_Status _CWORD78_SSFrameworkInterface(HANDLE hApp))。

Based on the parameter when creating Dispatcher,the session with SM is possible to not be build.(At this case,do not all _CWORD78_OnStart)

シーン別電源

Situation based power management

SystemManagerの電源状態遷移を参照

Refer to Power State Transition of SystemManager

Dispatcherの使用方法*[method to use Dispatcher]*

1.DispatcherのMain Loopを使用する方法(メインスレッド)*/method to use Main Loop of Dispacther*

Dispatcher内のMainLoopを使用する方法であり、_CWORD78_Dispatcherまたは_CWORD78_DispatcherWithArgumentsによって使用する事が出来る。

This is a way to use Main Loop of Dispatcher,also user can call _CWORD78_Dispatcher or _CWORD78_DispatcherWithArguments to use it.

2.MainLoopをService側で実装する方法 */implement way of MainLoop by Service*

Main Loopをサービスで実装する方法であり、Qtなどの他のフレームワークとframework_unifiedを共存させたい場合などに使用する事を想定している。

Implement way of Main Loop by Service,and also can be used if user wants to coexist other framework(Qt etc.) and framework_unified

_CWORD78_CreateDispatcherWithoutLoop() によって、Dispatcherインスタンスを生成する事ができ、
_CWORD78_GetDispatcherFD()により取得したfdをpollやselect などを用いて監視する事により、メッセージ受信タイミングを知る事ができる。メッセージ受信したタイミングで、
_CWORD78_DispatchProcessWithoutLoopを呼び出すことにより、Dispatcherにて受信処理が行われ、登録された Callback関数が呼び出される。

Dispatcher instance can be created by calling _CWORD78_CreateDispatcherWithoutLoop().Poll/select fd which is get by _CWORD78_GetDispatcherFD() and message receive timing will be known.

At message receive timing,call _CWORD78_DispatchProcessWithoutLoop(). Dispatcher will handle massage receiving process and registered callback function will be called.

3.Dispatcherの制限事項/Dispatcher limitations

hAppはスレッド間で共有する事はできない。

1スレッドに対して複数のDispatcherインスタンスを生成する事はできない。

hApp can not be shared between thread.

One thread can not create many Dispatcher instance.

1 対 1 のプロセス間通信 */one-to-one process communication*

1 対 1 のプロセス通信は、サービス名、コマンド、セッションの3要素を用いて行う。

例えば、メッセージ受信プロセスでは、_CWORD78_AttachCallbackToDispatcherで、引数に「送信元サービス名」、「コマンド」、「セッション」、関数ポインタを設定する事でメッセージ受信時に呼び出されるCallback関数を設定する。この場合、「送信元サービス名」、「コマンド」、 「セッション」がすべて一致したメッセージが届いた時に、Callback関数が呼び出される。

また、メッセージ送信側では_CWORD78_SendMsgに「送信先サービス名」、「コマンド」、「セッション」を指定してメッセージを送信する。

Execute one-to-one process communication by service name,command and session.

For example,message receive process calls _CWORD78_AttachCallbackToDispatcher() and sets parameter of 「sender service name」,「command」,「session」,function pointer.

By this way callback function which is called will be set when receiving message.

At this case,the callback function will be called only if the 「sender service name」,「command」,「session」 matched.

Message send side sets the of 「receiver service name」,「command」,「session」 of _CWORD78_SendMsg

コマンド */command*

コマンドは32bitで指定する事ができる。

ただし、後述するデフォルトセッション(後述)を使用して通信する場合、使用できるコマンドは
PROTOCOL__CWORD78__BASE_CMD<=コマンド＜PROTOCOL__CWORD78__MAX_CMDの範囲である。

コマンドは、「セッション」と「サービス」の組み合わせ毎にユニークである必要がある。(システム内でユニークである必要はない)

Command can specify 32bit.

But when use default session to communicate,the threshold of command is PROTOCOL__CWORD78__BASE_CMD<=command＜PROTOCOL__CWORD78__MAX_CMD.

Command of the pair of session and service should be unique.(No need to be unique inside system)

セッション */session*

セッションとは、プロセス間の接続を行う仕組みである。

クライアントサーバモデルにおいて、クライアントがセッションの接続要求をサーバに対して行い、サーバが要求に応じる事により、セッションが確立する。

クライアントは、確立されたセッションを使用して、サーバに機能実行要求を発行する事ができる。

サーバはクライアントに関する情報を、セッション毎に関連付けて記憶する事で、クライアント毎のコンテキストを管理する事が出来る。

Session provides function of process connect.

Server builds the session in response to session connect request of client based on client server model.

Client can use the session to send function request to server.

Server saves client information combined with session so that it can manage context of every client.

セッションID /session ID

セッションIDはセッションに対して割り当てられるIDであり、サーバのDispatcherインスタンス内でユニークである。

(IDは_CWORD78_CreateSession()によって、自動生成される)

Session ID is the assigned ID of session and it's unique in Dispatcher instance of server.

(ID is automatically generated by _CWORD78_CreateSession()).

デフォルトセッション /default session

セッションID=0はデフォルトセッションと呼ばれ特別な意味を持つ。

デフォルトセッションはDispatcherの作成と同時に自動的に生成される。すなわち、すべてのDispatcherはセッションID=0のセッションを持つことが保障されている。

デフォルトセッションは、セッションのOPEN/CLOSEやnotification_persistent_serviceとの通信などのframework_unified内部の通信に使用される。

サーバプロセスは明示的にセッションを構築せず、デフォルトセッションを用いてIFを提供(セッションレスでのIF提供)する事が出来るが、この場合、クライアント側のライフサイクルを知る事ができない。

Session ID = 0 is a default session and has a different meaning.

Default session is created automaticall when creating Dispatcher.That means, all the Dispatcher can have a session whose session ID=0.

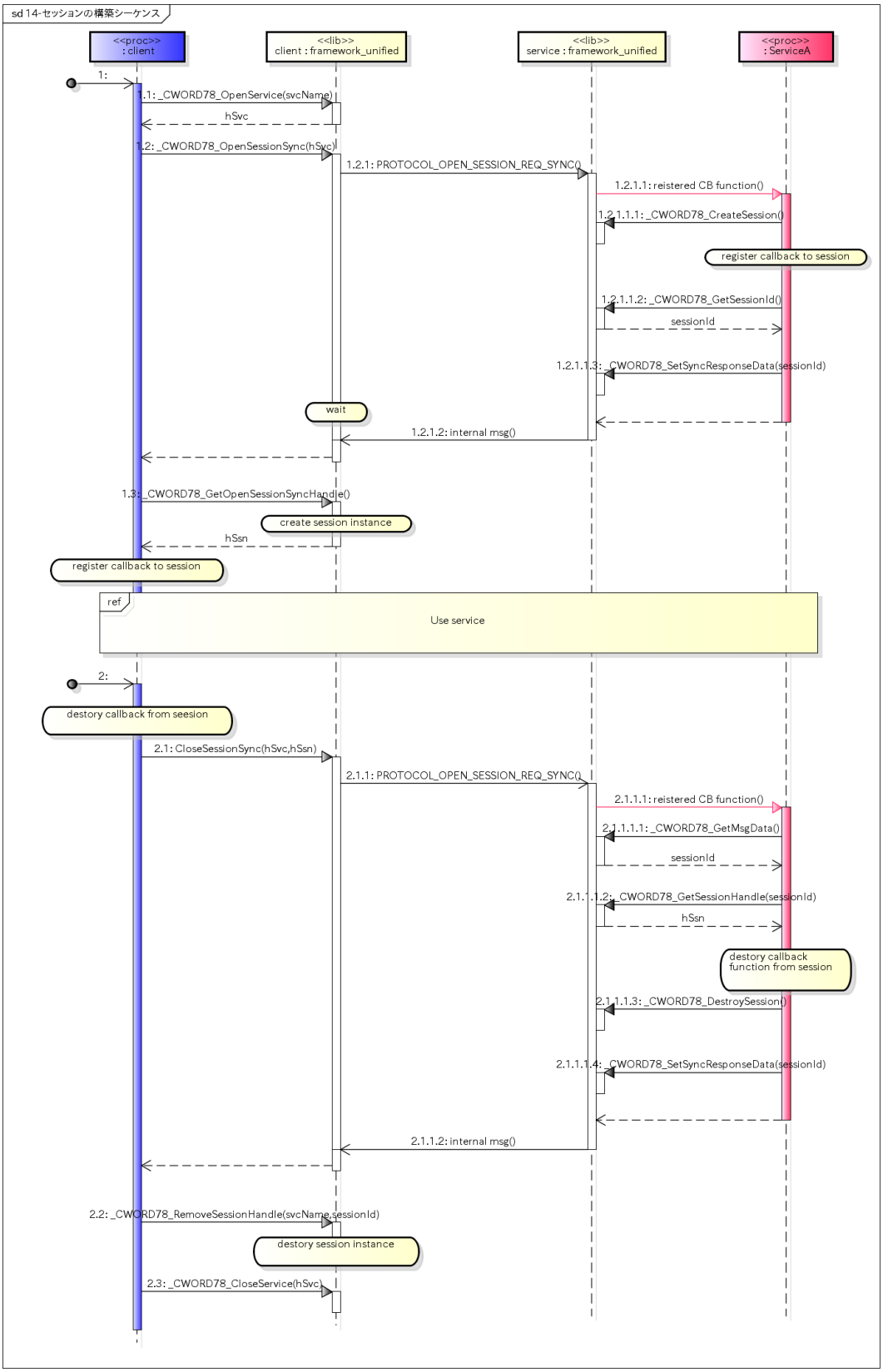
Default session is used at the case of session open/close,communication with notification_persistent_service etc.which are communication inside framework_unified

セッションの構築シーケンス */ session build sequence*

セッションの構築シーケンスを以下に示す。

なお、本シーケンスは同期APIを使用した例になっている。

Session build sequence is as follows.This sequence is an example when using sync API.



通信の種類 */communication type*

１対１のプロセス間通信をする方法は大きく非同期メッセージと同期メッセージの2種類の通信方法がある。

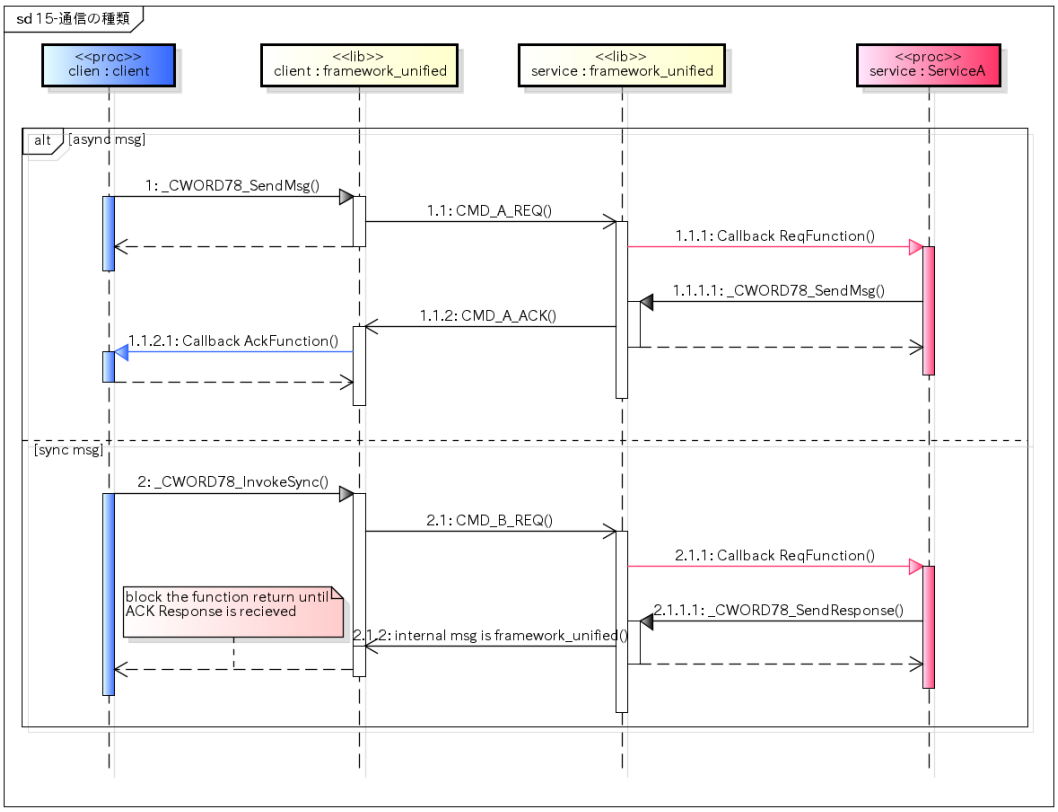
非同期メッセージと同期メッセージを適切に使い分ける事が、設計品質を向上させる上で重要である。

Sync message and asynchronous message are two method of one-to-one process communication.

Using Sync message and asynchronous message properly can upgrade design quality.

以下にクライアントとサービス間での通信例を記載する。

Communication example between client and service is as follows.



非同期メッセージ

Async Massage

非同期メッセージは、クライアントから_CWORD78_SendMsg(serviceName,CMD,prm,session)によって、指定したサービスにコマンドとパラメータを送信すると、すぐに関数がRETURNする。サービスからクライアントに応答を送る際も、同様に_CWORD78_SendMsgを用いて応答を送る事ができる。

sync Massage: Client uses function _CWORD78_SendMsg(serviceName,CMD,prm,session) to send command and parameter to the designated services, then return function immediately.

Service can use the same function(_CWORD78_SendMsg) to response client's request too.

- メリット：応答性能が良い

- merit: response performance is good

REQ～ACKの間で別メッセージを処理できる。(応答速度が速い)

Other message can be processed in the middle of REQ～ACK.(response speed is quick)

- デメリット：保守性が低い

- demerit: It's not good for maintenance.

送信処理と受信処理が分断され可読性が落ちる。

送信～受信の間にメッセージ受信するケースが増加し複雑化しやすい

Sending and reception process is divided into parts and readability degrade.

It easy to make reception process more complicated.e

同期メッセージ

Sync Massage

同期メッセージは、クライアントから_CWORD78_InvokeSync(serviceName,CMD,prm,session)を発行すると、サービス側でCMDを受信し、_CWORD78_SendResponce()によりクライアントに応答が返されるまで関数がブロックされる。

Sync Massage:Client can use _CWORD78_InvokeSync(serviceName,CMD,prm,session) to send CMD to the service.

Service use the _CWORD78_SendResponce() to response client's request, and the client is blocked before _CWORD78_SendResponce() is called.

- メリット：保守性が高い

- merit: maintainability　is good.

処理分断がないため、可読性が良い。

要求中に別メッセージが割り込まないため、管理する状態を少なく出来る。

Readability is good, because it is no necessary to divide processing into parts.

Management status is less because other message can not interrupt in the middle of one request processing.

- デメリット：応答性能が悪い

- demerit: response performance is bad

ACKが帰るまで別のメッセージが受ける事ができない。

Other message can not be accepted until response is completed.

※注意点：

note:

長時間の処理には適応するべきではない。

優先度がClient < Serviceでないと、優先度逆転が起こる。

It should not be a long time processing.

Priority will be changed if Client's priority is greater than service.

_CWORD78__ANY_SOURCE

受信Callbackの設定では、サービス名にワイルドカードとなる_CWORD78__ANY_SOURCEを指定する事ができる。サービス名に _CWORD78__ANY_SOURCEを指定した場合、セッションとコマンドさえ合致すれば、送信元サービスが何であっても、受信Callbackが呼び出される。

セッションを通じてクライアンと通信するサービスが送信元サービスに_CWORD78__ANY_SOURCEを指定する必要があるのは、セッションOPENコマンドの PROTOCOL_OPEN_SESSION_REQまたは、PROTOCOL_OPEN_SESSION_REQ_SYNC(同期APIの場合)のみである。

セッション確立後は、P2Pの通信が可能となるため、送信元が特定可能であるためである。

セッションレスのサービスは、デフォルトセッションを使用して、送信元サービスに_CWORD78__ANY_SOURCEを指定する事で、どのクライアントからでもメッセージを発行する事が可能となる。

一般に、_CWORD78__ANY_SOURCEの使用を限定的にし、セッションを使用したサービス間の通信を行う方が、通信路が集約され保守性および堅牢性が向上する。

It can use _CWORD78__ANY_SOURCE for a service name when reception Callback-information is setting.
when _CWORD78__ANY_SOURCE is used for service name,

no matter which service is, reception Callback will be called if session and command is matching.

It's necessary to designate _CWORD78__ANY_SOURCE when session is used and session open commend is

PROTOCOL_OPEN_SESSION_REQ and PROTOCOL_OPEN_SESSION_REQ_SYNC.

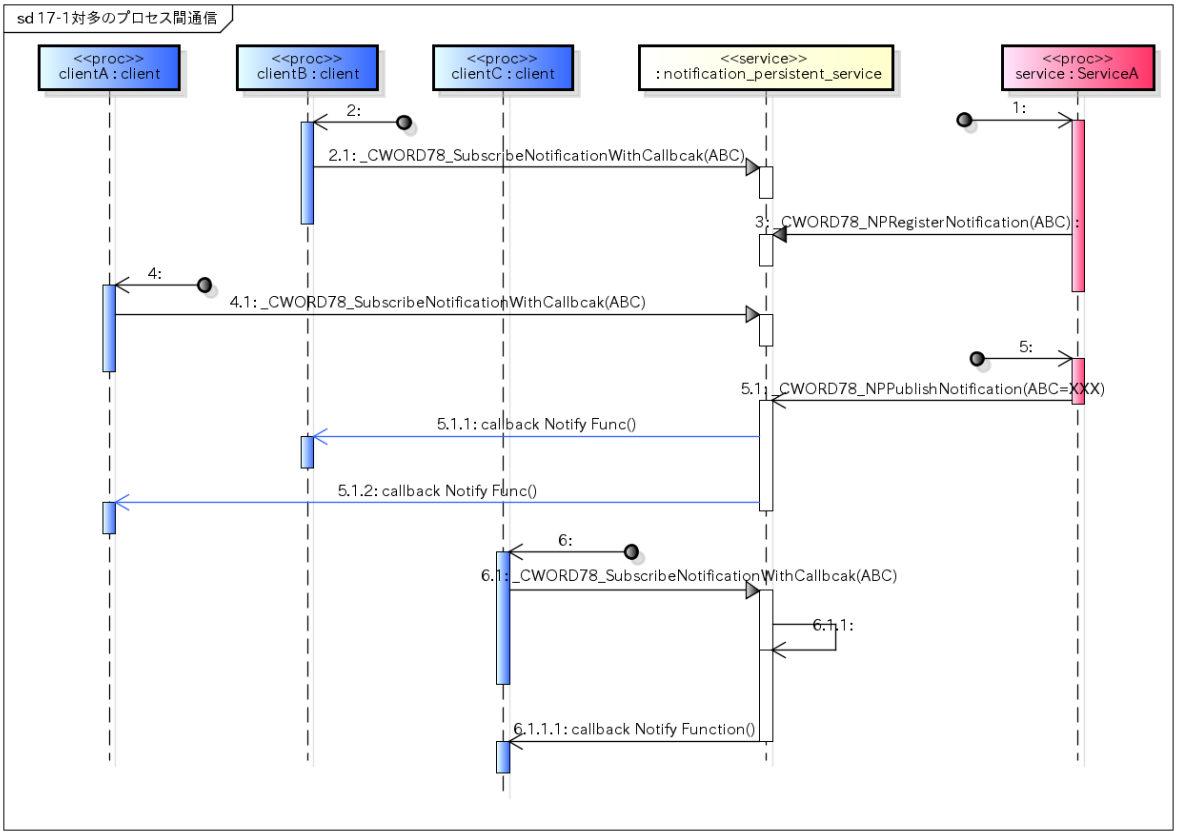
After session is established, it is possible to do P2P's communication. At this situation, Sender could be specific.

SessionRes's service:using default session and _CWORD78__ANY_SOURCE, can send message to any client.

Generally,_CWORD78__ANY_SOURCE's use is limited which can be used in the communication of session.

It will make communication intensive, also can improve maintainability and reliability.

1対多のプロセス間通信



1対多の通信に関しては、notification_persistent_serviceがその仕組みを提供する。

1対多の通信は、１対１通信に比べ通信コストが高いため、1対1通信で設計可能なものには採用するべきではない。

上記はServiceAがブロードキャストするメッセージをClientA~Cが受信する例である。

ClientAは、NORTIFICATION名を用いて監視する通知を特定する。(例ではABCがそれにあたる)

NORTIFICATION名は文字列であり、”送信元サービス名/メッセージ名”とし、(この例ではServiceA/ABCとなる)メッセージ名を送信元サービスでユニークに設計する事により、システム全体でそのユニーク性を保証する。

notification_persistent_service provide function of one to many communication.

One to many communication will take more cost, so it should not be used when one to one's communication can be used.

Using ServiceA broadcast to ClientA~C for an example.

ClientA use NORTIFICATION to specific monitor notification.

NORTIFICATION is string, message name is designed unique, so it can assurance the system's unique.

ブロードキャストするSeriveAは、_CWORD78_NPRegisterNotification()により、NORTIFICATION名をnotification_persistent_serviceに登録し、_CWORD78_NPPublishNotification()により、全体へブロードキャストする。

SeriveA uses _CWORD78_NPRegisterNotification() to register the NORTIFICATION name into notification_persistent_service,

and uses _CWORD78_NPPublishNotification() to broadcast to entirety.

ClientA~Cは、_CWORD78_SubscribeNotificationWithCallback()にNORTIFICATION名を設定する事で監視するNORTIFICATIONを指定する。

なお、Serviceが_CWORD78_NPRegisterNotification()を行うより前に、より前Clientが_CWORD78_SubscribeNotificationWithCallback()した場合でも、Clientはメッセージを受信する事ができる。

なお、本例は、_CWORD78_NPRegisterNotification()で、e_CWORD78_PersistedStateVarを指定した場合の例であり、e_CWORD78_NotificationVarを指定する事で変化トリガのみを受信する事も出来る。

ClientA~C designate monitor notification through setting NORTIFICATION name into _CWORD78_SubscribeNotificationWithCallback().

Yet, if Client called _CWORD78_SubscribeNotificationWithCallback() before service call _CWORD78_NPRegisterNotification(), client can receive the message.

Yet, in this example, Using _CWORD78_NPRegisterNotification() with e_CWORD78_PersistedStateVar designated.

If e_CWORD78_NotificationVar is designated, it only need to change trigger to receive message.

セッションの使用例

セッションのライフサイクルをどのように設計するは、提供するサービスに依存するが、参考に動画再生機能を提供する架空のサービスPlayServiceを例にセッションの使用例を下図を使用して説明する。

この例では、PlayServiceは動画の再生機能を持っており、OPENされたセッションに対して、１つのデコーダ(DEC)リソースを割り当てる仕様とする。

MainMoniterHMIはhSsn1通じてPlayServiceと通信しDEC1を使用して再生する。SubMonitorHMIはhSsn2を通じてPlayServiceと通信しDEC2を使用して再生している。

このように、セッションをOPENしている間にのみ、関連づけられたDECリソースを占有し使用できるようにする事で、DECリソースの状態をクライアントに紐づけ管理する事が出来る。

また、PlayServiceがDEC1とDEC2を同時再生できる場合、セッション毎に独立したDECリソースを割り当て制御する事で同時再生を行う事もできる。

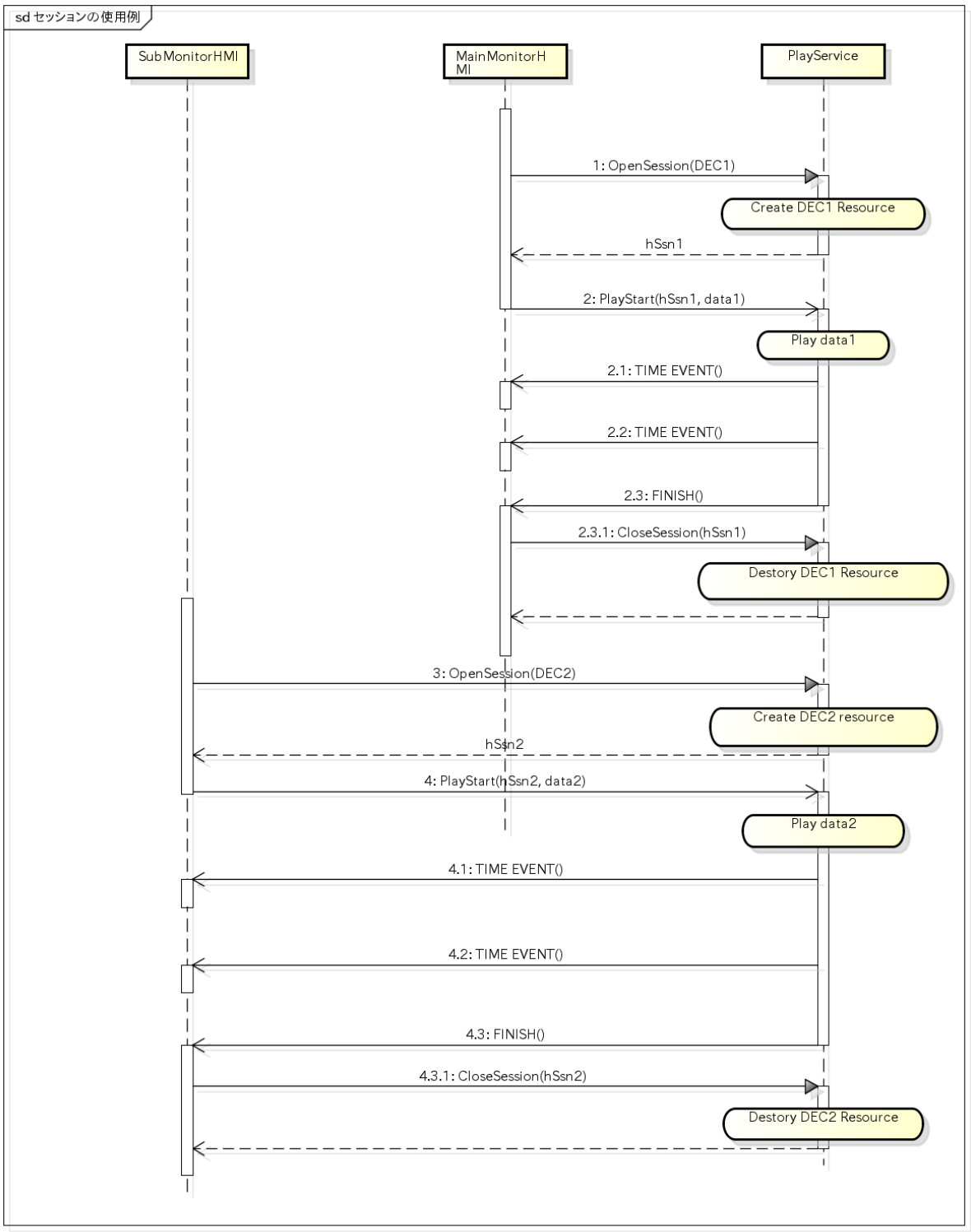
Using animation play function for an example to explain how to design session's lifecycle and dependence sevices.

In this example,for one opend sesstion, playService which has animation play function using one DEC resource for a design.

MainMoniterHMI using hSsn1 to communicaton with PlayService, and using DEC1 to play, and then using DEC2 to play.

Like this way, only in the time of session opening, can exclusive relation DEC resource and can use it , and can manage DEC resource status.

And if playService can paly DEC1 and DEC2 at same time because every session is independence.



Abnormal Monitor

framework_unifiedにおける異常監視が提供する機能は以下。

The functions of abnormal monitor of framework_unified are as follows.

1.Session確立時の相互の異常監視

サーバー/クライアントが相手の異常終了を検出した場合、SessionをCloseしリソースを開放する。

framework_unifiedからユーザロジックに異常終了の検出通知用のCallback登録I/Fを提供する。

1.mutual abnormal monitor when session is build

If server founds abnormal end of client or client founds abnormal end of server,close session and free resource.

framework_unified provides callback register I/F for user logic to notify abnormal end message when abnormal end occurs.

2.同期API(_CWORD78_InvokeSync)の呼び出し中の異常監視

MessageCenterでサービスの異常終了を検出しエラー返却する。

2.abnormal monitor when calling sync API(_CWORD78_InvokeSync)

MessageCenter returns the error of abnormal end.

3.マルチキャストサービスの異常監視

登録クライアントが異常終了した場合、登録エントリを破棄する。

3.abnormal monitor of multi cast service

When abnormal end of registered client occurs,delete register entry.

<内部実装検討> */<internal implement consideration>*

Session確立時の異常監視、および同期API実行時の異常監視は、UNIXドメインのSTREAMソケットをプロセス間で張る。

STREAMソケットの仕様により、片方のプロセスが終了すると、相手のディスクリプタがreadyになる事を利用する。

Abnormal monitor when session is build ,and execute sync API, stretch the stream socket of UNIX domain among processes.

According to stream socket specification,when process of one side is end,the descriptor of other side is ready and also can be used

・Session確立時のシーケンス

1. サービス側でサービス名に紐づく待受ソケットをOpenしlisten

2. Session確立処理でクライアント側から待受ソケットにconnect

3. サービス側は待受ソケットがreadyになった事をepollで検出しaccept。接続ソケットはResourceControlerへ登録

4. クライアント側も接続ソケットをResourceControlerへ登録

・sequence of session build

1. service opens/listens socket that used to receive data,which is specified by service name

2. client connect to the receive socket

3. epoll checks whether receive socket of service is ready or not,and if it's ready,set accept 。connect socket will register to ResourceControler

4. register connect socket of client to ResourceControler

・サービス異常終了時のシーケンス

1. サービス異常終了によりクライアント側接続ソケットがready

2. epollからResourceControlerIDを取得、ResourceContorlerからサービスを特定しSessionをClose

・sequence of sevice end abnormality

1. service end abnormality because client access socket is ready.

2. After get ResourceControlerID from epoll, designation service is closed.

・ResourceControler

C言語 I/Fである事。

参照カウンタを持ち、参照0で実行するCallbackを登録できる事。

_CWORD78_GetResource:単純なGetと、_CWORD78_AcquireResouce:参照カウンタをインクリメントするGetを別に

_CWORD78_ReleaseResource:参照カウントデクリメント、_CWORD78_RegistResouce.
_CWORD78_UnregistResouceがある。

・*ResourceControler*

I/F of C language。

This module has a reference counter and can register callback that execute with reference counter 0。

_CWORD78_GetResource is a simple get function and different with other get function,such as _CWORD78_AcquireResouce(increment reference counter) .

Also there are functions such as _CWORD78_ReleaseResource(decrement refernce counter), _CWORD78_RegistResouce,_CWORD78_UnregistResouce function.

ソフトウェア構成図 *[software block]*

framework_unifiedのブロック図を以下に示す。

Block diagram of framework_unified is as follows.

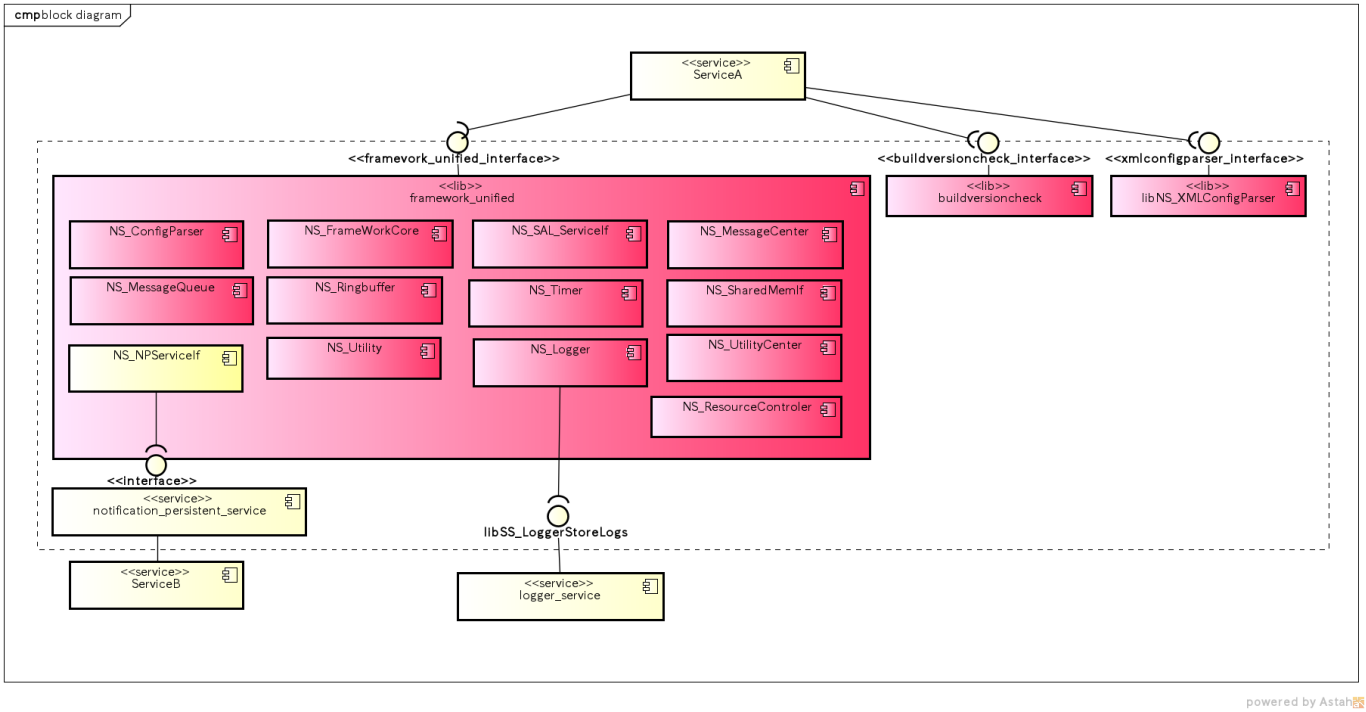


図. ソフトウェア構成図

ユースケースとAPI一覧 *[Use case and API lists]*

外部要因 ユースケース一覧 *[outside factor use-case list]*

表. 外部要因ユースケース一覧

ユースケース番号*[Use-case number]*	機能カテゴリ*[function category]*	ユースケース名*[use-case name]*	ユースケースを実現するAPI *[API to realize use-case]*	備考*[notes]*
framework_unified_Dispatcher_001	Dispatcher	サービス起動	_CWORD78_CreateDispatcherWithoutLoop	

		Start service	_CWORD78_DestroyDispatcherWithoutLoop _CWORD78_Dispatcher _CWORD78_DispatcherWithArguments _CWORD78_DispatchProcessWithoutLoop _CWORD78_GetDispatcherFD _CWORD78_OnInitialization _CWORD78_SimpleDispatcherSendSystemErrorToSystemManager	
framework_unified_Dispatcher_002		Callback/イベント購読登録 *Callback/event subscription*	_CWORD78_AttachCallbacksToDispatcher _CWORD78_AttachCallbacksToDispatcherWithFd _CWORD78_AttachCallbackToDispatcher _CWORD78_AttachCallbackToDispatcherWithFd _CWORD78_AttachLostSessionCallbackToDispatcher _CWORD78_AttachParentCallbacksToDispatcher _CWORD78_DestroySession _CWORD78_DetachCallbacksFromDispatcher _CWORD78_GetLostSessionData _CWORD78_SubscribeNotificationsWithCallback _CWORD78_SubscribeNotificationWithCallback	
framework_unified_Dispatcher_003		Callback/イベント購読解除 *cancel callback/event subscription*	_CWORD78_DetachCallbackFromDispatcher _CWORD78_DetachCallbackFromDispatcherWithFd _CWORD78_DetachCallbacksFromDispatcher _CWORD78_DetachCallbacksFromDispatcherWithFd _CWORD78_DetachParentCallbacksFromDispatcher _CWORD78_DetachServiceFromDispatcher _CWORD78_UnsubscribeNotificationsWithCallback _CWORD78_UnsubscribeNotificationWithCallback	
framework_unified_Dispatcher_004		サービス利用状態 *service use status*	_CWORD78_GetSelfAvailability _CWORD78_GetServiceNameOnServiceAvailabilityNotification _CWORD78_IsServiceAvailable _CWORD78_PublishServiceAvailability _CWORD78_RegisterServiceAvailabilityNotification _CWORD78_SubscribeNotificationWithCallback _CWORD78_UnRegisterServiceAvailabilityNotification	
framework_unified_Dispatcher_005		アプリケーションデータ設定/取得 *application data set/get*	_CWORD78_GetAppData _CWORD78_GetAppName _CWORD78_GetThreadSpecificData _CWORD78_IsStateMachineApp _CWORD78_SetAppData _CWORD78_SetMandatoryServiceInfo _CWORD78_SetThreadSpecificData	
framework_unified_Dispatcher_006		DeferMessage *DeferMessage*	_CWORD78_DeferMessage _CWORD78_IsDeferQueueEmpty _CWORD78_RetrieveDeferMessage	
framework_unified_Session_001	Session	セッションオープン/クローズ *session open/close*	_CWORD78_CloseSession _CWORD78_CloseSessionSync _CWORD78_CreateSession _CWORD78_DestroySession _CWORD78_GetCurrentSessionHandle _CWORD78_GetMsgDataOfSize _CWORD78_GetMsgSessionId _CWORD78_GetOpenSessionHandle _CWORD78_GetOpenSessionSyncHandle _CWORD78_GetSessionHandle	

			_CWORD78_GetSessionId _CWORD78_GetSessionName _CWORD78_OpenService _CWORD78_OpenSession _CWORD78_OpenSessionSync _CWORD78_RemoveSessionHandle _CWORD78_SendMsg _CWORD78_SetSessionHandle _CWORD78_SetSyncResponseData	
framework_unified_Session_002		セッションイベント通知 *session event notification*	_CWORD78_AttachCallbackToDispatcher _CWORD78_DefinePrivateStateEvents _CWORD78_DefinePublicStateEvents _CWORD78_DetachCallbackFromDispatcher _CWORD78_PublishEvent _CWORD78_PublishPublicEvent _CWORD78_RegisterEvent _CWORD78_SubscribeToSessionEventWithCallback _CWORD78_UnRegisterEvent _CWORD78_UnSubscribeSessionEventWithCallback	
framework_unified_Message_001	Message	メッセージキューオープン/クローズ *message queue open/close*	McClose McOpenReceiver McOpenReceiverNotBlocked McOpenSender McOpenSyncSender _CWORD78_McClose _CWORD78_McOpenSender	
framework_unified_Message_002		同期通信 *sync communication*	McClose McCreateInvokerName McInvokeSync McOpenSyncReceiver McSendSyncResponse _CWORD78_InvokeSync	
framework_unified_Message_003		非同期通信 *asynchronous communication*	McClose McOpenSender McSend _CWORD78_McClose _CWORD78_McOpenSender _CWORD78_SendMsg _CWORD78_SendResponse	
framework_unified_Message_004		レスポンスデータ送信 *send response data*	McClose McOpenReceiver _CWORD78_AttachCallbackToDispatcher _CWORD78_InvokeSync _CWORD78_SendMsg _CWORD78_SendResponse _CWORD78_SetSyncResponseData	
framework_unified_Message_005		メッセージデータ受信 *receive message data*	McOpenReceiver _CWORD78_AttachCallbackToDispatcher McGetQueueFD McReceive _CWORD78_GetMsgProtocol _CWORD78_GetMsgSrc _CWORD78_GetMsgLength _CWORD78_GetDataPointer McGetSysInfoData _CWORD78_GetLastNotification _CWORD78_GetMsgDataOfSize _CWORD78_ClearMsgData	
framework_unified_Message_006		ゼロコピー通信 *zero copy communication*	McClose McGetDataOfSize McGetLength McOpenReceiver McReceive McZcGetBuf McZcOpenSender McZcSend McZcSetParam	
framework_unified_Message_007		非ブロッキングメッセージ受信 *none-blocked message receive*	McClose McOpenReceiverNotBlocked McOpenSender McReceive McSend	
framework_unified_Message_008		受信メッセージ破棄 *delete received message*	McClose McFlushReceiver McOpenReceiverNotBlocked	

			McOpenSender McSend	
framework_unified_Thread_001	Thread	子スレッド起動_ループなし *start sub thread(no loop)*	C_CWORD78_ThreadPriorities::GetPriority _CWORD78_AttachCallbacksToDispatcher _CWORD78_CreateDispatcherChild _CWORD78_DispatchProcessWithoutLoop _CWORD78_GetDispatcherFD	
framework_unified_Thread_002		子スレッド起動_ループあり(子スレッド間通信) *start sub thread_has loop(sub thread communication)*	_CWORD78_CreateChildThread _CWORD78_DestroyChildThread _CWORD78_SendChild _CWORD78_SendParent _CWORD78_StartChildThread _CWORD78_StopChildThread	
framework_unified_Timer_001	Timer	タイマー（コールバック） *timer(callback)*	_CWORD78_AttachTimerCallback _CWORD78_DetachTimerCallback	
framework_unified_Timer_002		タイマー（API） *timer(API)*	MSToNS NS_TimerCreate NS_TimerDebugOn NS_TimerDelete NS_TimerGetTime NS_TimerSetTime _CWORD78_AttachCallbackToDispatcher RemainderMs RemainderMs WholeSeconds	
framework_unified_Timer_003		タイマー（NSTimerクラス） *timer(NSTimer class)*	NSTimer::~NSTimer NSTimer::IsRunning NSTimer::NSTimer NSTimer::SetNotifyMethod NSTimer::SetRepeatTimer NSTimer::SetTime NSTimer::Start NSTimer::Start NSTimer::Stop _CWORD78_AttachCallbackToDispatcher	
framework_unified_RingBuffer_001	RingBuffer	RingBufferの読込み/書込み *read/write ringBuffer*	CNSRingBuffer::CNSRingBuffer CNSRingBuffer::~CNSRingBuffer CNSRingBuffer::ClearBuf CNSRingBuffer::Close CNSRingBuffer::CNSRingBuffer CNSRingBuffer::DumpToFile CNSRingBuffer::Open CNSRingBuffer::Read CNSRingBuffer::Write _CWORD78_SendMsg	
framework_unified_Configuration_001	Configuration	Configuration Fileの読込み/書込み *read/write configuration file*	CNSConfigParser::~CNSConfigParser CNSConfigParser::CNSConfigParser CNSConfigParser::Parse CNSConfigReader::GetBool CNSConfigReader::GetDouble CNSConfigReader::GetFloat CNSConfigReader::GetInt CNSConfigReader::GetString CNSConfigWriter::Save CNSConfigWriter::Set CNSConfigWriter::SetBool	
framework_unified_Logger_001	Logger	初期化 *initialize*	NsLogDetermineLogMethod NsLogSetLogMethod NsLogSet_CWORD78_LogParams NsLogSetProcessName _CWORD78__SET_ZONES ZONEMASK	
framework_unified_Logger_002		**ログ出力** ***log output***	**NsLog** **NsLog0** **NsLogData** **NsLogIsPLogEnabled** **NSLogPrintPerformanceLog** **NsLogSetLogMethod** **NsLogSetProcessName**	

			NsLogSetRealtimeLog NsLogTime _CWORD78__SET_ZONES _CWORD78_LOG
framework_unified_Logger_003		強制クリア *force clear*	NsForceClose
framework_unified_Logger_004		ログ出力の制御ワード（設定・取得） *control word of log output (set・get)*	IS_ZONE_SET NsLogGetControlMask NsLogIsZoneSet NsLogSetControlMask NsLogSet_CWORD78_LogFlag NsLogSetZones _CWORD78__SET_ZONES
framework_unified_Logger_005		ログファイル操作 *logfile handle*	NSLogGet_CWORD78_logFileName NSLogGet_CWORD78_logFileTotalNum NSLogGet_CWORD78_logIndex _CWORD78__SET_ZONES
framework_unified_Logger_006		LogLevel（設定・取得） *LogLevel(Set・Get)*	NsLogGet_CWORD78_LogFlag NsLogInitialize NsLogSet_CWORD78_LogFlag _CWORD78__SET_ZONES
framework_unified_Logger_007		リアルタイムログ出力切替設定 *Change to realtime log output*	NsLogGetRealtimeLog NsLogInitialize NsLogSetRealtimeLog _CWORD78__SET_ZONES
framework_unified_Logger_008		ログイベント送信 *send log event*	NsLog_Cnt NsLog_Evt
framework_unified_Utility_001	Utility	Mutex *mutex*	CMutex::~CMutex CMutex::CMutex CMutex::ReadLock CMutex::Unlock CMutex::WriteLock
framework_unified_Utility_002		RWLock *RWLock*	CRWLock::~CRWLock CRWLock::CRWLock CRWLock::ReadLock CRWLock::Unlock CRWLock::WriteLock
framework_unified_Utility_003		Version *version*	C_CWORD78_Version:: ~C_CWORD78_Version C_CWORD78_Version::Build C_CWORD78_Version:: C_CWORD78_Version C_CWORD78_Version::Date C_CWORD78_Version::DateStr C_CWORD78_Version::Major C_CWORD78_Version::Minor C_CWORD78_Version::Product C_CWORD78_Version:: ProductVersion C_CWORD78_Version::Revision C_CWORD78_Version:: Signature C_CWORD78_Version:: StrucVersion C_CWORD78_Version:: VersionStr
framework_unified_StateMachine_001	StateMachine	HSMEvent *HSMEvent*	_CWORD78_AttachHSMEventsToDispatcher _CWORD78_AttachHSMEventToDispatcher _CWORD78_CreateHSMChildThread _CWORD78_DetachHSMEventFromDispatcher _CWORD78_DetachHSMEventsFromDispatcher _CWORD78_HSMDispatcherWithArguments
framework_unified_ServiceAbstructLayer_001	ServiceAbstructLayer	セッション管理(DataPool) *session management(data pool)*	C_CWORD78__CWORD77_Service:: ~C_CWORD78__CWORD77_Se

			rvice C_CWORD78__CWORD77_Service::AddNotification C_CWORD78__CWORD77_Service:: CloseServiceOnUnavailability C_CWORD78__CWORD77_Service::CloseSession C_CWORD78__CWORD77_Service:: C_CWORD78__CWORD77_Service C_CWORD78__CWORD77_Service::m_cbResponse C_CWORD78__CWORD77_Service::m_cbSessionACK C_CWORD78__CWORD77_Service:: OpenServiceOnAvailability C_CWORD78__CWORD77_Service::OpenSession C_CWORD78__CWORD77_Service::OpenSessionAcks C_CWORD78__CWORD77_Service::SetResponseCallback C_CWORD78__CWORD77_Service:: Set_CWORD77_OpenSessionACK C_CWORD78__CWORD77_Service::SetSession C_CWORD78__CWORD77_Service::SubscribeNotifications C_CWORD78__CWORD77_Service::UnSubscribeNotifications C_CWORD78__CWORD77_Session:: ~C_CWORD78__CWORD77_Session C_CWORD78__CWORD77_Session::AttachResponseCallbacks C_CWORD78__CWORD77_Session:: C_CWORD78__CWORD77_Session C_CWORD78__CWORD77_Session:: DetachResponseCallbacks C_CWORD78__CWORD77_Session:: m_ResponseTo_CWORD77_ C_CWORD78__CWORD77_Session:: OpenSessionAcknowledge C_CWORD78__CWORD77_Session::SetSessionType GetRespDataFrom_CWORD77_DataPool GetRespNotfnDataFrom_CWORD77_DataPool _CWORD78_GetMsgDataOfSize _CWORD78_GetOpenSessionHandle _CWORD78_SetSessionHandle SetReqDataIn_CWORD77_DataPool	
frameworkUnified_SharedMemReader_001	SharedMemoryReader	共有メモリオブジェクトをオープンする *Open shared memory object.*	CNSSharedMemReader::Open	
frameworkUnified_SharedMemReader_002		共有メモリオブジェクトのオープン状態を確認する *Check the open state of the shared memory object.*	CNSSharedMemReader:: isOpen	
frameworkUnified_SharedMemReader_003		共有メモリオブジェクトをクローズする *Close shared memory object.*	CNSSharedMemReader::Open CNSSharedMemReader::Close	
frameworkUnified_SharedMemReader_004		共有メモリオブジェクトからデータ読み込み *Read data from shared memory object.*	CNSSharedMemReader::Open CNSSharedMemReader:: isOpen CNSSharedMemReader::Read	
frameworkUnified_SharedMemReader_005		共有メモリオブジェクトのデータをファイルへ書き出し *Write the data of the shared memory object to a file.*	CNSSharedMemReader::Open CNSSharedMemReader:: isOpen CNSSharedMemReader:: DumpToFile	
frameworkUnified_SharedMem				

Reader_006		共有メモリオブジェクトのデータサイズを取得 *Get the data size of the shared memory object.*	CNSSharedMemReader::Open CNSSharedMemReader::isOpen CNSSharedMemReader::GetSize	
frameworkUnified_SharedMemReader_007		共有メモリオブジェクトのデータ読み出しポインタを初期化 *Initializes the data read pointer of the shared memory object.*	CNSSharedMemReader::Open CNSSharedMemReader::isOpen CNSSharedMemReader::SetReadPtrToWritePtr	
framework_unified_SharedMemWriter_001	SharedMemoryWriter	共有メモリオブジェクトをオープンする *Open shared memory object.*	CNSSharedMemWriter::Open	
framework_unified_SharedMemWriter_002		共有メモリオブジェクトのオープン状態を確認する *Check the open state of the shared memory object.*	CNSSharedMemWriter::IsOpen	
framework_unified_SharedMemWriter_003		共有メモリオブジェクトをクローズする *Close shared memory object.*	CNSSharedMemWriter::IsOpen CNSSharedMemWriter::Close	
framework_unified_SharedMemWriter_004		共有メモリオブジェクトにデータを書き込む *Write the data to shared memory object.*	CNSSharedMemWriter::Open CNSSharedMemWriter::IsOpen CNSSharedMemWriter::Write	
framework_unified_SharedMemWriter_005		共有メモリバッファーをクリアする *Clear shared memory buffer.*	CNSSharedMemWriter::Open CNSSharedMemWriter::IsOpen CNSSharedMemWriter::ClearBuf	
framework_unified_SharedMemReadWrite_001	SharedMemoryReadWrite	共有メモリ操作 *Operation of the shared memory.*	SetDataToShared GetDataFromShared GetLengthOfDataFromShared DiscardDataFromShared	
framework_unified_SharedMemHandle_001	SharedMemHandle	共有メモリオブジェクトをオープンする *Open shared memory object.*	CNSSharedMemWriter::Open	
framework_unified_SharedMemHandle_002		共有メモリオブジェクトのオープン状態を確認する *Check the open state of the shared memory object.*	CNSSharedMemWriter::IsOpen	
framework_unified_SharedMemHandle_003		共有メモリオブジェクトをクローズする *Close shared memory object.*	CNSSharedMem::IsOpen CNSSharedMem::Close	
framework_unified_SharedMemHandle_004		共有メモリオブジェクトからデータの読み込み *Read data from shared memory object.*	CNSSharedMem::Open CNSSharedMem::isOpen CNSSharedMem::Read	
framework_unified_SharedMemHandle_005		共有メモリオブジェクトのデータをファイルへ書き出し *Write the data of the shared memory object to a file.*	CNSSharedMem::Open CNSSharedMem::isOpen CNSSharedMem::Read CNSSharedMem::DumpToFile	
framework_unified_SharedMemHandle_006		共有メモリオブジェクトのデータサイズを取得 *Get the data size of the shared memory object.*	CNSSharedMem::Open CNSSharedMem::isOpen CNSSharedMem::Read CNSSharedMem::GetSize	
framework_unified_SharedMemHandle_007		共有メモリオブジェクトのデータ読み出しポインタを初期化 *Initializes the data read pointer of the shared memory object.*	CNSSharedMem::Open CNSSharedMem::isOpen CNSSharedMem::SetReadPtrToWritePtr	
framework_unified_SharedMemHandle_008		共有メモリオブジェクトにデータを書き込む *Write the data to shared memory object.*	CNSSharedMem::Open CNSSharedMem::IsOpen CNSSharedMem::Write	
framework_unified_SharedMemHandle_009		共有メモリバッファーをクリアする *Clear shared memory buffer.*	CNSSharedMem::Open CNSSharedMem::IsOpen CNSSharedMem::Write	

			CNSSharedMem::ClearBuf	

内部処理 エラーユースケース一覧 *[internal processing error use-case list]*

表.内部処理エラーユースケース一覧

エラー番号 *[error number]*	機能カテゴリ *[function category]*	ユースケース名 *[use-case name]*	戻り値 *[return value]*	備考 *[notes]*
framework_unified_RingBuffer_Error_001	RingBuffer	write/read ringbuffer	e_CWORD78_StatusFail	
framework_unified_Timer_Error_001	Timer	timer（API）	NULL	

サービス起動 */start service*

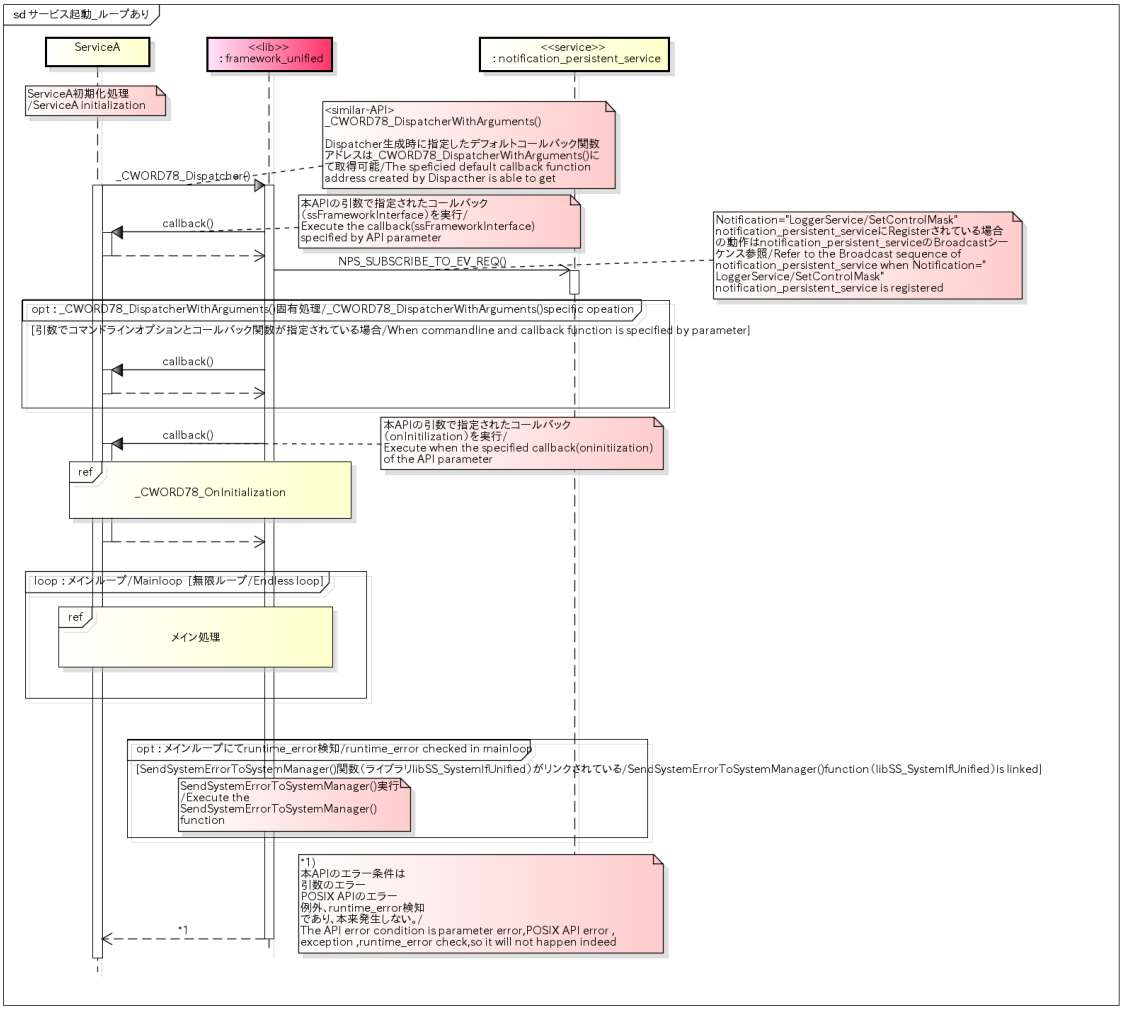
概要 *[Overview]*

本項にてサービスが起動する際のシーケンスをDispatcher内MainLoop、Dispacher外MainLoopの場合についてそれぞれ記載する。

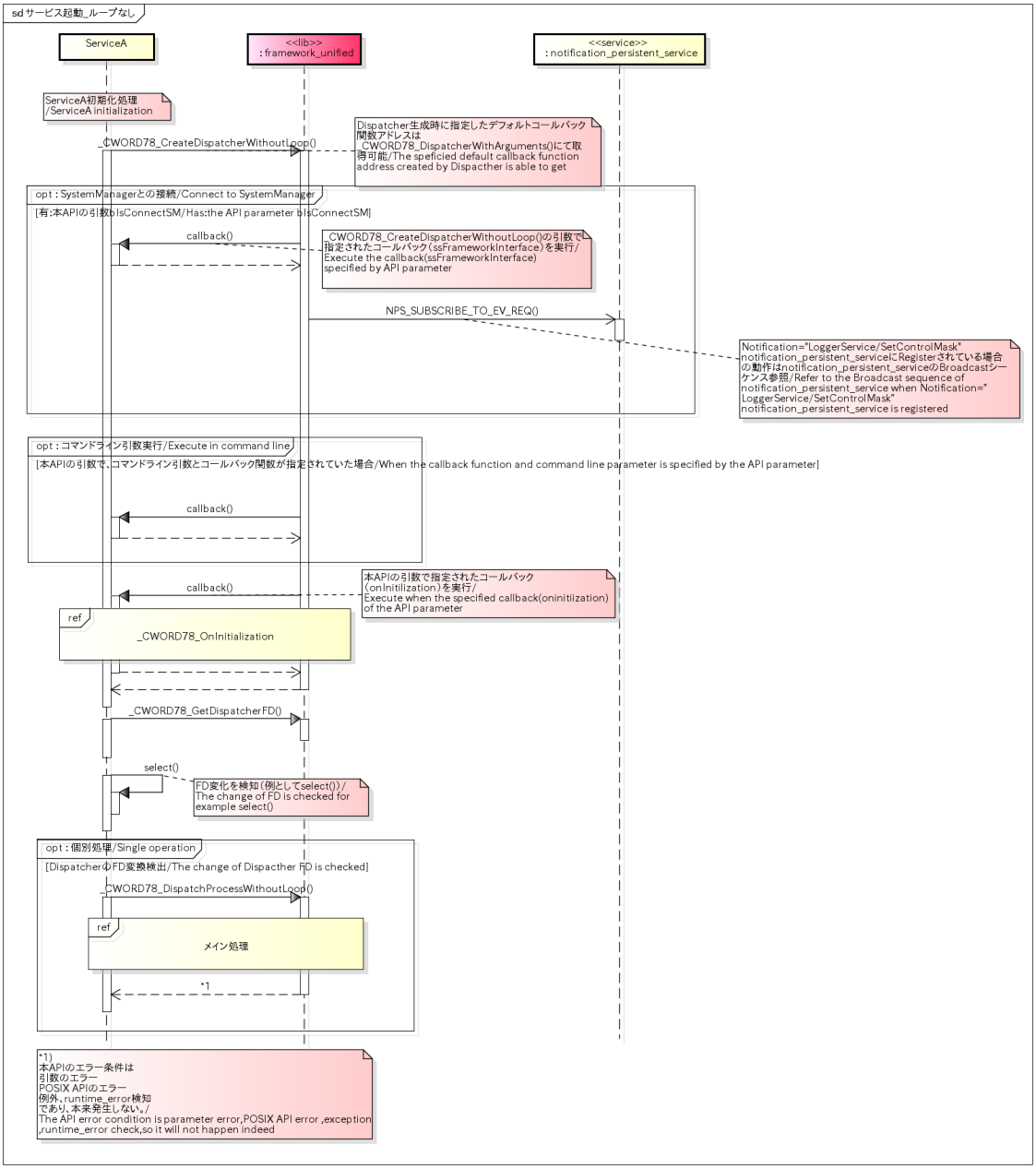
Sequences of start service at the case of MainLoop inside Dispatcher,MainLoop outside Dispatcher are as follows.

シーケンス *[Sequence]*

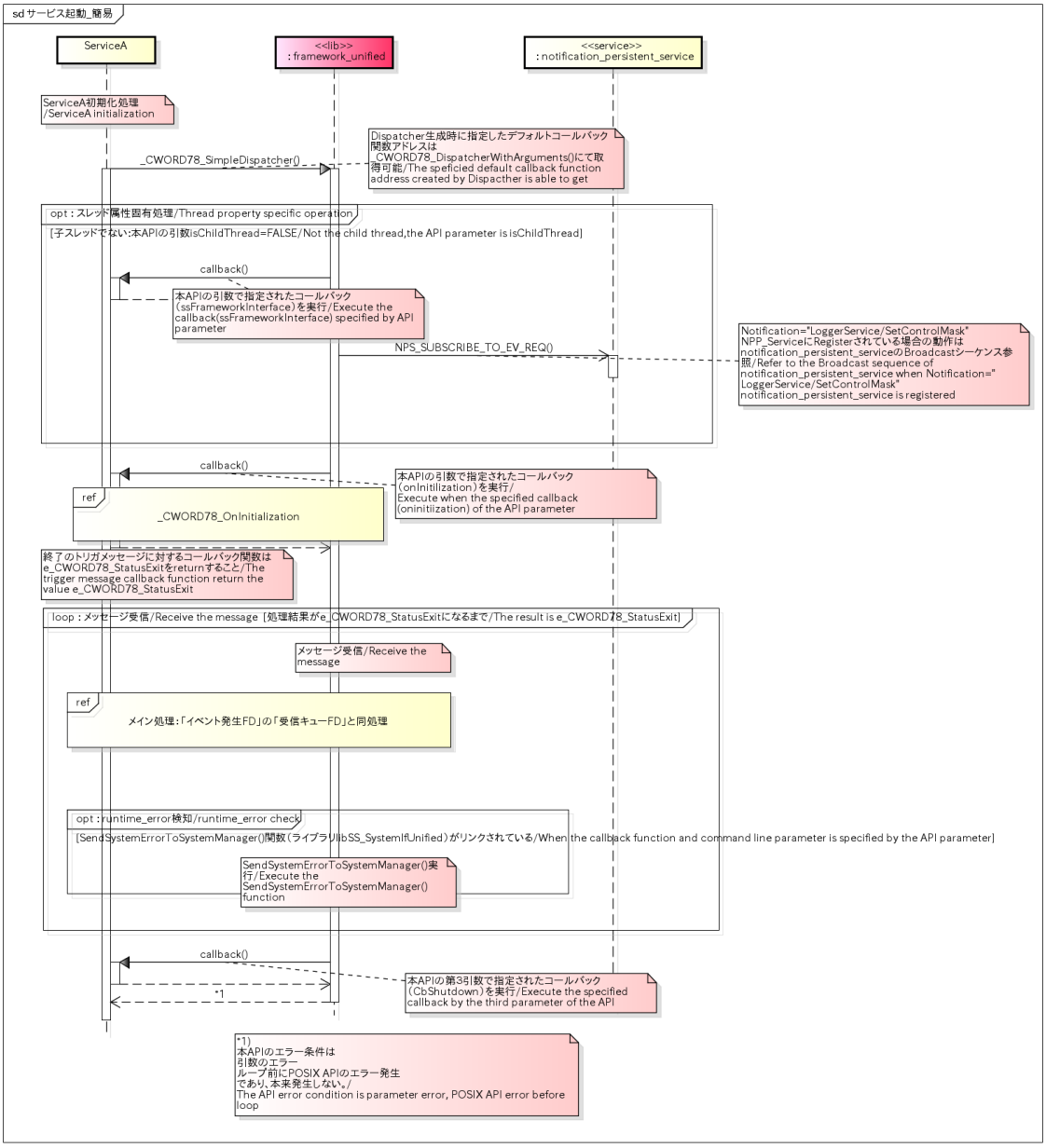
サービス起動_ループあり */start service(has loop)*



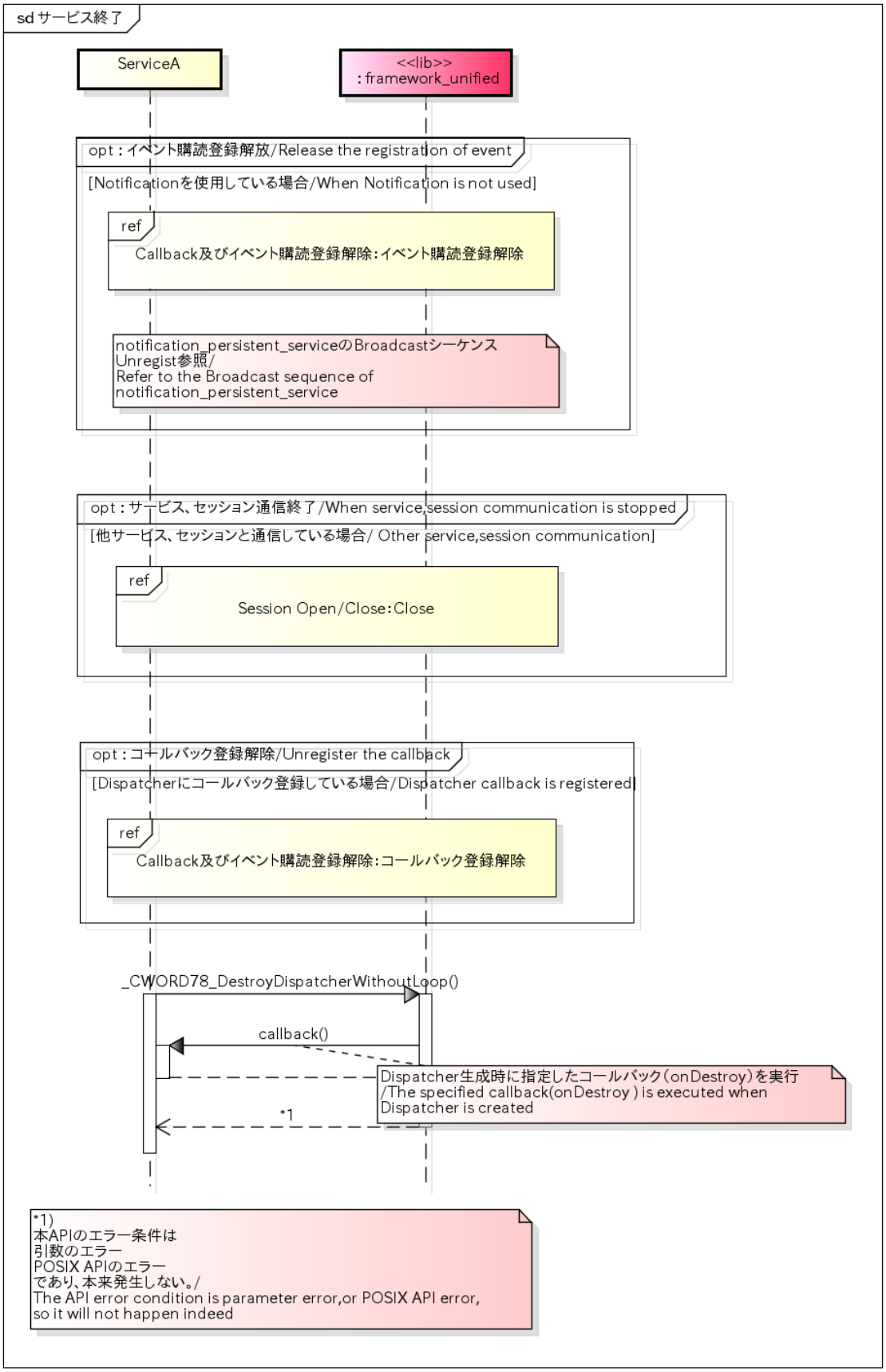
サービス起動_ループなし/*start service(no loop)*



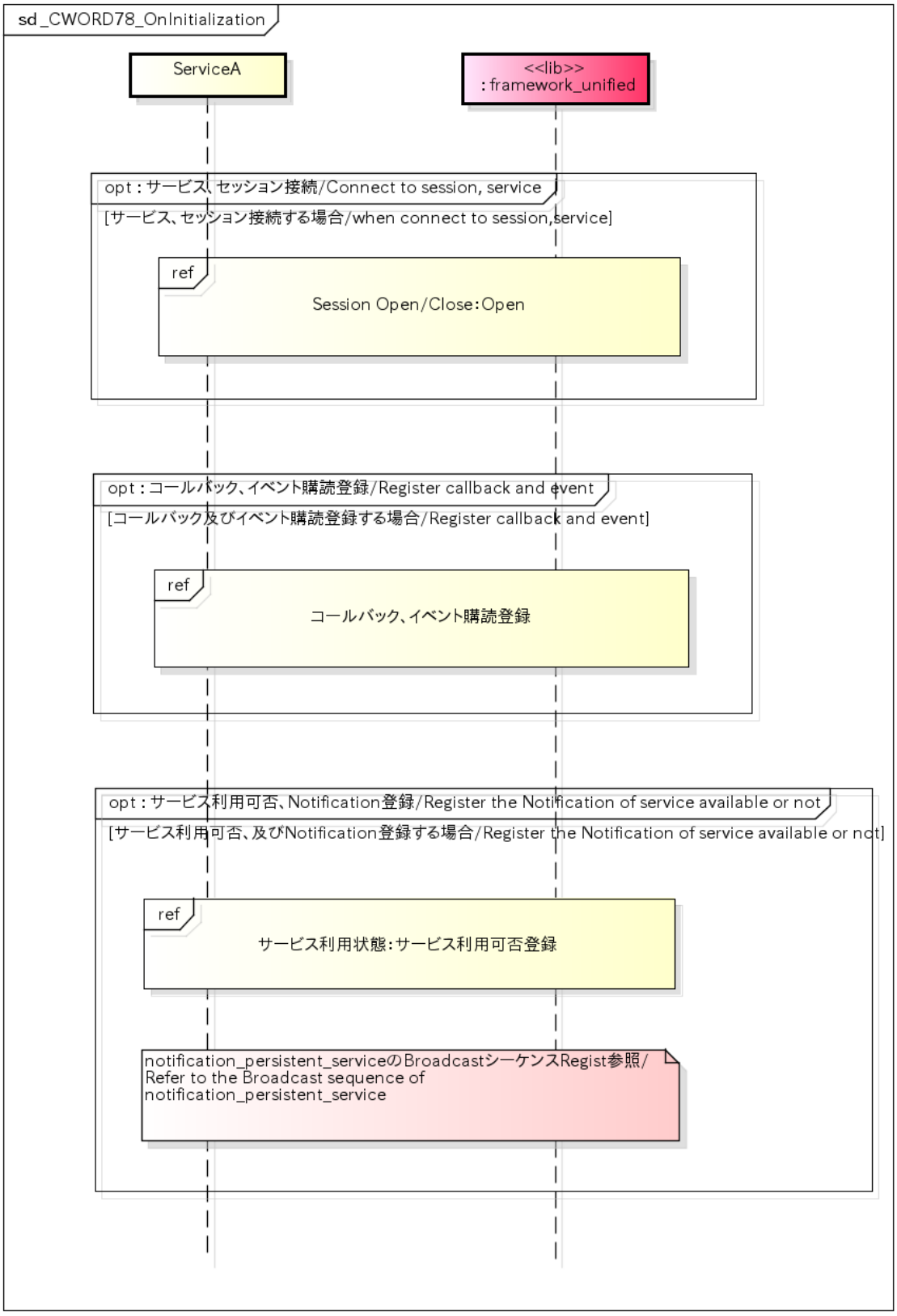
サービス起動_簡易/*start service(simple)*



サービス終了/*stop service*



_CWORD78_OnInitialization */_CWORD78_OnInitialization*



Callback/イベント購読登録 */Callback/event subscription*

概要 *[Overview]*

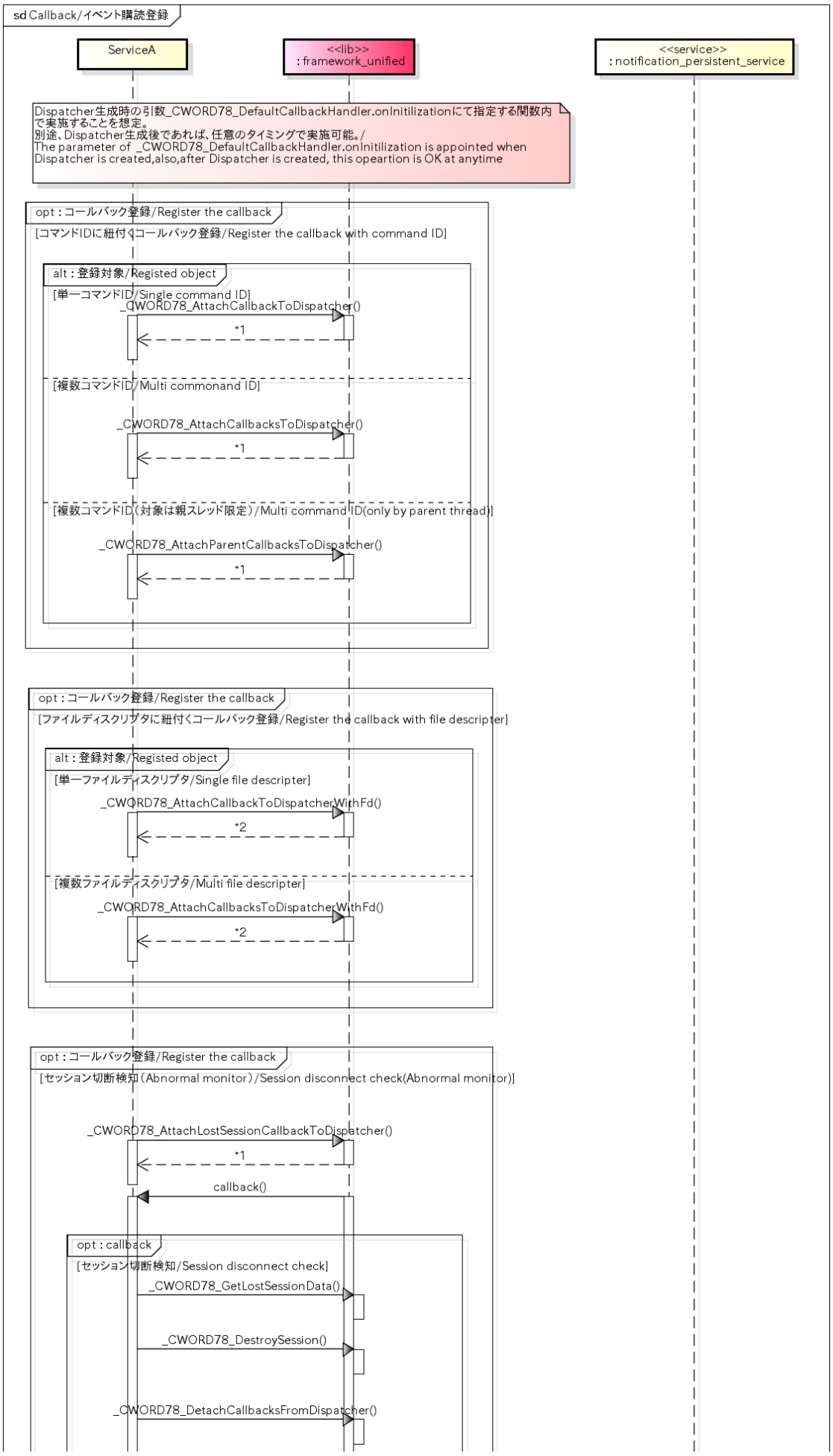
Callback関数及びイベント購読のCallback関数登録シーケンスを以下に記載する。
Callback関数を登録しておくことにより、対応するメッセージ／イベントを受信すると、DispatcherがCallback関数を実行する。

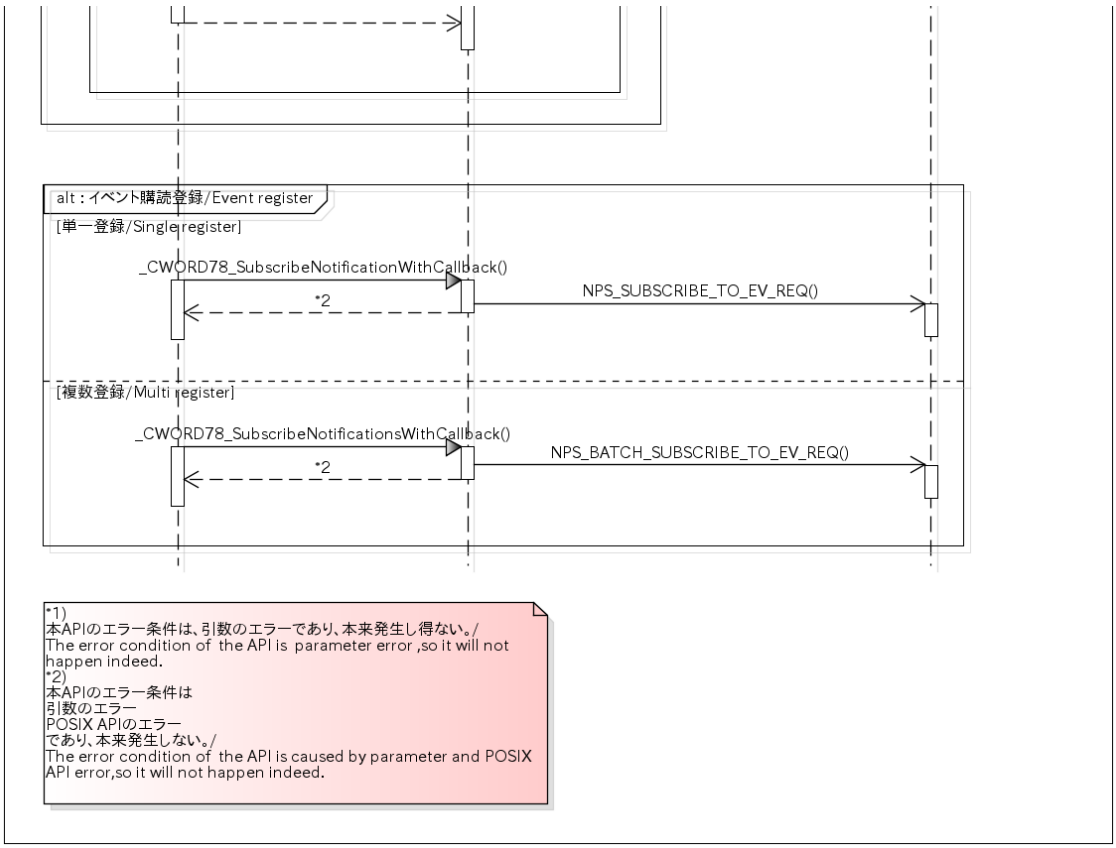
callback function and callback function register of event subscription is as follows.

If callback function is registered,when receiving the correspond message and event,Dispatcher willexecute the callback function

シーケンス *[Sequence]*

Callback/イベント購読登録 */Callback/event subscription*





Callback/イベント購読解除 */cancel callback/event subscription*

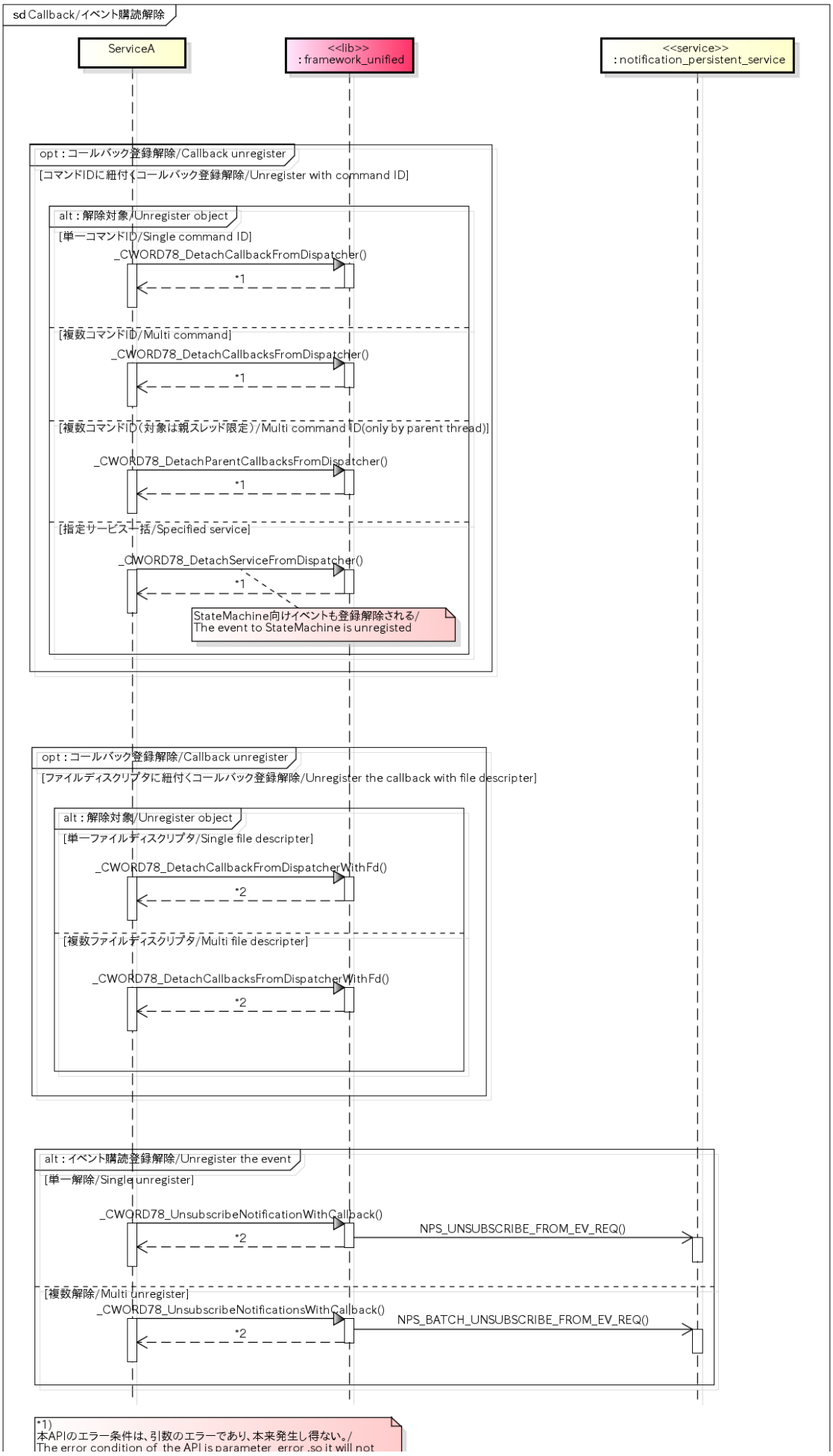
概要 *[Overview]*

Callback関数及びイベント購読のCallback関数登録解除シーケンスを以下に記載する。
Callback関数の登録解除を実施すると、Dispatcherは同Callback関数を実行しない。

Sequence of callback function and callback function cancel of event subscription is as follows.
If cancel the callback function register,Dispatcher will not execute the same callback function.

シーケンス *[Sequence]*

Callback/イベント購読解除 */cancel callback/event subscription*



happen indeed.
*2)
本APIのエラー条件は
引数のエラー
POSIX APIのエラー
であり、本来発生しない。/
The error condition of the API is caused by parameter and POSIX API error,so it will not happen indeed.

サービス利用状態 */service use status*

概要 *[Overview]*

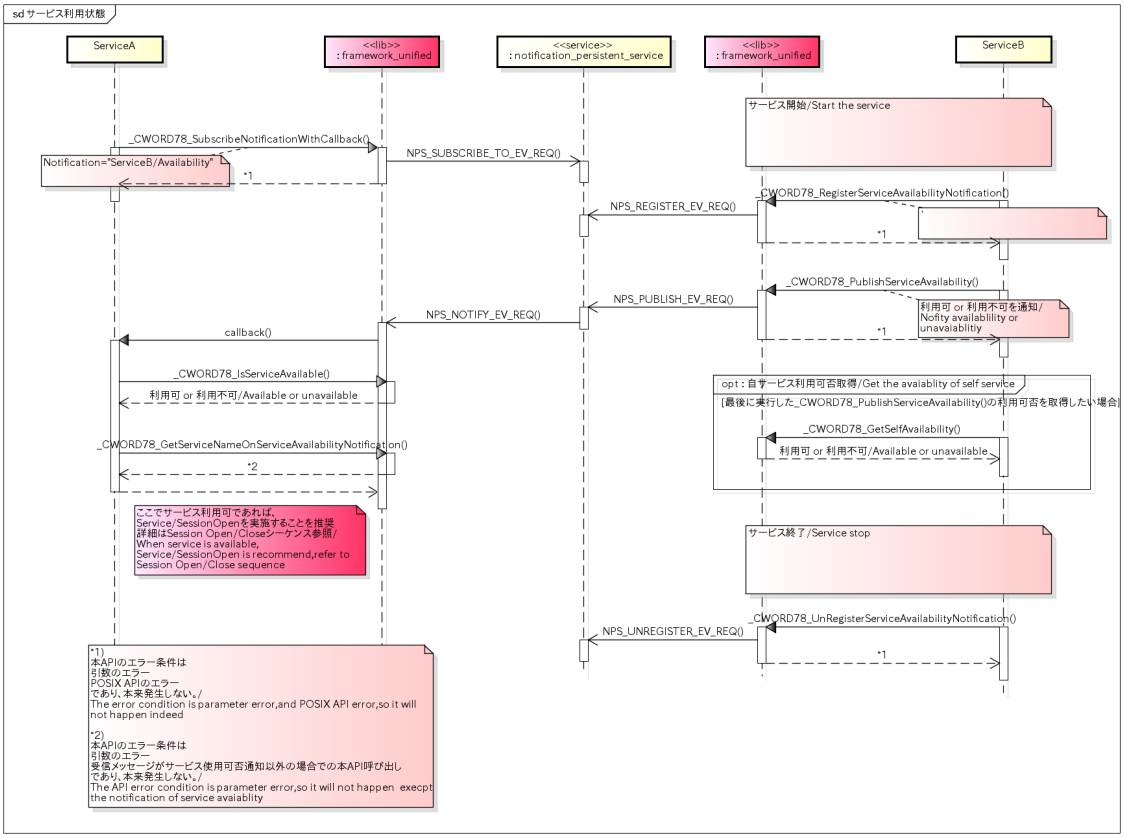
サービスの利用可否状態通知についてシーケンスを以下に記載する。
サービスは機能提供の可否状態をクライアントに通知する責務を持つ。

Sequence of notifying service use status is as follows.

Service should notify client whether function can be used ot not.

シーケンス *[Sequence]*

サービス利用状態 */service use status*



アプリケーションデータ設定/取得 */application data set/get*

概要 *[Overview]*

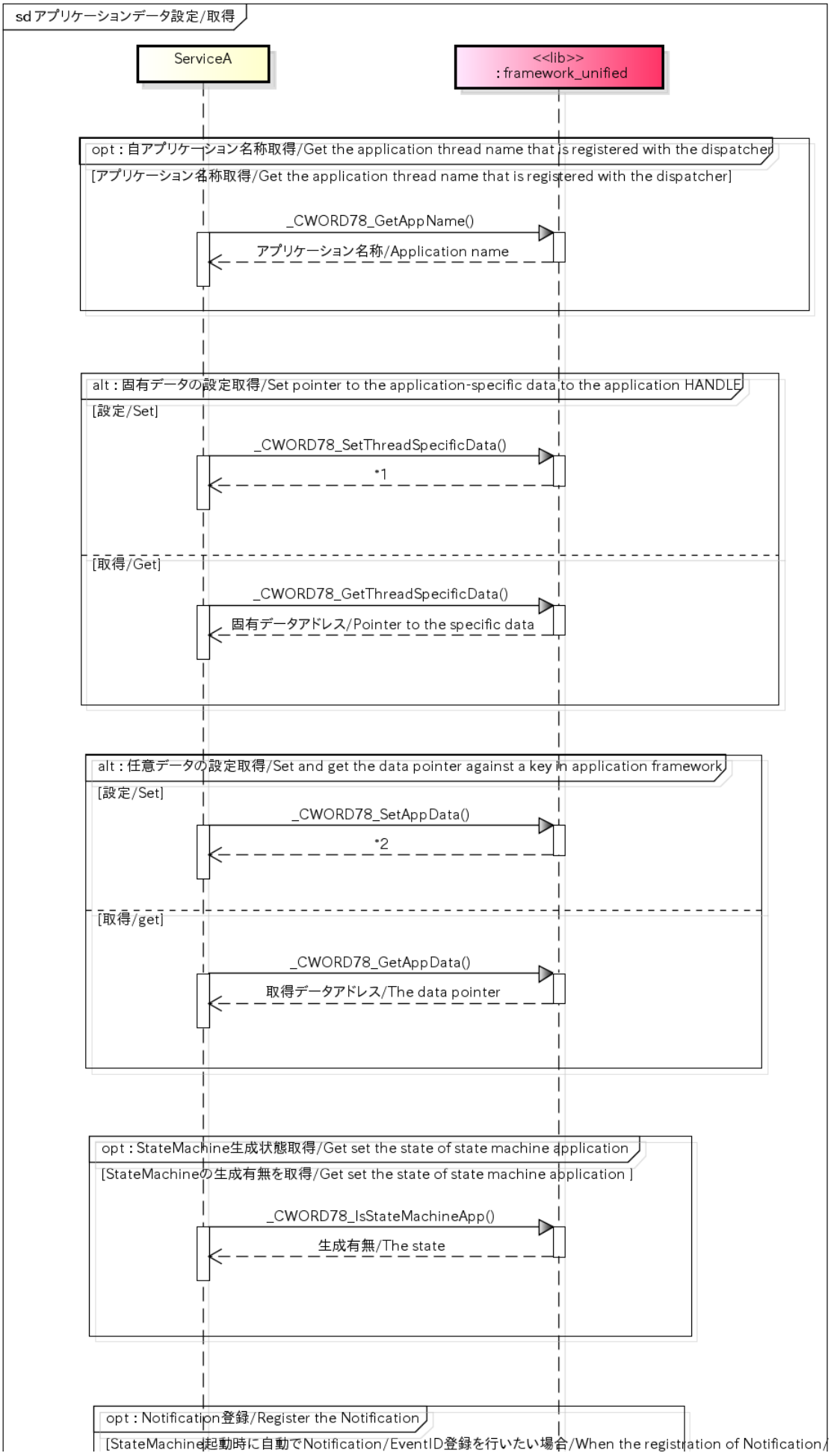
アプリケーションデータの設定／取得についてシーケンスを以下に記載する。
アプリケーションデータはDispatcherにて保持され、任意のタイミングで設定／取得可能である。

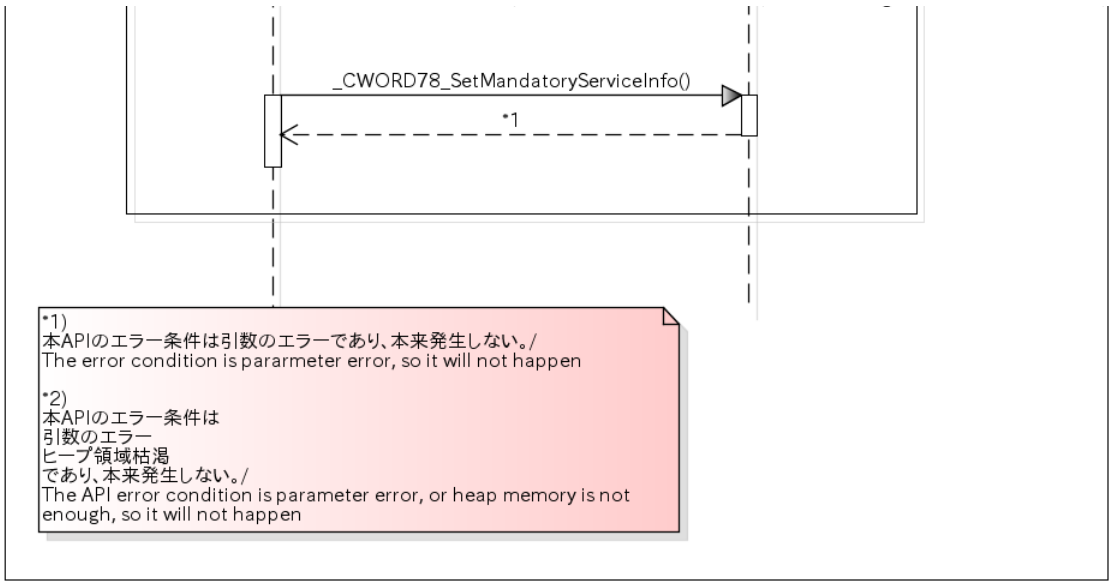
Sequence of application data set/get is as follows.

Application data is maintained by Dispatcher and be set/get at any time.

シーケンス *[Sequence]*

アプリケーションデータ設定/取得 */application data set/get*





DeferMessage */DeferMessage*

概要 *[Overview]*

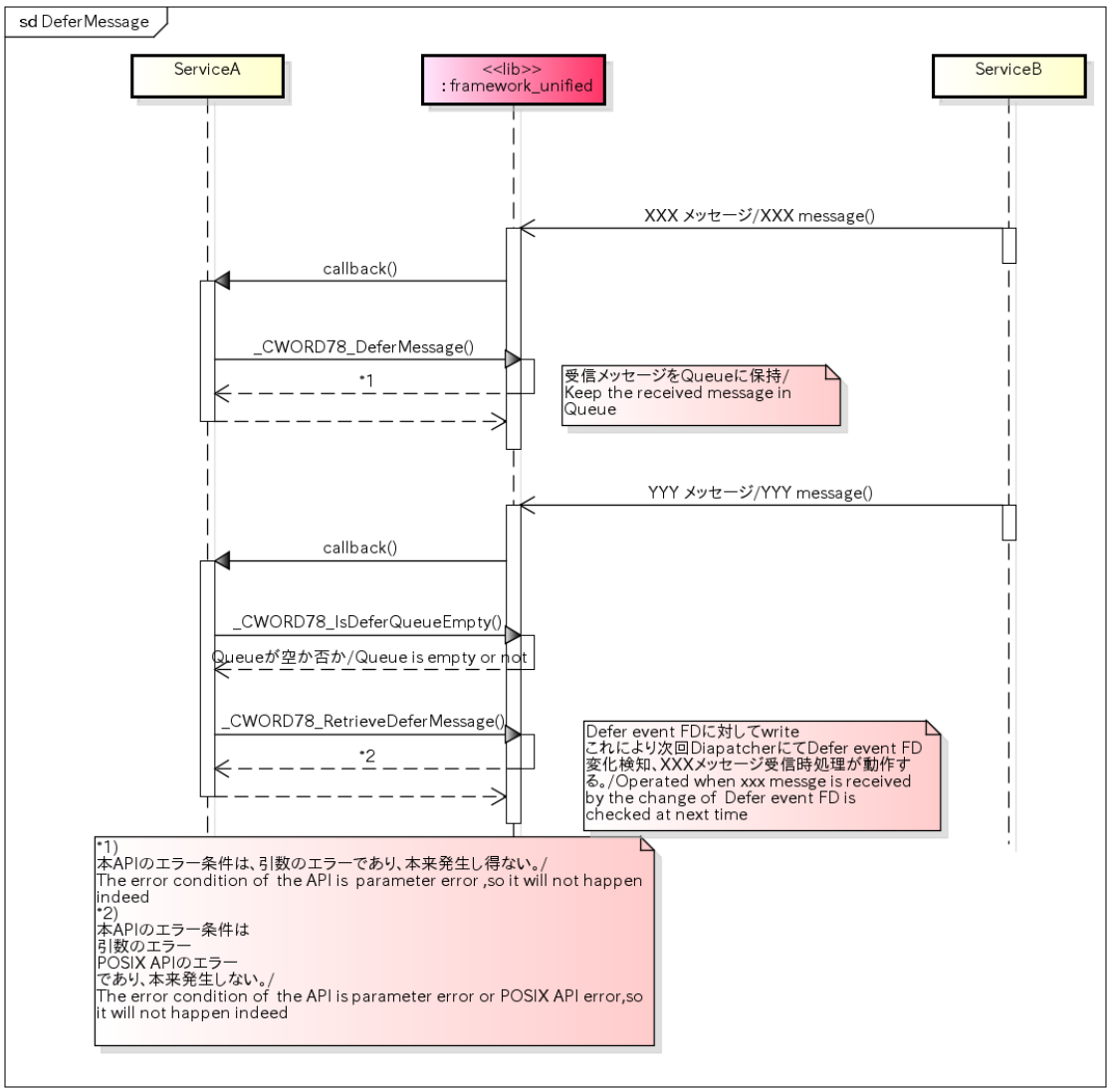
DeferMessageについてシーケンスを以下に記載する。
DeferMessageにより、受信済のメッセージを任意のタイミングで再度受信させることができる。

Sequence of DeferMessage is as follows.

The Message that has been received can be received again by DeferMessage.

シーケンス *[Sequence]*

DeferMessage */DeferMessage*



セッションオープン/クローズ */session open/close*

概要 *[Overview]*

本処理はクライアント/サービス間のセッションを確立する機能を提供する。
クライアントからサービスへセッションのオープンを要求し、サービスが生成したセッションをクライアントへ返す。
クライアントは取得したセッションを利用してサービスとの通信を行う。
セッションが不要となった際は、クライアントからサービスへセッションのクローズを要求し、サービスがセッションを破棄する。

This process provides function to build client/server session.

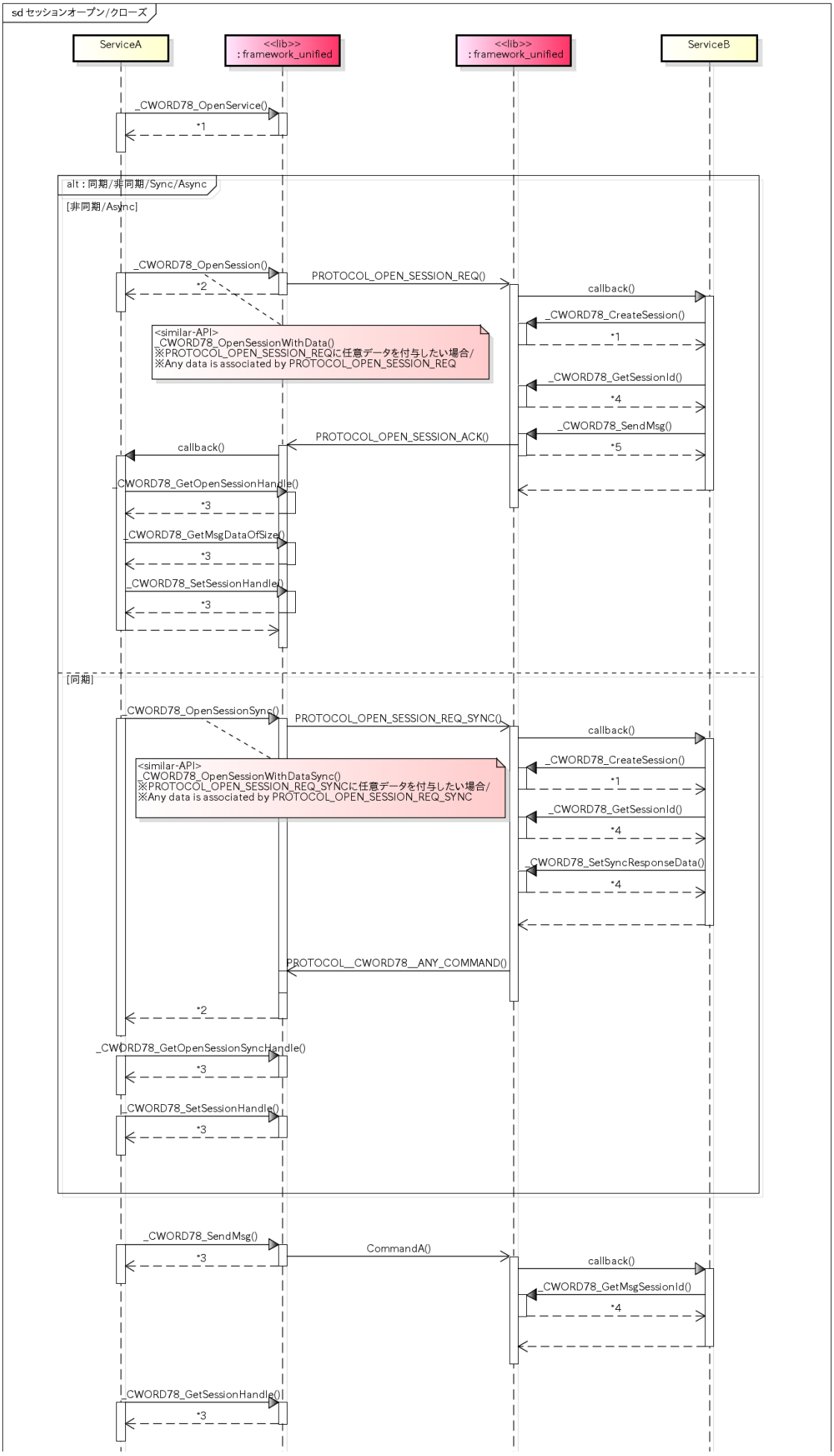
Client send session open request to service,then service creates session and return it to client.

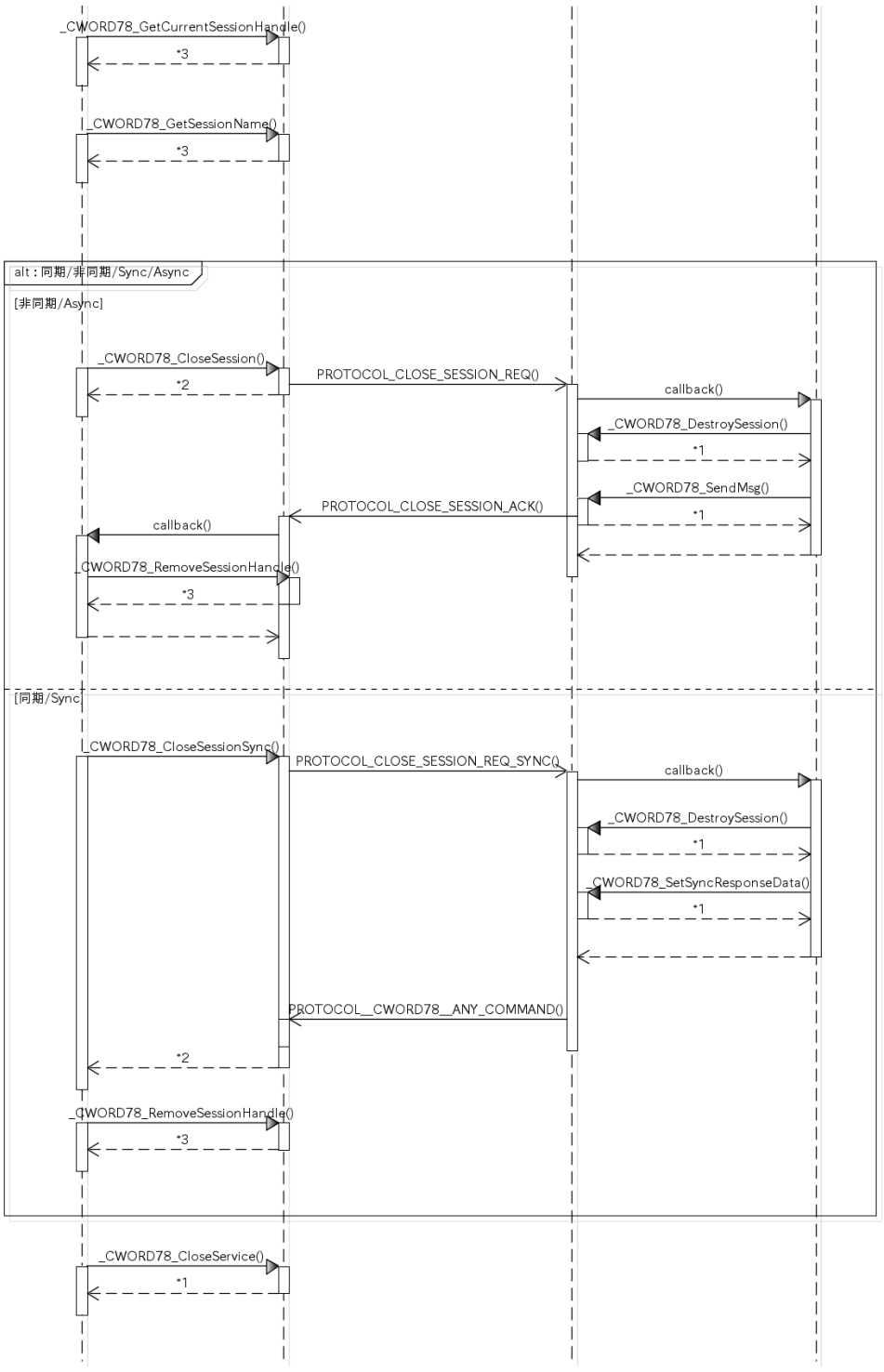
Client use this session to communicate with service.

If Client does not want to use session any more,send session close request to service,then service will delete the session.

シーケンス *[Sequence]*

セッションオープン/クローズ */session open/close*





*1)
本APIのエラー条件は引数のエラー、もしくはヒープの枯渇であり、本来発生しない。
エラーリターンされた場合は、_CWORD78_DetachCallbackFromDispatcher()等によりコールバック関数登録を解除し、終了処理を行うこと。/
The API error condition is parameter error, or heap not enough,so it will not happen indeed,
when error returned, the unregistration of callback function by _CWORD78_DetachCallbackFromDispatcher and stop operation are necessary.

*2)
本APIのエラー条件は
引数のエラー、
ヒープの枯渇
メッセージ送信システムコールの失敗
であり、本来発生しない。
また、エラー発生時、メッセージ送信は行わない。
エラーリターンされた場合は、_CWORD78_CloseService()によるキューの解放、_CWORD78_DetachCallbackFromDispatcher()等によりコールバック関数登録を解除し、終了処理を行うこと。/
The API error condition is parameter error, heap not enough,message system call failed, so it will not happen indeed,when error happened,message sending will not execute,when error returned, the callback registered by _CWORD78_DetachCallbackFromDispatcher need unregister,_CWORD78_CloseService are necessary.

*3)
本APIのエラー条件は引数のエラーであり、本来発生し得ない。
エラーリターンされた場合は、_CWORD78_CloseService()によるキューの解放、_CWORD78_DetachCallbackFromDispatcher()等によりコールバック関数登録を解除し、終了処理を行うこと。/
The API error condition is parameter error, so it will not happen indeed,when error happened,message sending will not execute,when error returned, the callback registered by _CWORD78_DetachCallbackFromDispatcher need unregister,_CWORD78_CloseService are necessary.

*4)
本APIのエラー条件は引数のエラーであり、本来発生し得ない。
エラーリターンされた場合は、_CWORD78_DestroySession()によるキューの解放、_CWORD78_DetachCallbackFromDispatcher()等によりコールバック関数登録を解除し、終了処理を行うこと。/

セッションイベント通知 */session event notification*

概要 *[Overview]*

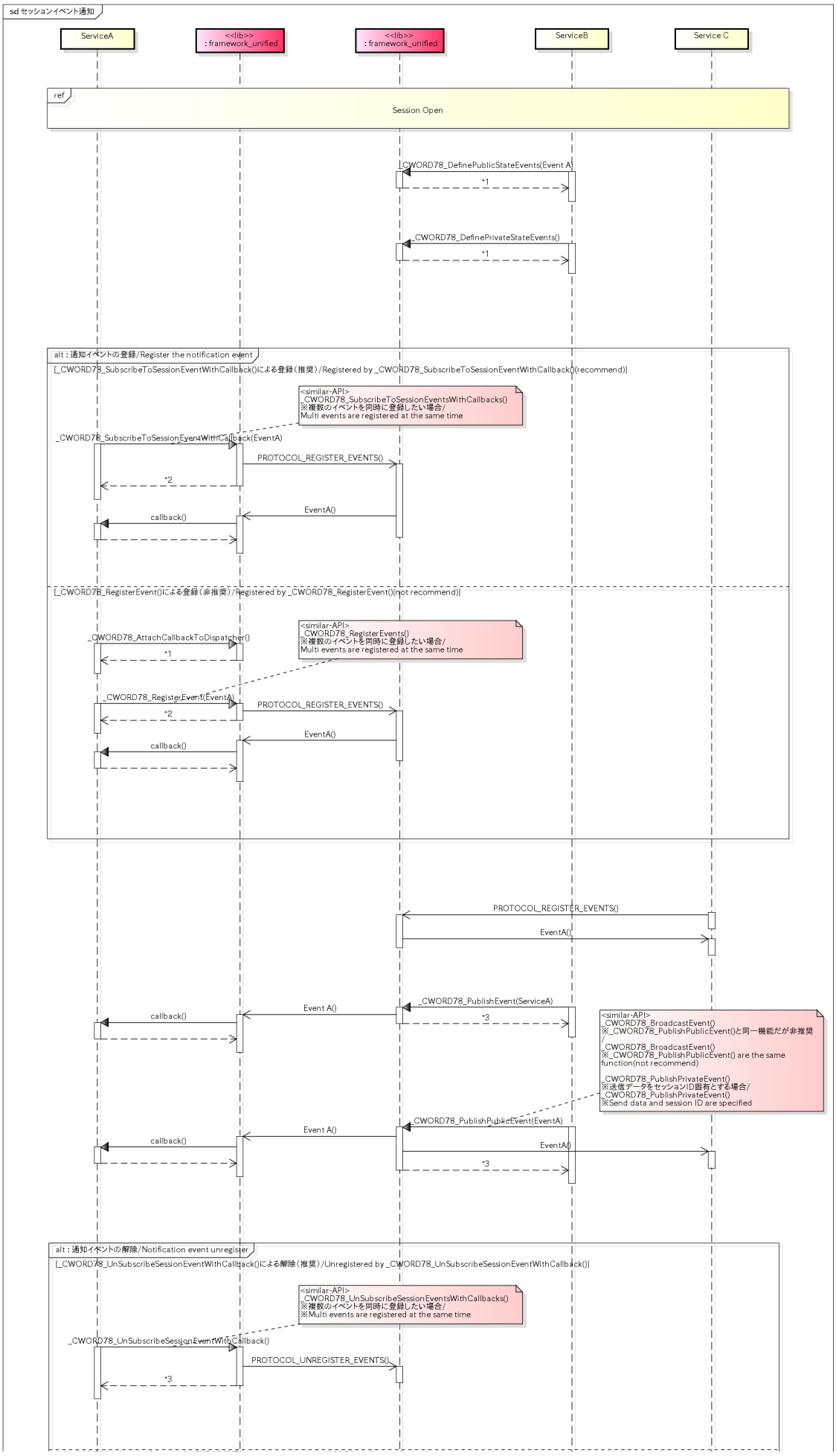
本処理は、確立されたセッション上にてPublish-Subscribe方式の通信を行う機能を提供する。
サービスは、Publishされたコマンドを Subscribeされたクライアントへ配送する。

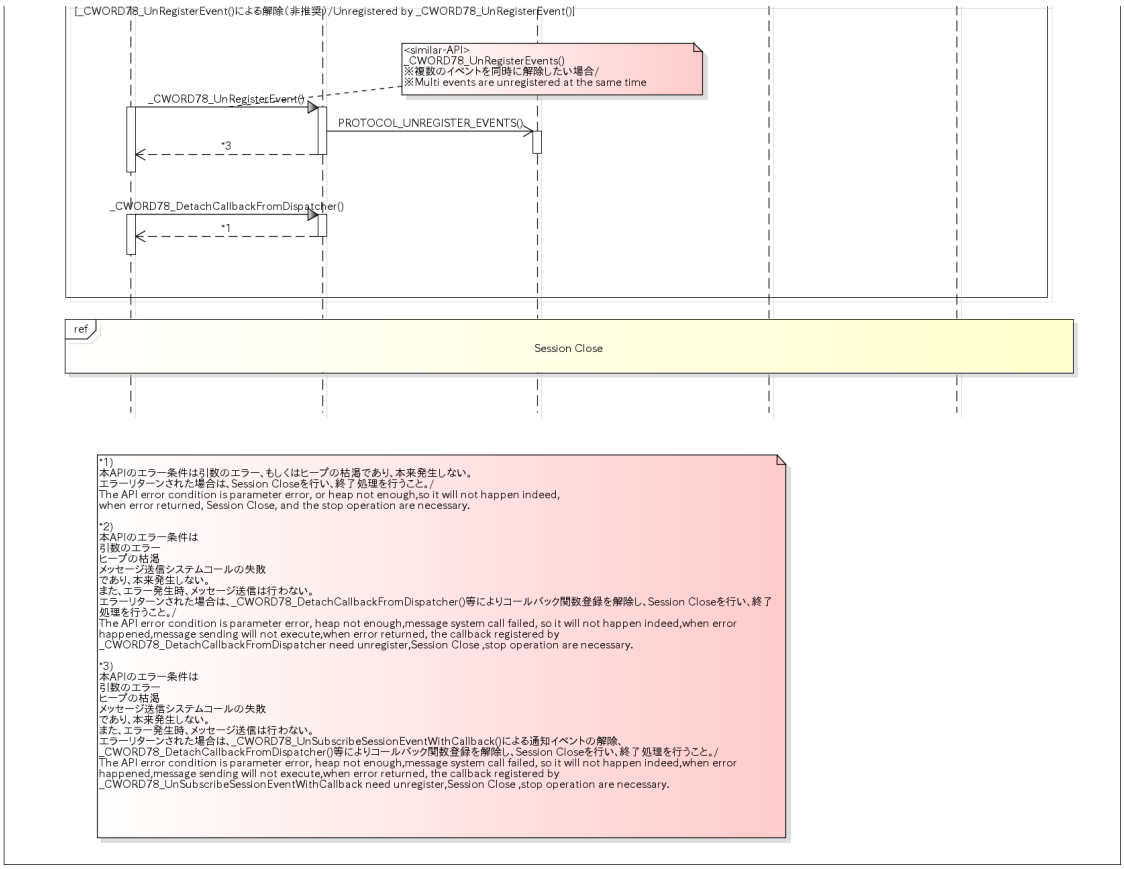
Provide communication in Publish-Subscribe way between build session.

Service sends Publish command to Subscribe client.

シーケンス *[Sequence]*

セッションイベント通知 */session event notification*





メッセージキューオープン/クローズ */message quene open/close*

概要 *[Overview]*

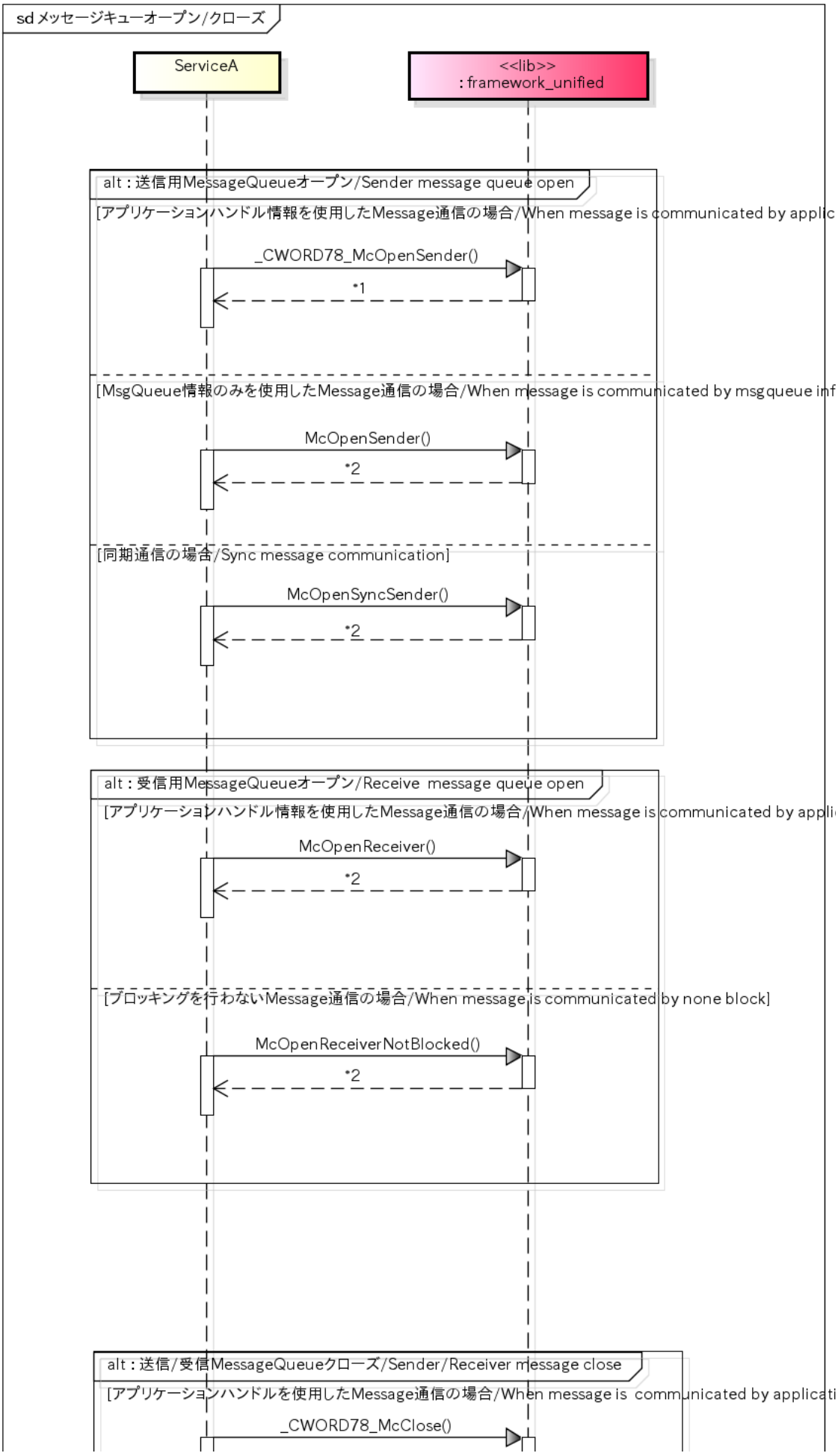
メッセージの送受信を行うために必要なメッセージキューのオープン、クローズのためのシーケンスを以下に記載する。
送信、受信、同期通信等の用途に応じてコールするAPIを切り分ける必要がある。

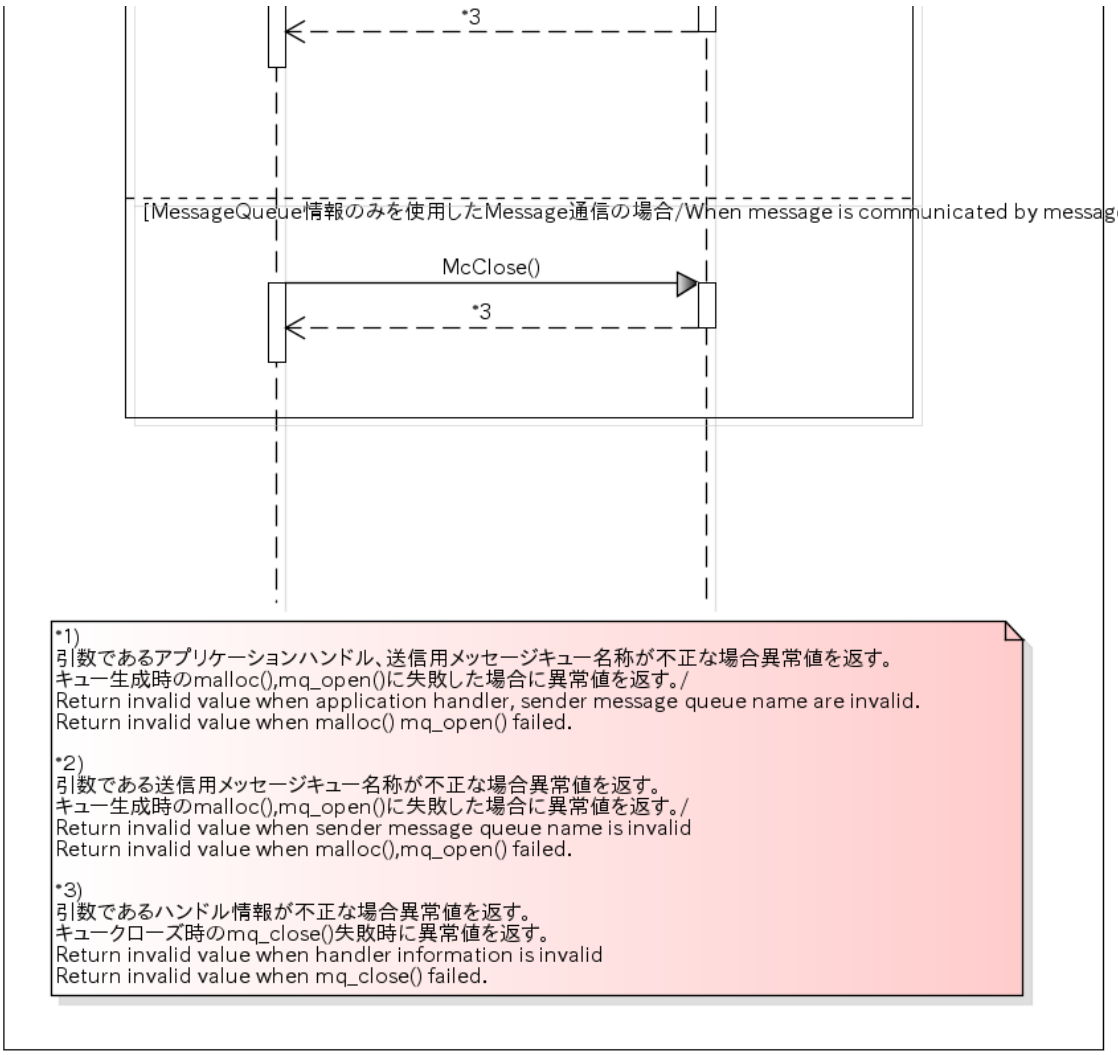
Sequence of message quene open/close is as follows,which is need when send and receive message.

The call API should be spared correspond to send,receive, sync communication etc.

シーケンス *[Sequence]*

メッセージキューオープン/クローズ */message quene open/close*





同期通信 */sync communication*

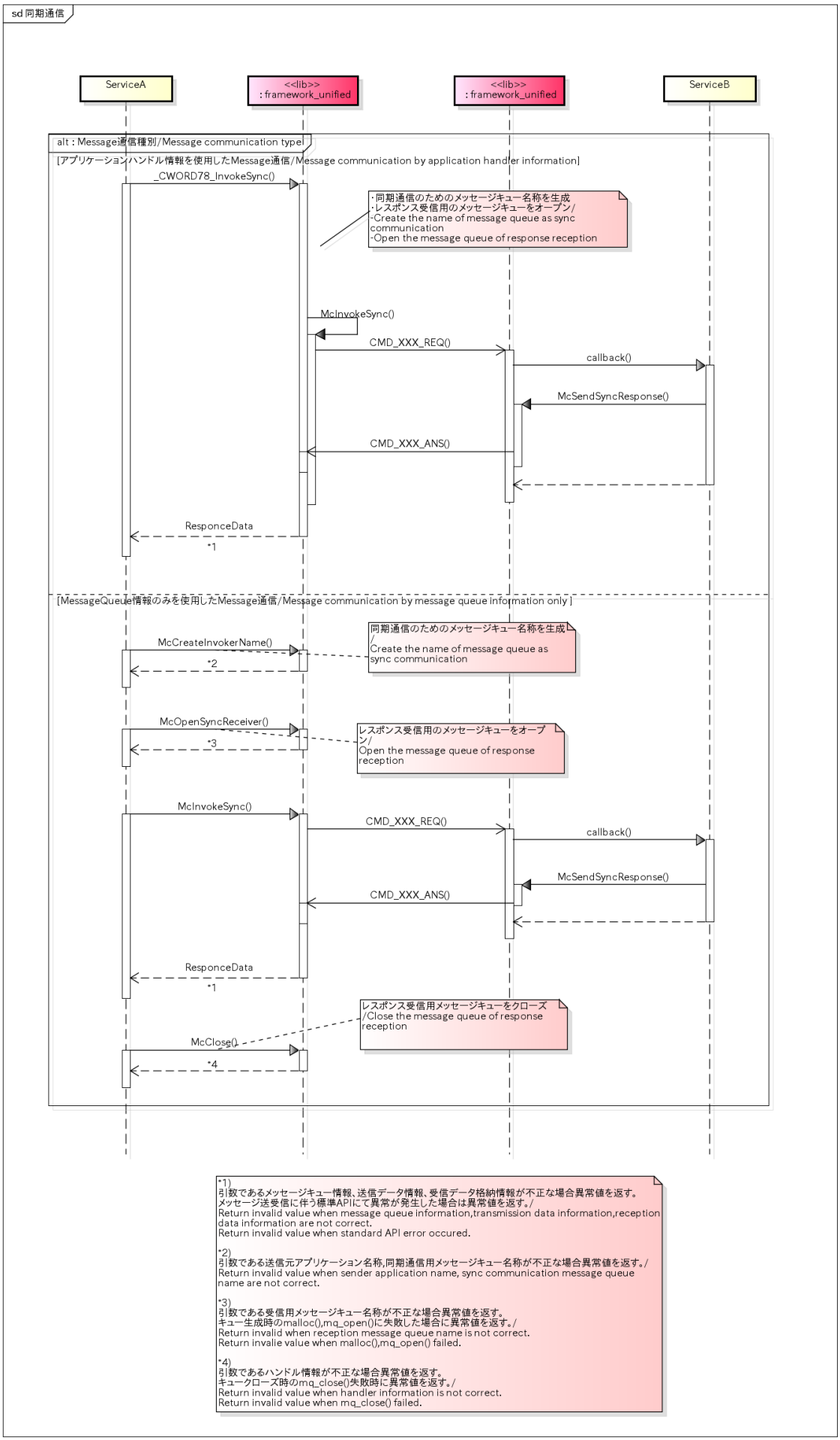
概要 *[Overview]*

同期メッセージの送信による同期通信を行うためのシーケンスを以下に記載する。

Sequence of sync communication by sending sync message is as follows.

シーケンス *[Sequence]*

同期通信 */sync communication*



非同期通信 */asynchronous communication*

概要 *[Overview]*

非同期メッセージの送信による非同期通信を行うためのシーケンスを以下に記載する。

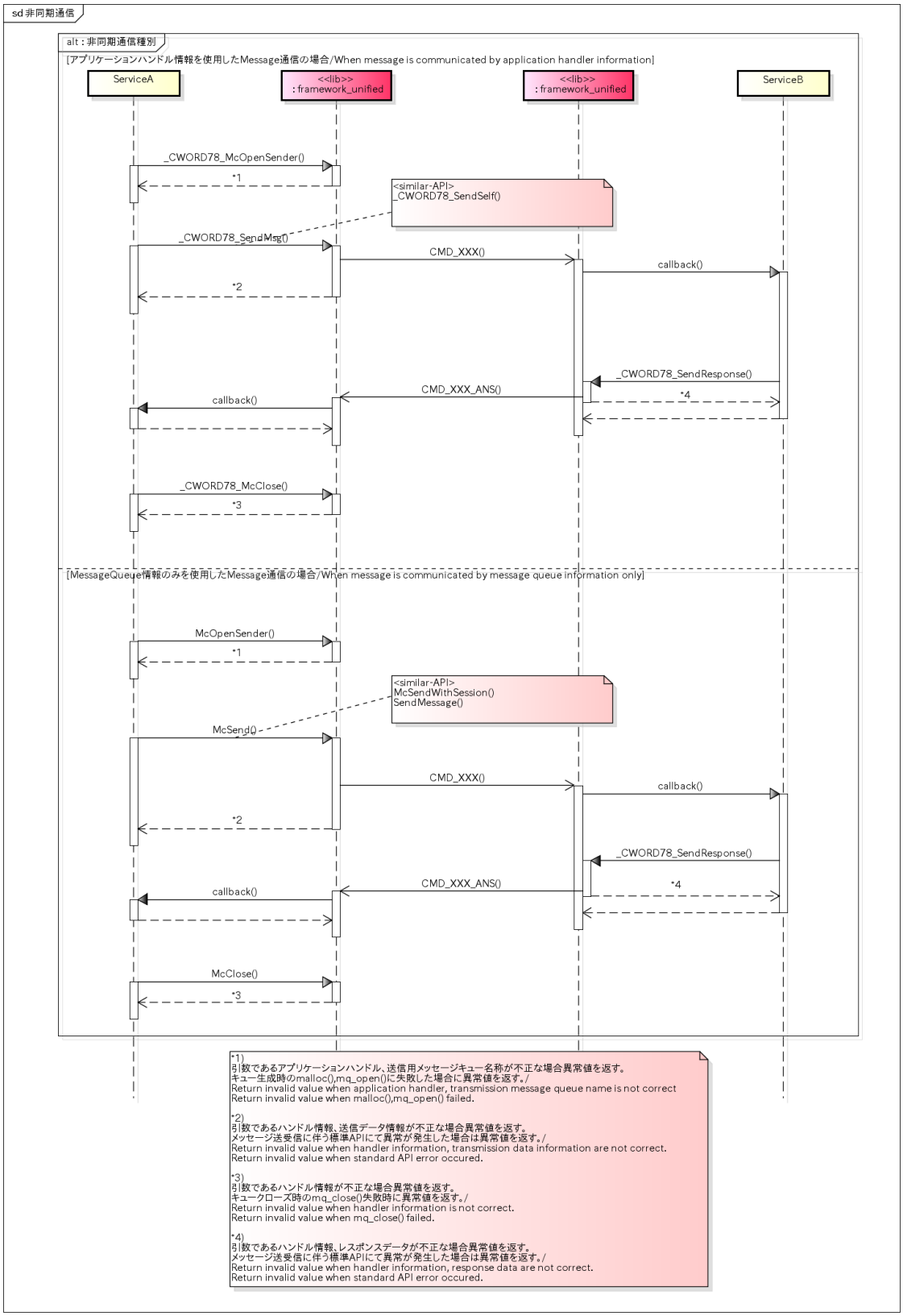
非同期メッセージの送信は、クライアントから _CWORD78_SendMsg(serviceName,CMD,prm,session)を呼び出すことで実現する。

Sequence of asynchronous communication by sending asynchronous message is as follows.

Call _CWORD78_SendMsg(serviceName,CMD,prm,session) ny client to send asynchronous message.

シーケンス *[Sequence]*

非同期通信 */asynchronous communication*



レスポンスデータ送信 */send response data*

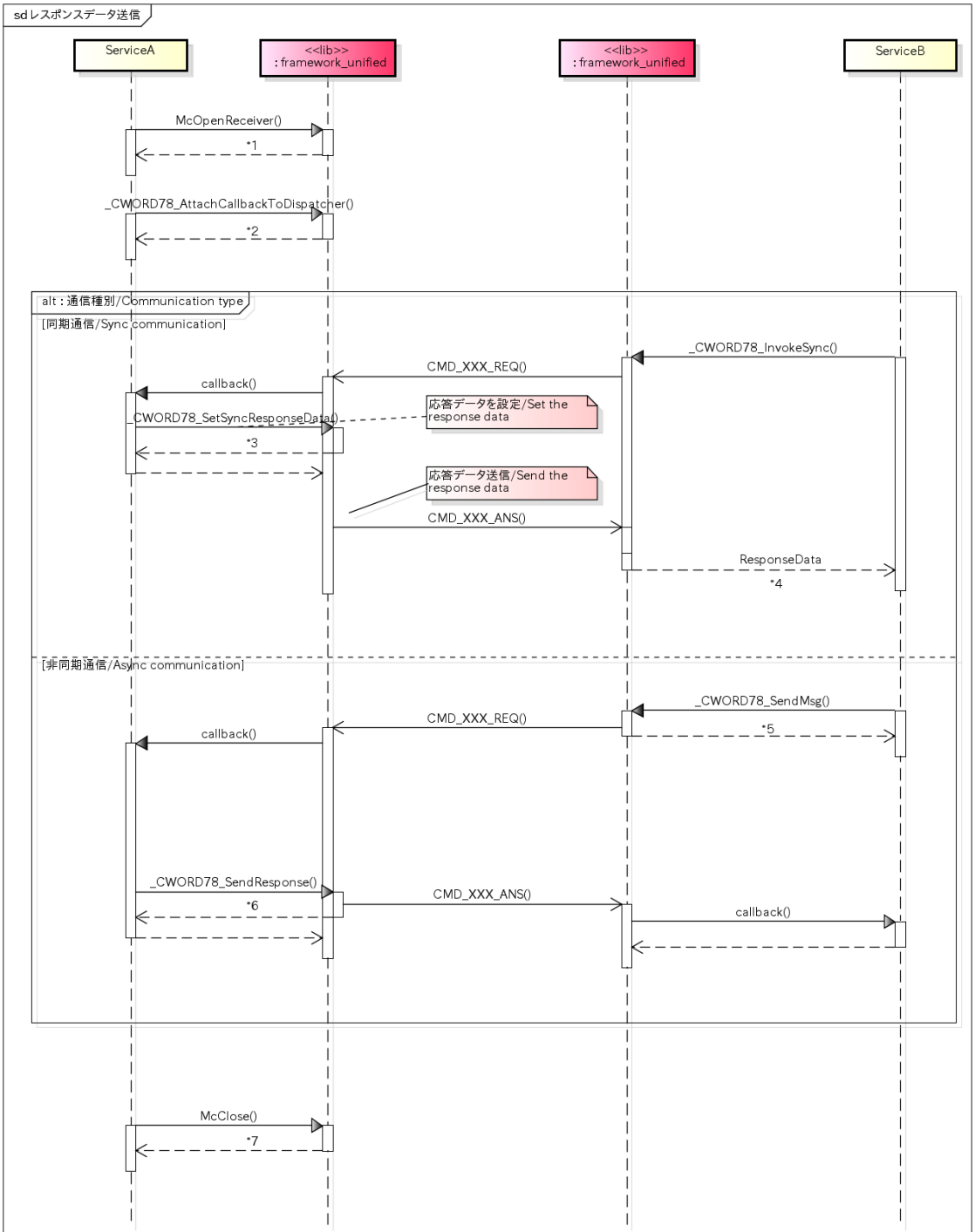
概要 *[Overview]*

同期通信、非同期通信におけるレスポンス応答のためのシーケンスを以下に記載する。

Sequence of response data by sync and asynchronous communication way.

シーケンス *[Sequence]*

レスポンスデータ送信 */send response data*



*1)
引数である受信用メッセージキュー名称が不正な場合異常値を返す。
キュー生成時のmalloc(),mq_open()に失敗した場合に異常値を返す。/
Return invalid value when the reception message queue name is invalid
Return invalid value when malloc(),mq_open() failed.

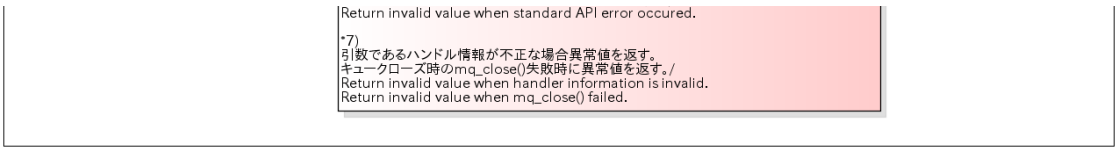
*2)
引数であるハンドル情報、サービス名称情報、コールバック情報が不正な場合は異常値を返す。/
Return invalid value when handler information,service name information,callback information are invalid

*3)
引数であるハンドル情報、レスポンスデータが不正な場合異常値を返す。/
Return invalid value when handler information,response data are invalid

*4)
引数であるメッセージキュー情報、送信データ情報、受信データ格納情報が不正な場合異常値を返す。
メッセージ送受信に伴う標準APIにて異常が発生した場合は異常値を返す。/
Return inavlid value when message queue information, transmission information, reception data information is invalid

*5)
引数であるハンドル情報、送信データ情報が不正な場合異常値を返す。
メッセージ送受信に伴う標準APIにて異常が発生した場合は異常値を返す。/
Return invalid value when handler information, transmission data information is invalid.
Return invalid value when standard API error occured.

*6)
引数であるハンドル情報、レスポンスデータが不正な場合異常値を返す。
メッセージ送受信に伴う標準APIにて異常が発生した場合は異常値を返す。/
Return invalid value when handler information,response data are invalid.



メッセージデータ受信 */receive message data*

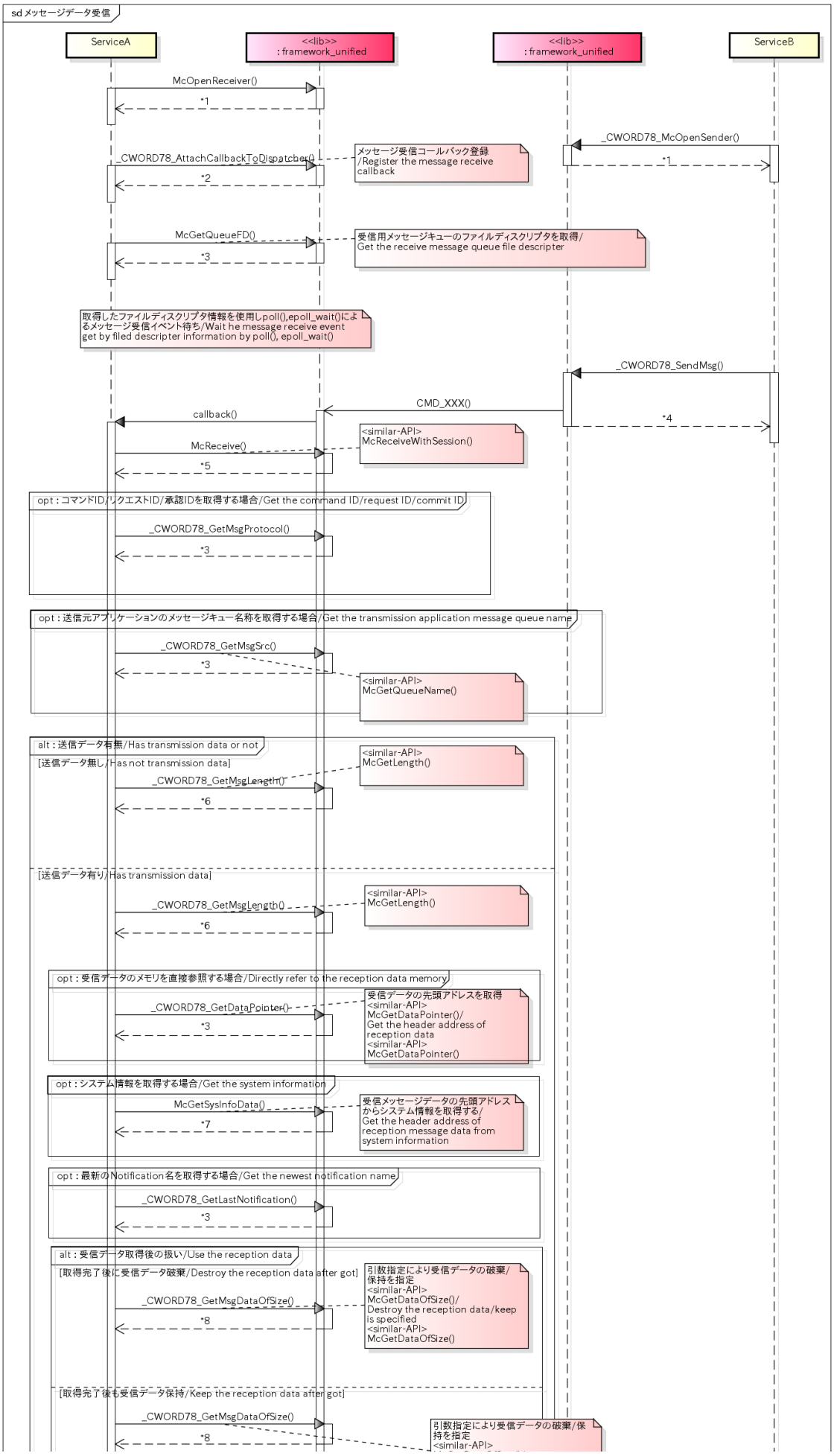
概要 *[Overview]*

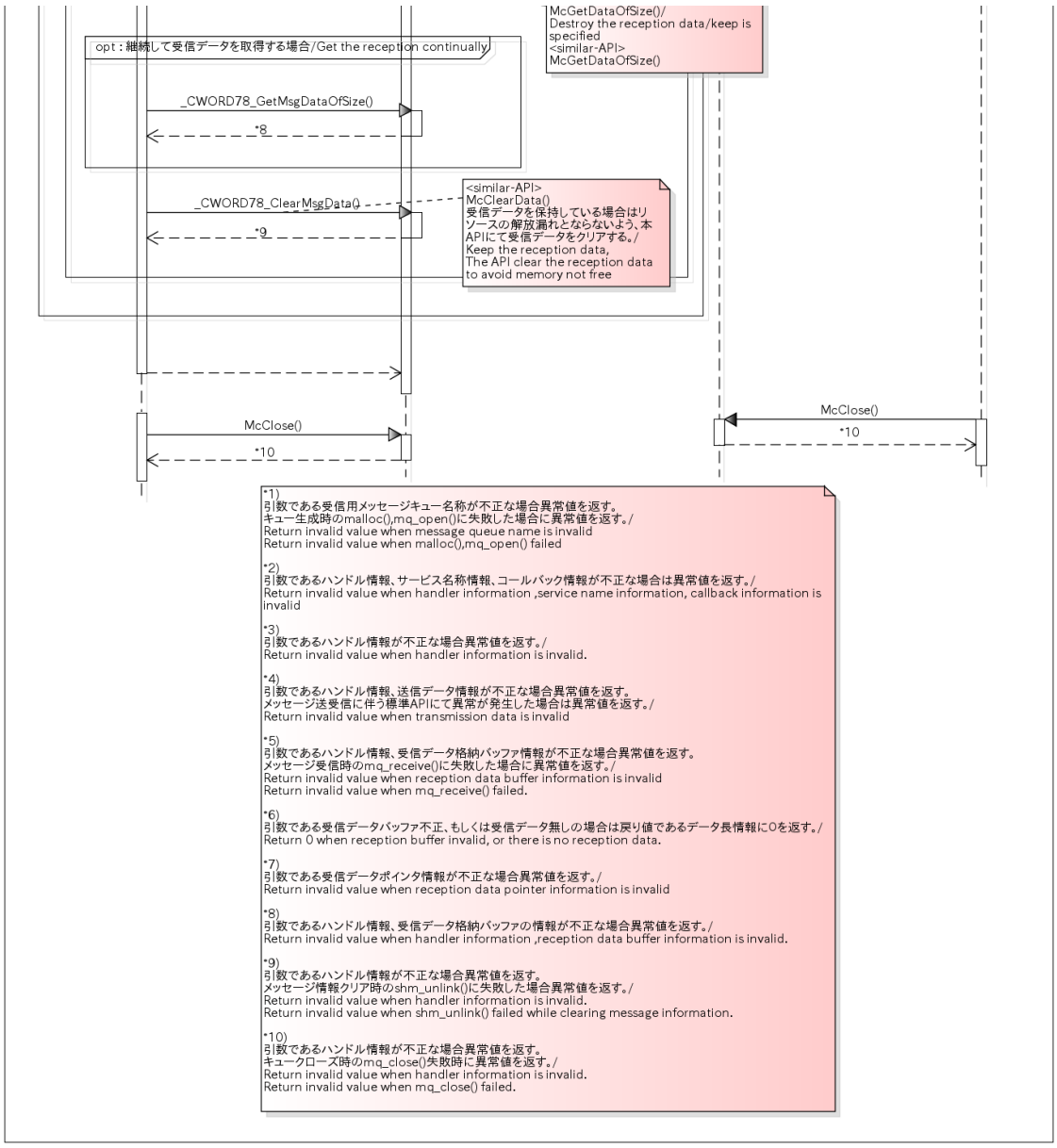
メッセージを受信した際に添付されている受信データを取得するためのシーケンスを以下に記載する。

Sequence of get received data which is attached when receiving message is as follows.

シーケンス *[Sequence]*

メッセージデータ受信 */receive message data*





ゼロコピー通信 */zero copy communication*

概要 *[Overview]*

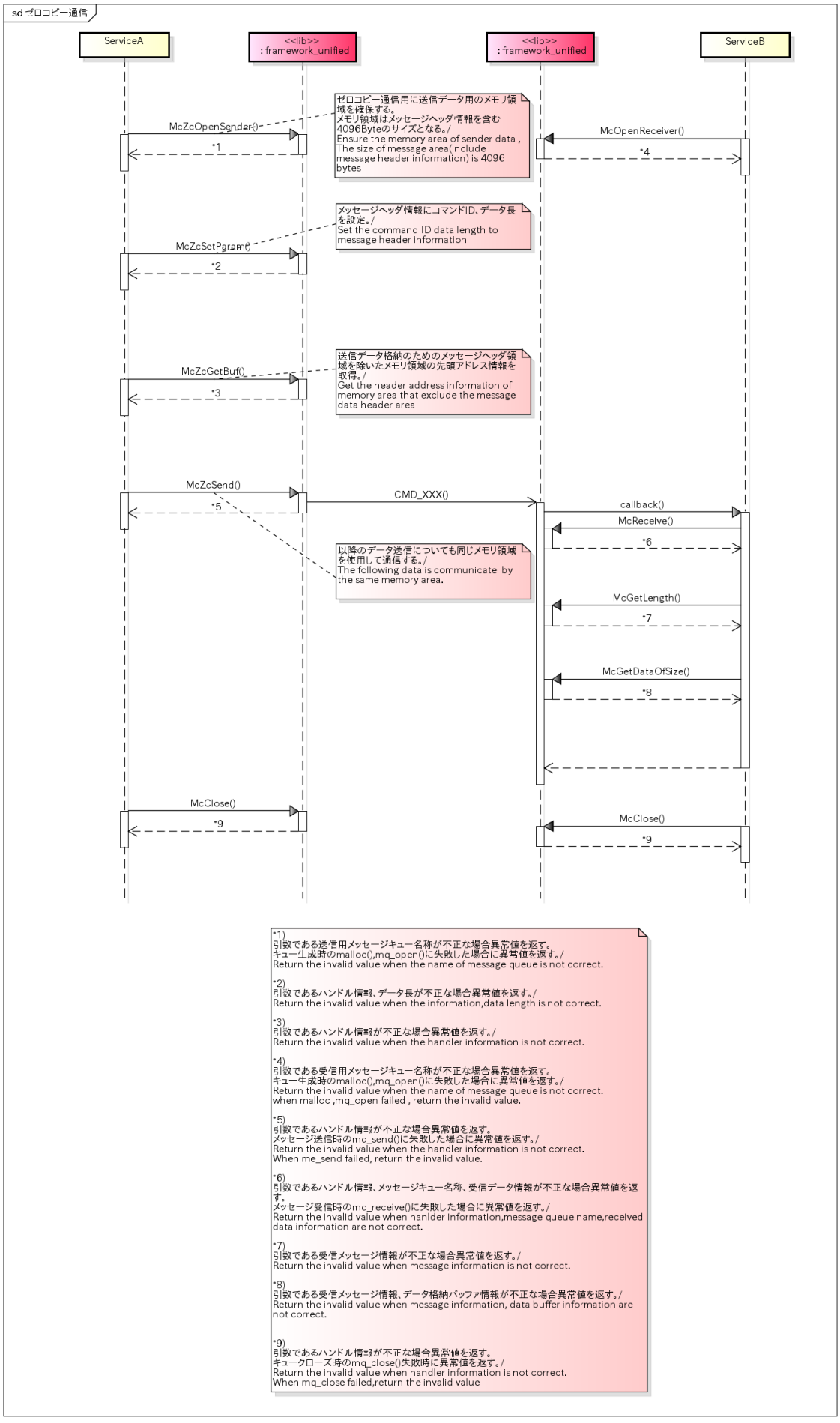
本通信はMsgの送信において、送信用のデータ格納バッファ(4KByte)のメモリ領域を通信開始時に動的メモリより確保し、以降のMsg送信時にも同メモリに送信データをコピーして使用することによりデータコピー頻発によるパフォーマンス低下を軽減しMsg通信時の負荷を軽減することを目的としている。

This communication ensures data buffer(4KByte) of sending data when communication starts, and copy sending data to the same memory after starting,

so that data copy will not be done at a high frequency,which will reduce system burden and has lower effect to performance.

シーケンス *[Sequence]*

ゼロコピー通信 */zero copy communication*



非ブロッキングメッセージ受信 */none-blocked message receive*

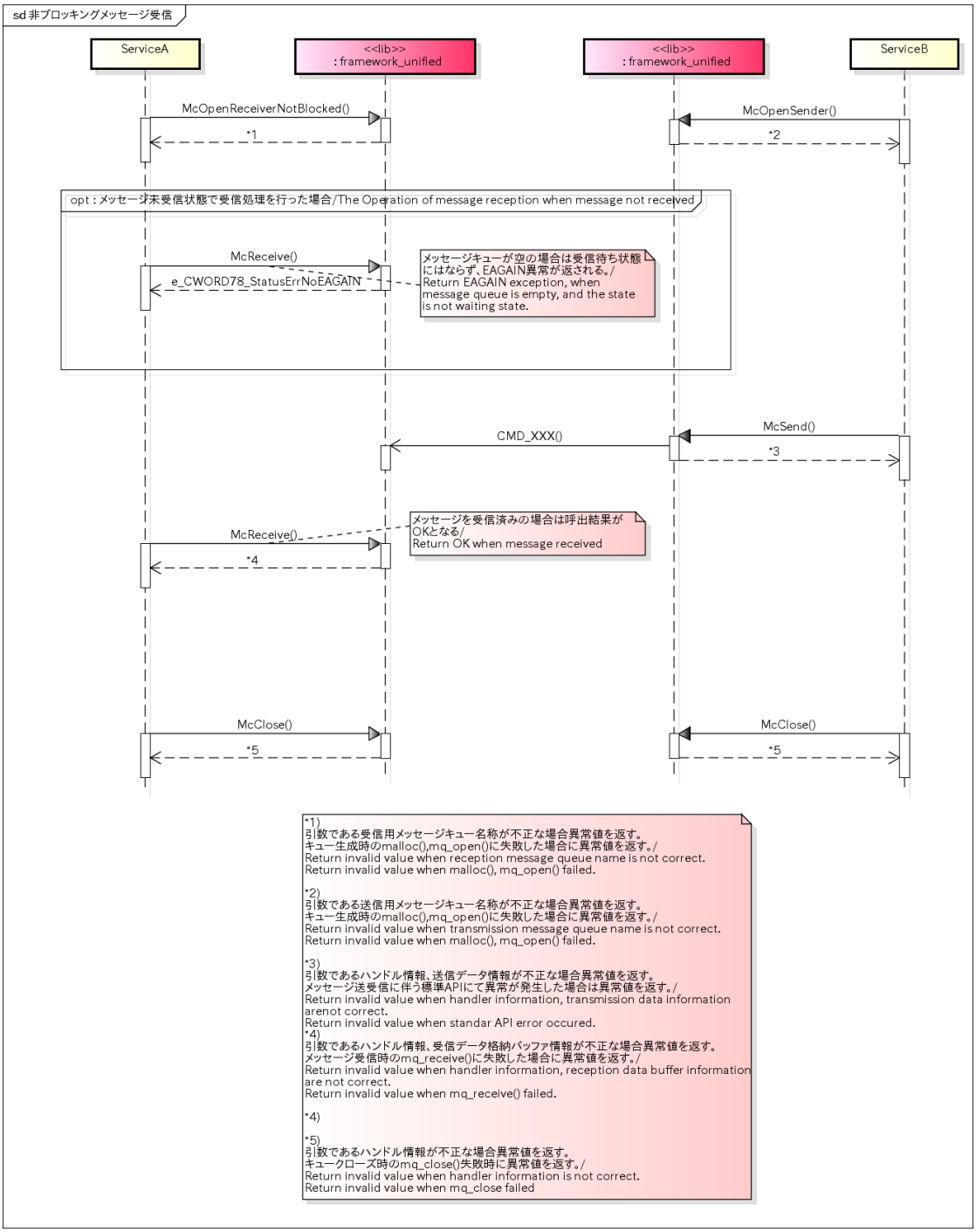
概要 *[Overview]*

メッセージを受信する際に非ブロッキング形式で受信を行うためのシーケンスを以下に記載する。

Sequence of receiving message at none-blocked way is as follows

シーケンス *[Sequence]*

非ブロッキングメッセージ受信 */none-blocked message receive*



受信メッセージ破棄 */delete received message*

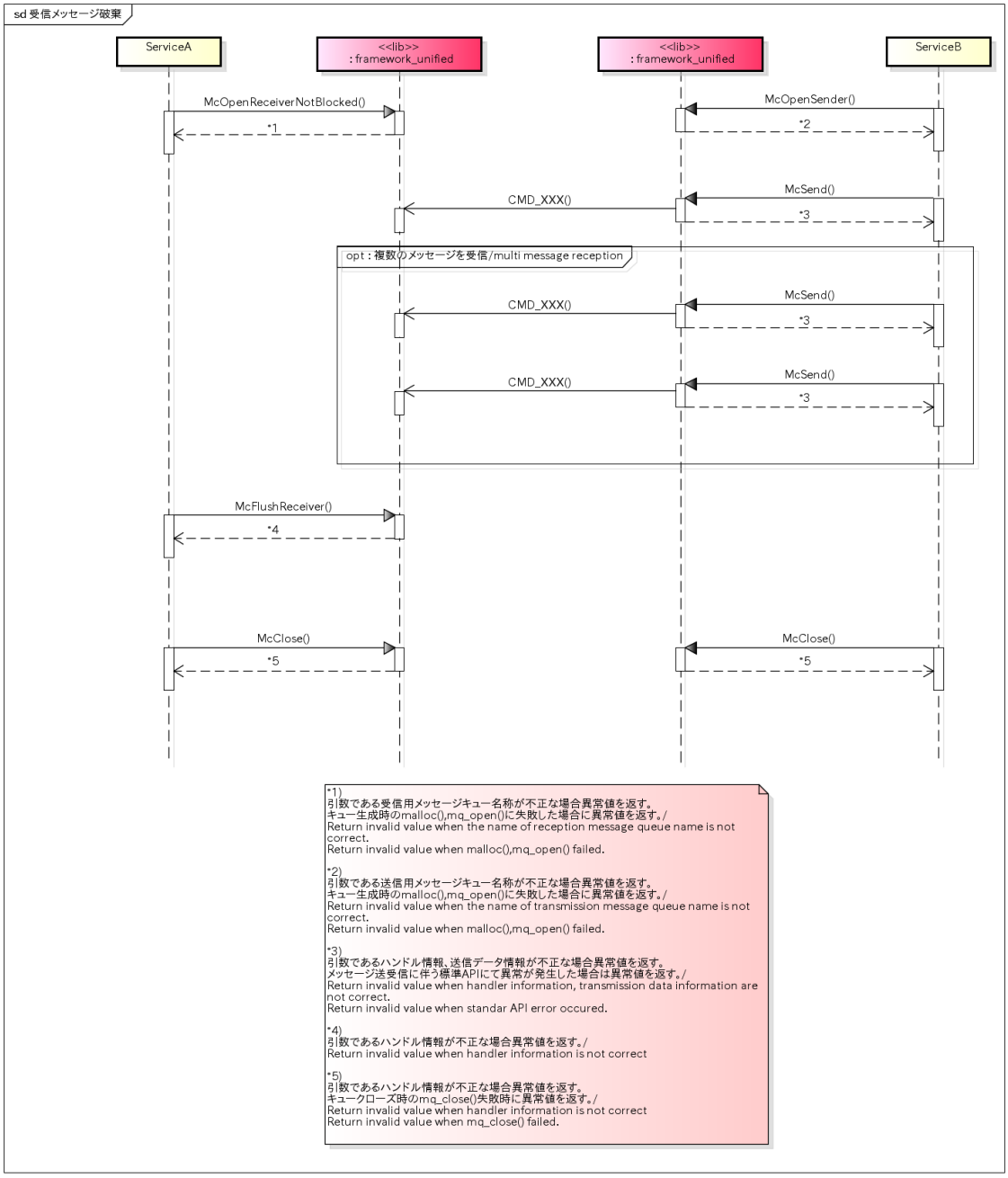
概要 *[Overview]*

メッセージキューに蓄積されているメッセージ情報を破棄するためのシーケンスを以下に記載する。

Sequence of deleting received message that is saved in message queue is as follows.

シーケンス *[Sequence]*

受信メッセージ破棄 */delete received message*



子スレッド起動_ループなし */start sub thread(no loop)*

概要 *[Overview]*

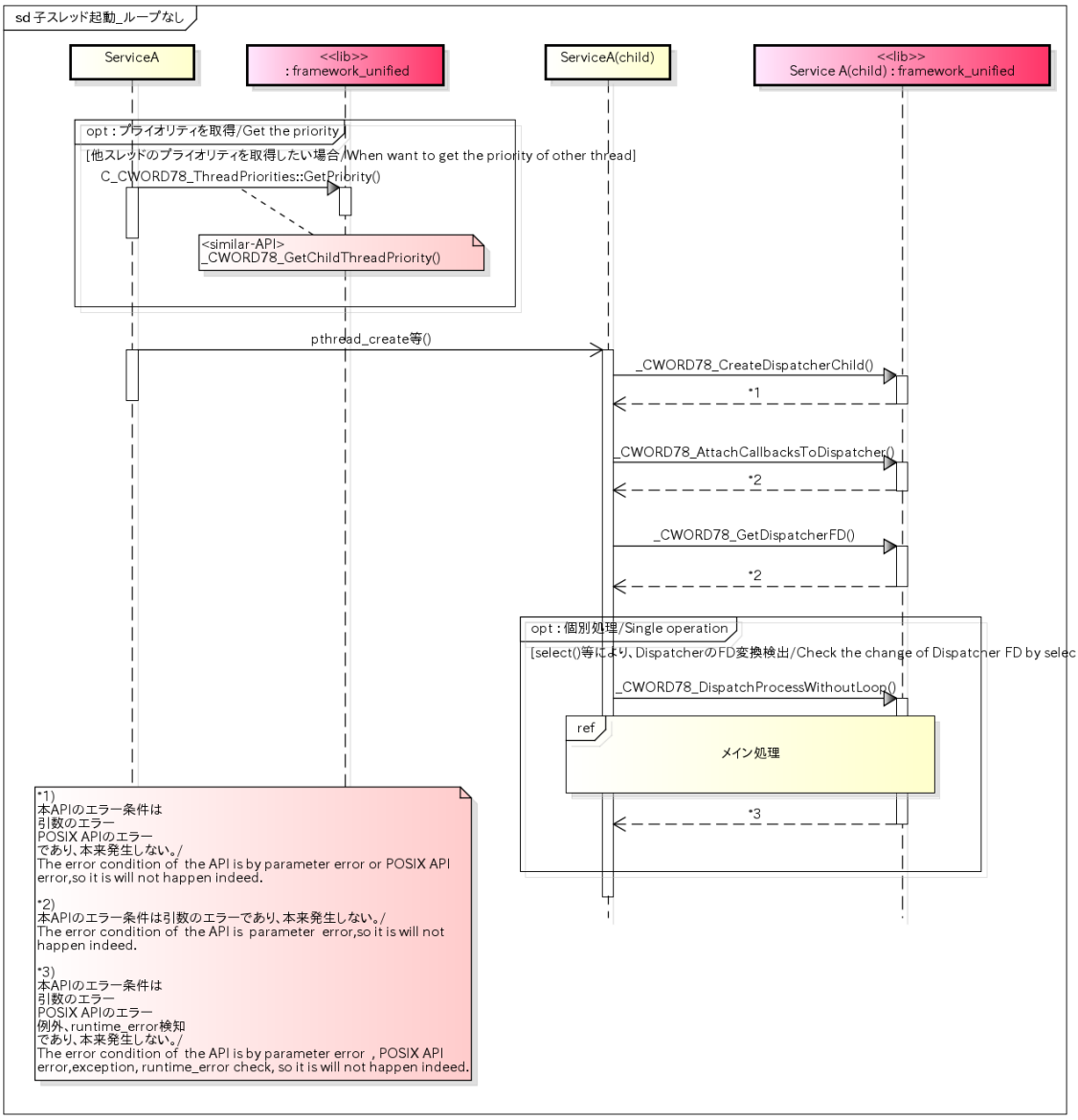
メインループを持たない子スレッドの起動シーケンスを以下に記載する。
メインループが必要な場合は自作および次項の子スレッド起動_ループありにて実現すること。

Sequence of starting sub thread that has no main loop is as follows.

If main loop is need,implement it with self made and next sequence [start sub thread_has loop]

シーケンス *[Sequence]*

子スレッド起動_ループなし */start sub thread(no loop)*



子スレッド起動_ループあり(子スレッド間通信) */start sub thread_has loop(sub thread communication)*

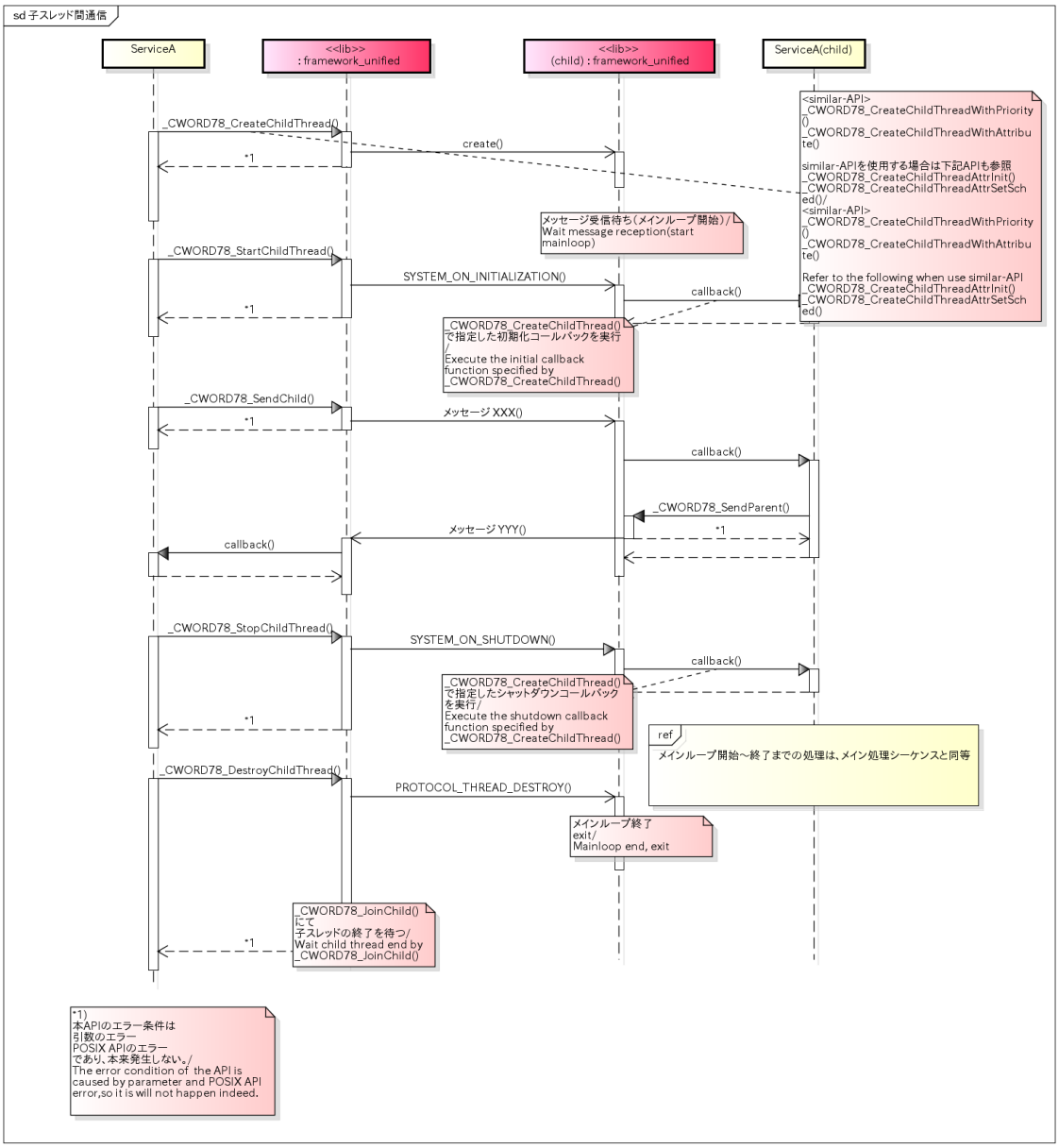
概要 *[Overview]*

メインループを持つ子スレッドの起動、および子スレッド、親スレッド間の通信シーケンスを以下に記載する。

Sequence of sub thread(has mainloop) start,communication between sub thread and parent thread is as follows.

シーケンス *[Sequence]*

子スレッド起動_ループあり(子スレッド間通信)*/start sub thread_has loop(sub thread communication)*



タイマー（コールバック） */timer(callback)*

概要 *[Overview]*

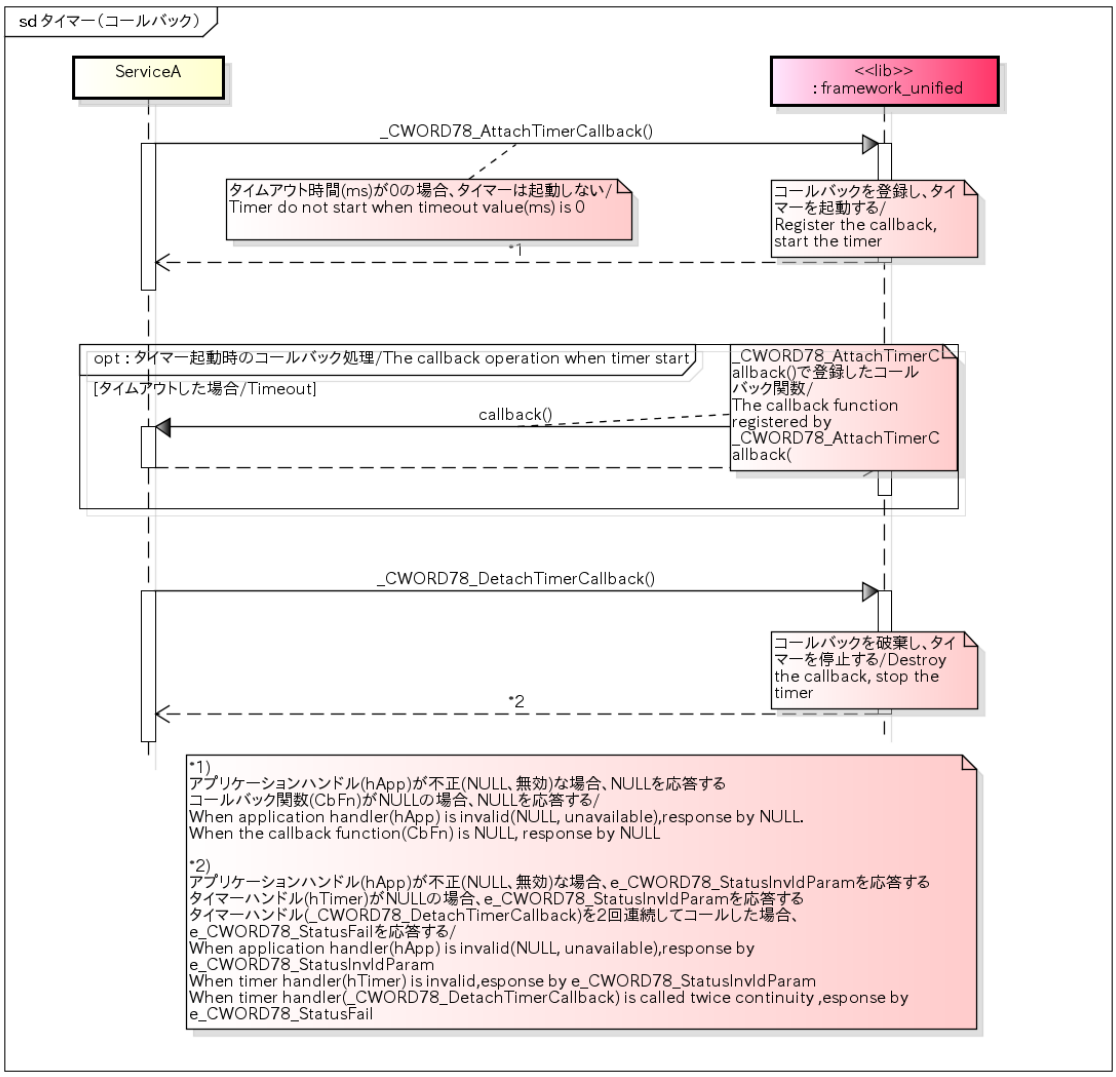
本処理はコールバック関数の登録とタイマーの開始処理が一体となったタイマーの機能を提供する。タイムアウトした際に対象APIで登録したコールバック関数が呼ばれる。

Provide function to timer which combines callback function register and timer start process.

When timeout happens,call the registered API object.

シーケンス *[Sequence]*

タイマー（コールバック） */timer(callback)*



タイマー（API） */Timer(API)*

概要 *[Overview]*

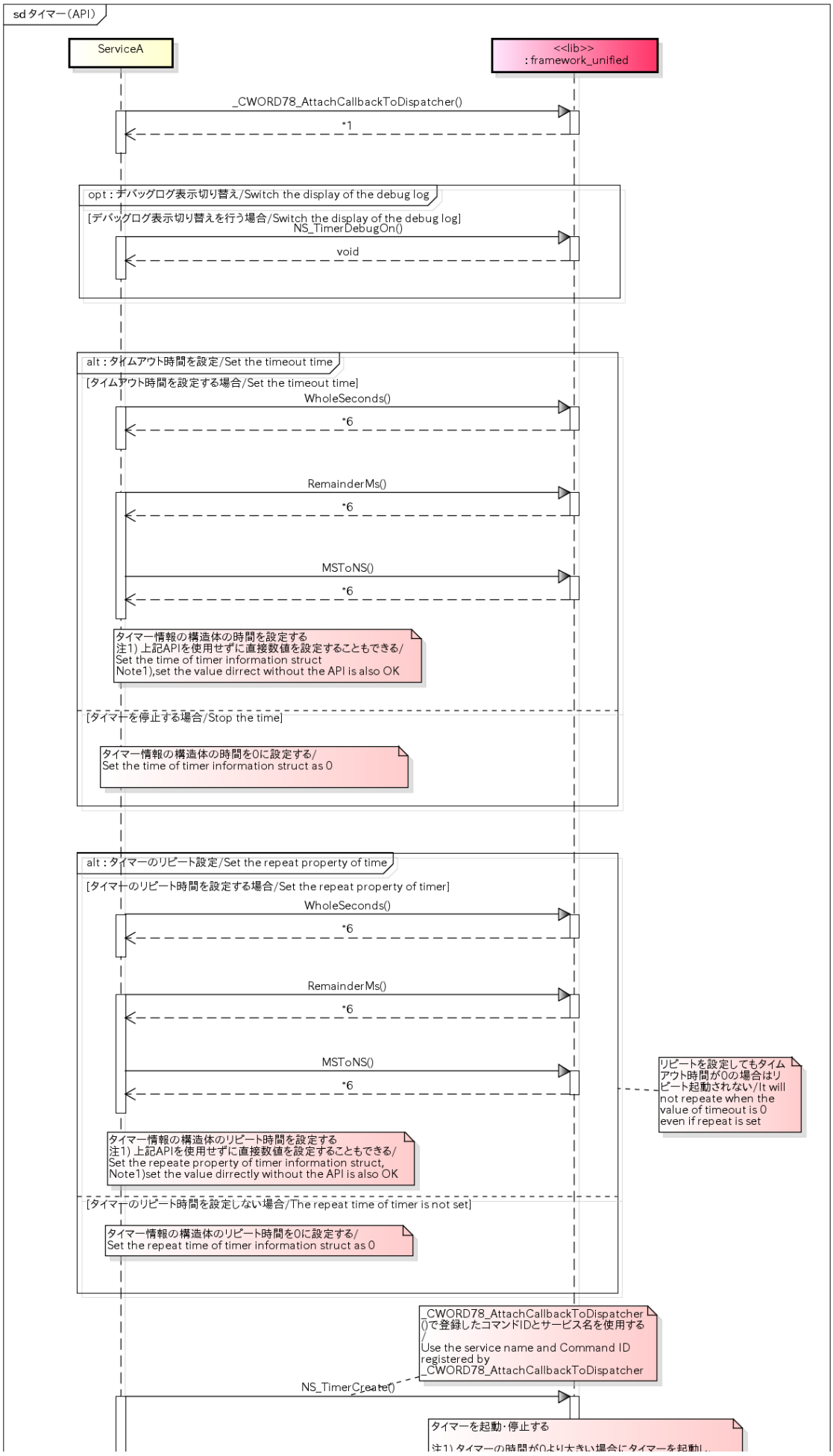
本処理はタイマーの基本的な機能を提供する。本処理を使用するには事前にコールバック関数を登録する必要があり、タイムアウトした際に対象APIで登録したコールバック関数が呼ばれる。

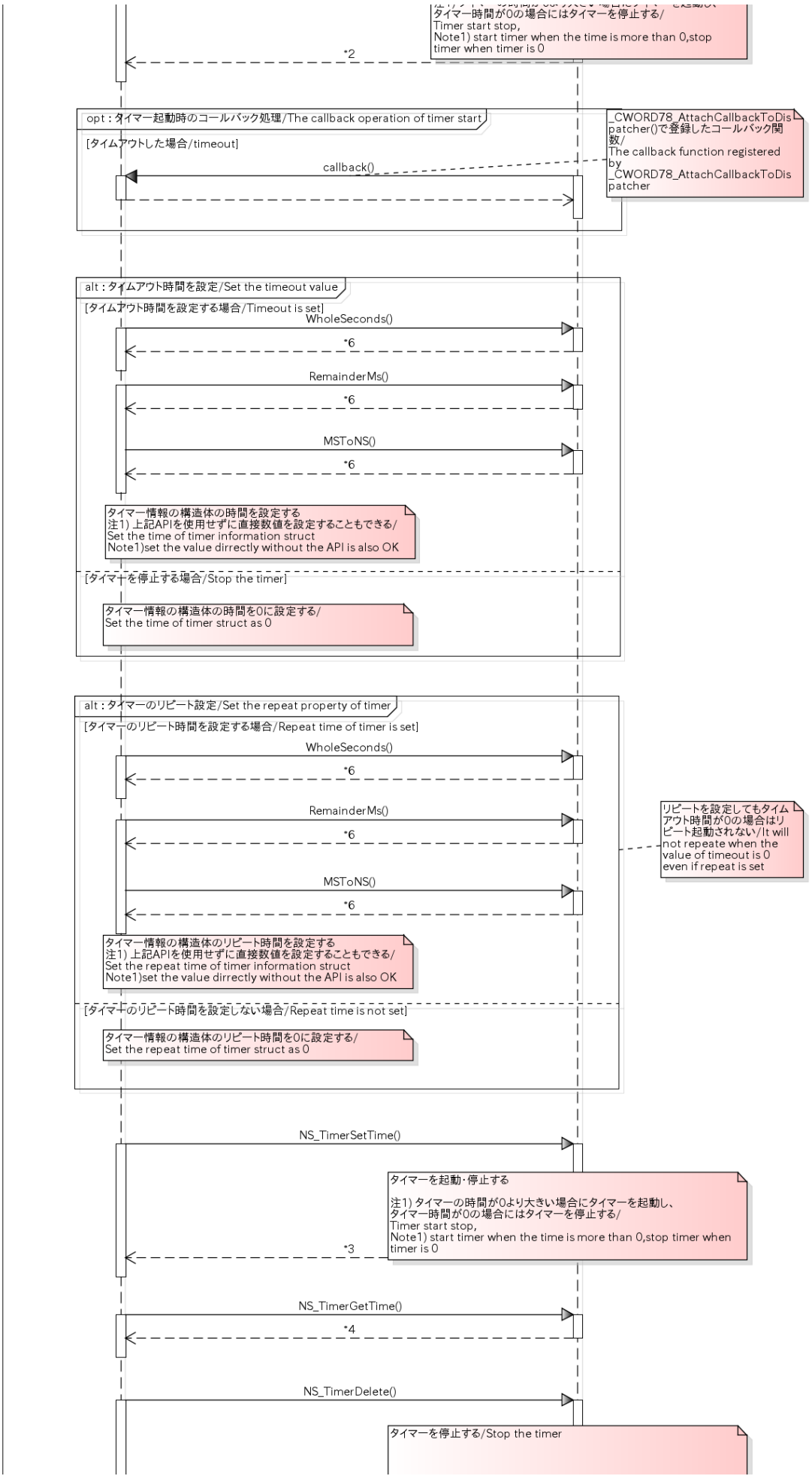
Provide basic function of timer handle.

Before using this API,callback function should be registered.When timeout happens,the registered API object will be called

シーケンス *[Sequence]*

タイマー（API） */Timer(API)*





タイマー（NSTimerクラス） */timer(NSTimer class)*

概要 *[Overview]*

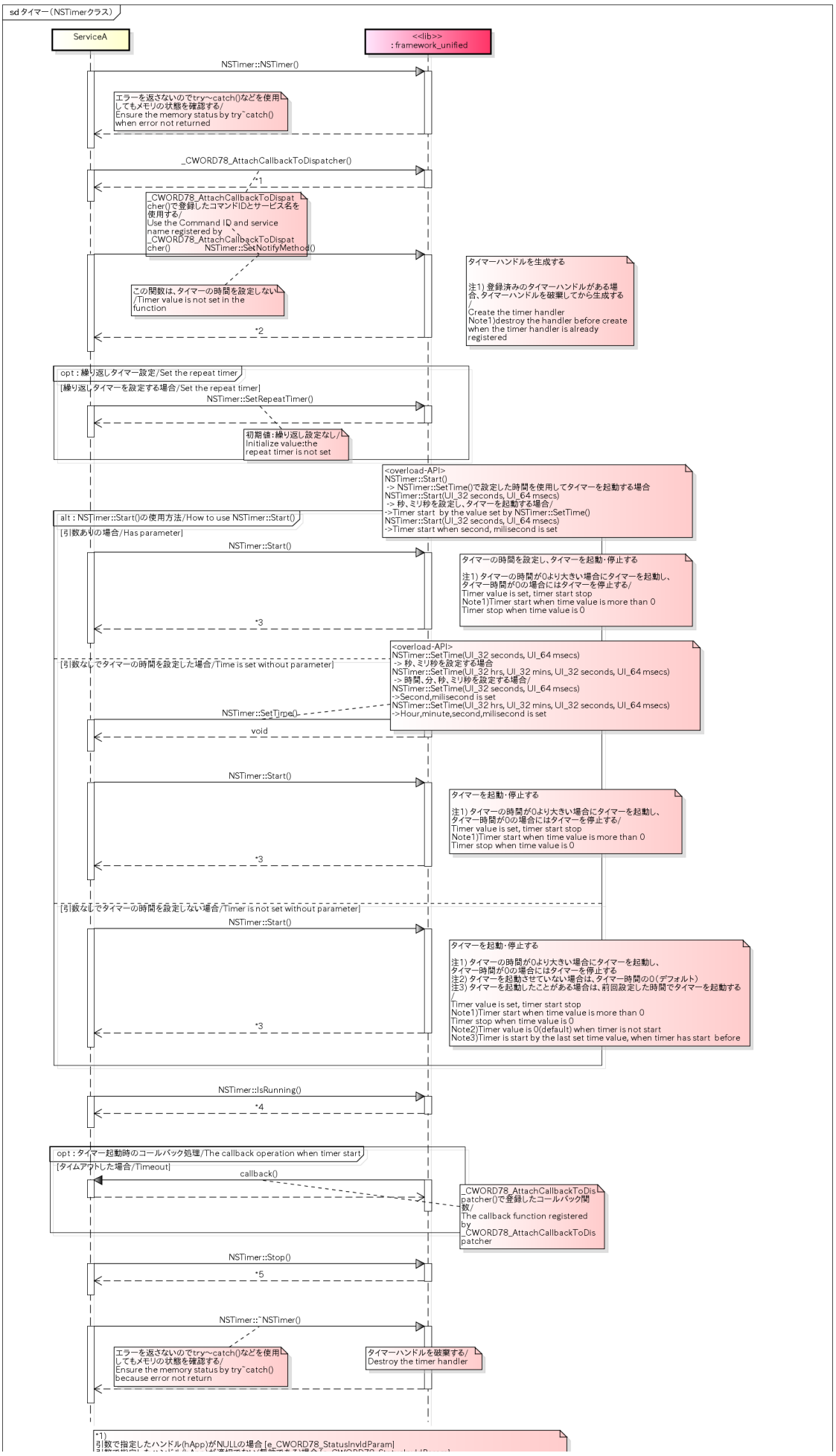
本処理はクラスを用いたタイマーの機能を提供する。本処理を使用するには事前にコールバック関数を登録する必要があり、タイムアウトした際に対象APIで登録したコールバック関数が呼ばれる。

Provide function of Timer class.

Before using this API,callback function should be registered.When timeout happens,the registered API object will be called

シーケンス *[Sequence]*

タイマー（NSTimerクラス） */timer(NSTimer class)*



RingBufferの読込み/書込み */read/write ringBuffer*

概要 *[Overview]*

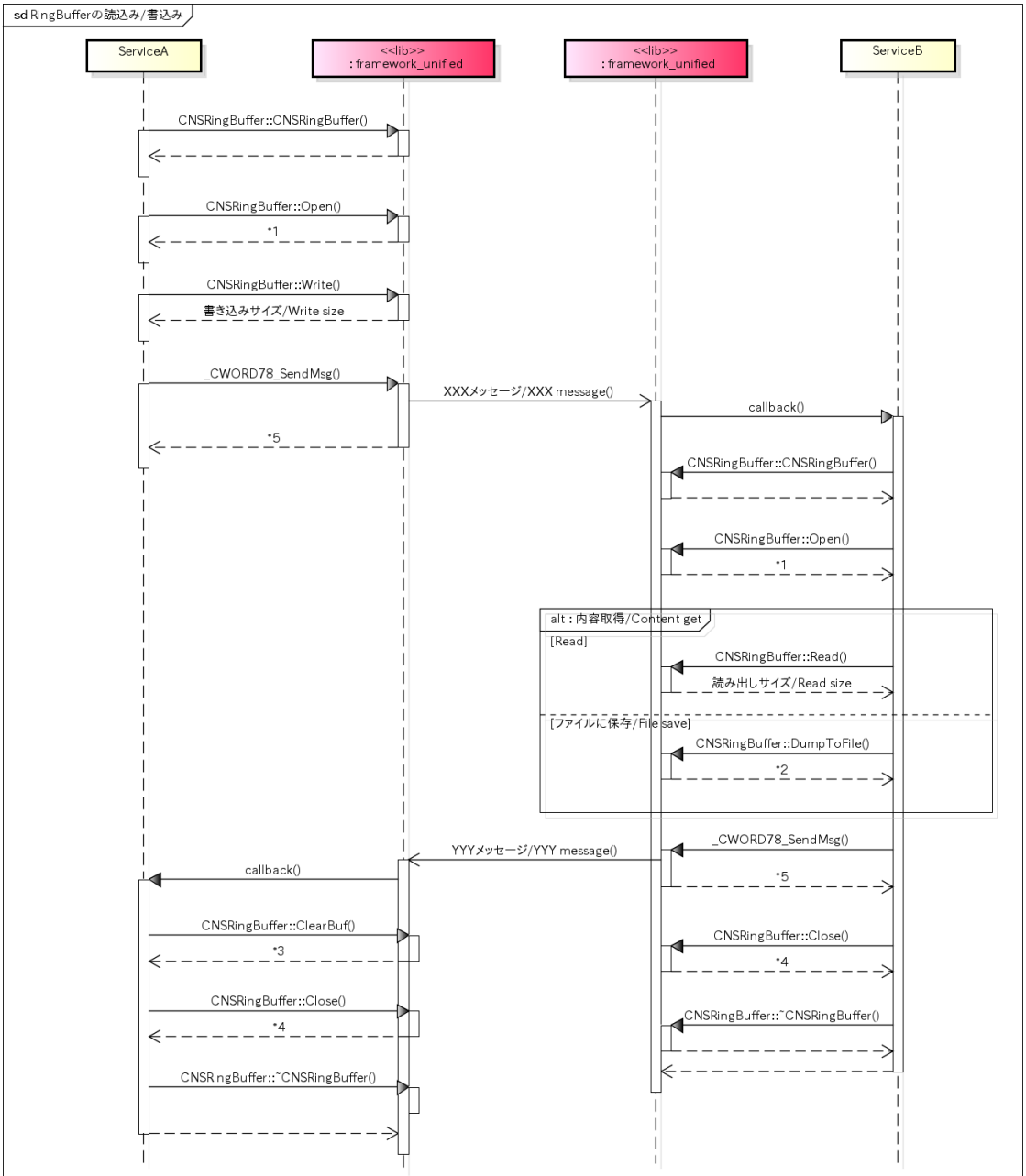
RingBufferの読み込み／書き込みシーケンスを以下に記載する。
本シーケンスは、書き込み完了および読み込み完了をメッセージ通信にて通知するケースである。

Sequence of ringbuffer read/write is as follows.

This sequence described the case of notifying the completion pf reading and writing by message.

シーケンス *[Sequence]*

RingBufferの読込み/書込み */read/write ringBuffer*



Configuration Fileの読込み/書込み */read/write configuration file*

概要 *[Overview]*

Configurationは、設定ファイルの読み込み及び更新機能を提供する。Configurationには、以下の機能用APIが提供されている。

1.コンフィギュレーションファイルの解析
2.コンフィギュレーションファイルから値の読込み
3.コンフィギュレーションファイルへの値の書込み

Configuration provides function of reading configurationfile and update function.Functions of Configuration are as follows.

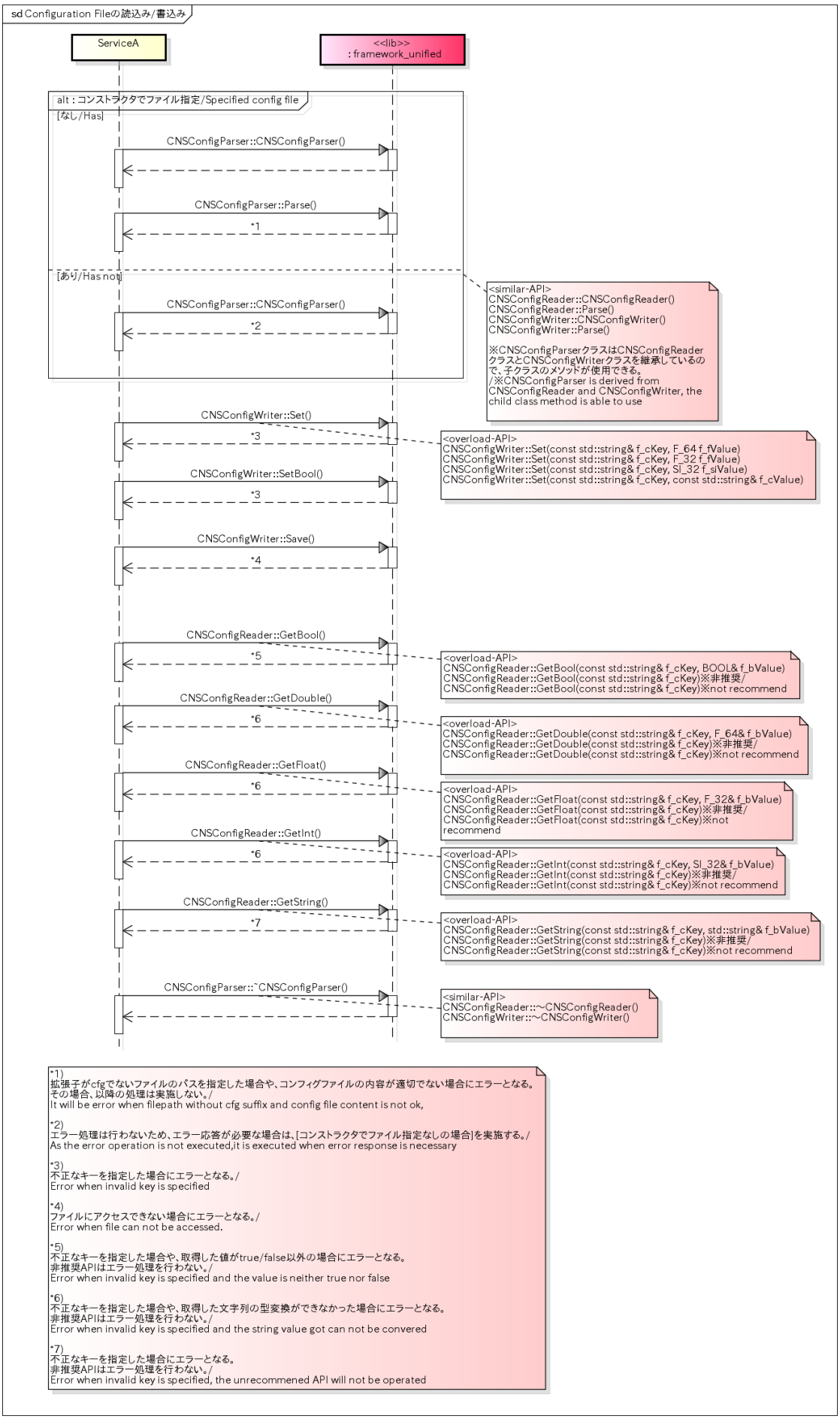
1.analysis configuration file

2.read value from configuration file

3.write value to configuration file

シーケンス *[Sequence]*

Configuration Fileの読込み/書込み */read/write configuration file*



初期化 */Initialize*

概要 *[Overview]*

アプリ起動時に_CWORD78__SET_ZONES()をコールすることでログ機能の初期化を行う。

_CWORD78__SET_ZONES()は 「ns_logger__CWORD78_log_dcu.cfg」または「ns_logger__CWORD78_log_meu.cfg」に定義されているデータを元にアプリ 毎の初期化を行った後、リングバッファとメモリの設定等を行う。

_CWORD78__SET_ZONES()とアプリ名を設定するNsLogSetProcessName()はどちらを先にコールしても問題はない。

ログの出力方法を変更する必要がない場合はロギング方法の変更をコールする必要はない。また、ロギング方法変更用のAPIを一度でもコールした場合には再度コールするまで設定は変更されない。

Call _CWORD78__SET_ZONES() to initialize log function when application starts.

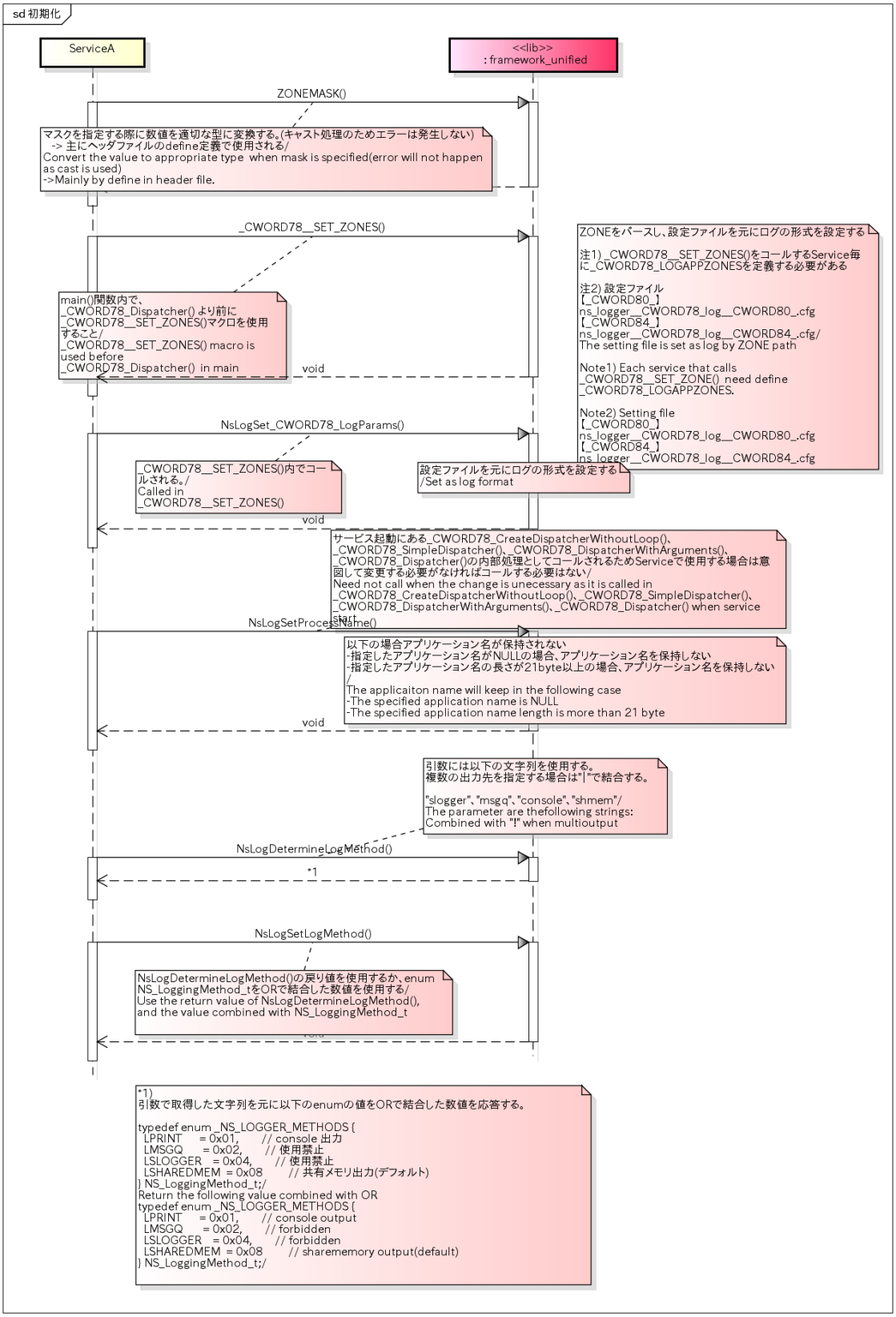
_CWORD78__SET_ZONES() initializes the data that be defined in 「ns_logger__CWORD78_log_dcu.cfg」 and 「ns_logger__CWORD78_log_meu.cfg」,and sets ring buffer and memory.

There is no problem whether call _CWORD78__SET_ZONES() first or call NsLogSetProcessName() first,which is used to set application name.

If users do not want to change log ouput way, there is no need to call the logging method.Because only if loggging change method is called again,the log configuration will not be changge.

シーケンス *[Sequence]*

初期化 */Initialize*



ログ出力 */log output*

概要 *[Overview]*

ログを出力する際には_CWORD78_LOG()もしくはNsLogData()を使用する。

※本処理を実行する前に_CWORD78__SET_ZONES()をコールしておく必要がある。

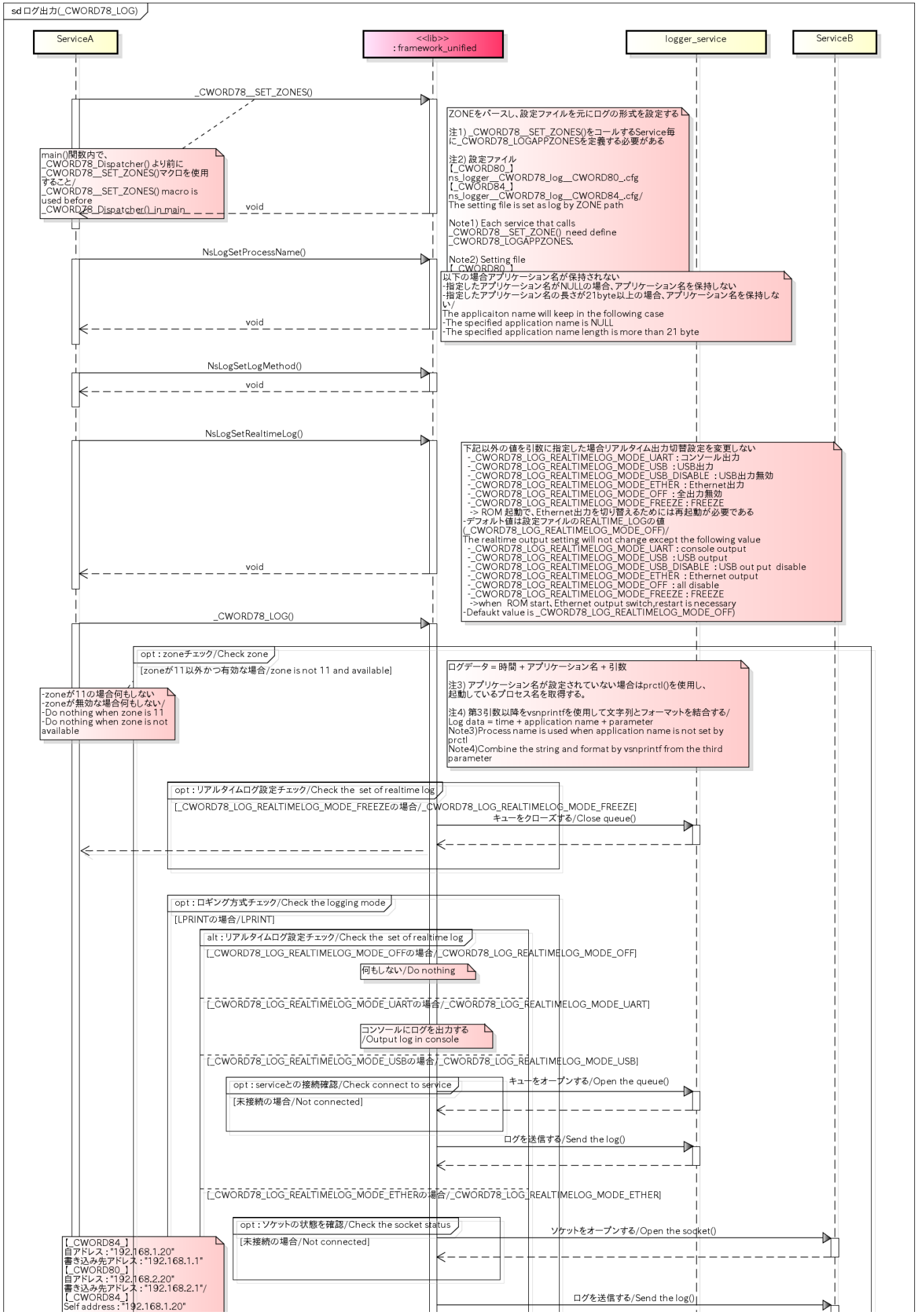
また、_CWORD78_LOG()の代わりにNsLog()を使用してログを出力することも可能。

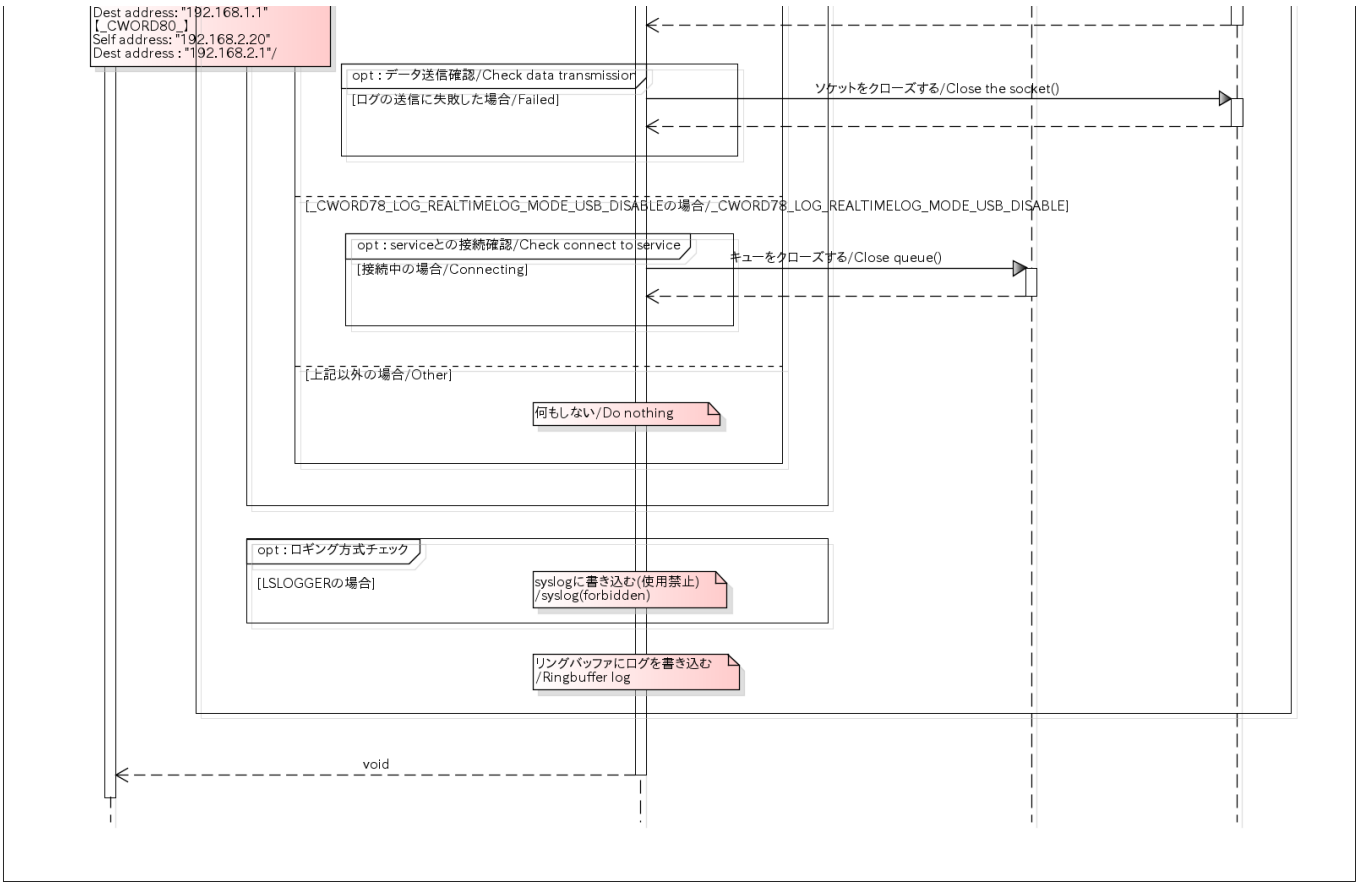
Use _CWORD78_LOG() or NsLogData() when output log.

_CWORD78__SET_ZONES() should be called before this action. NsLog() can also be used to output log instead of _CWORD78_LOG().

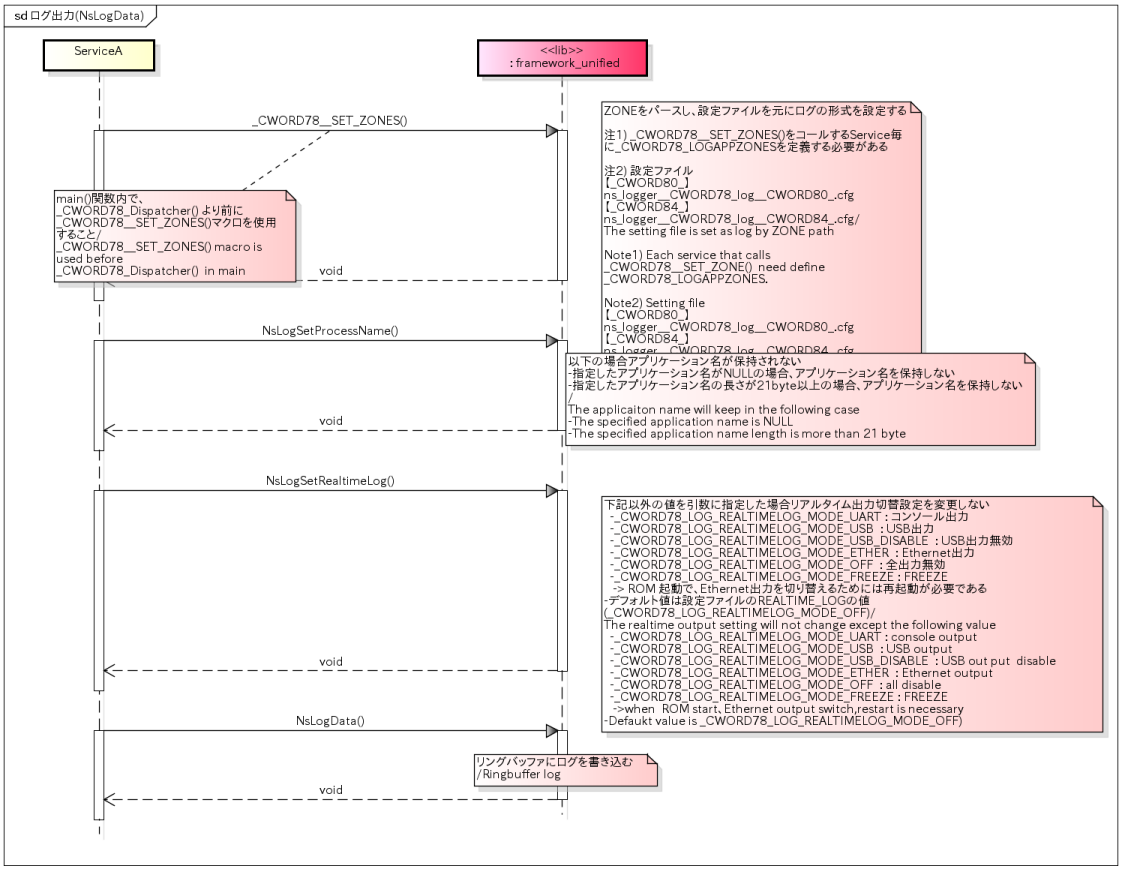
シーケンス *[Sequence]*

ログ出力(_CWORD78_LOG) */log output(_CWORD78_LOG)*

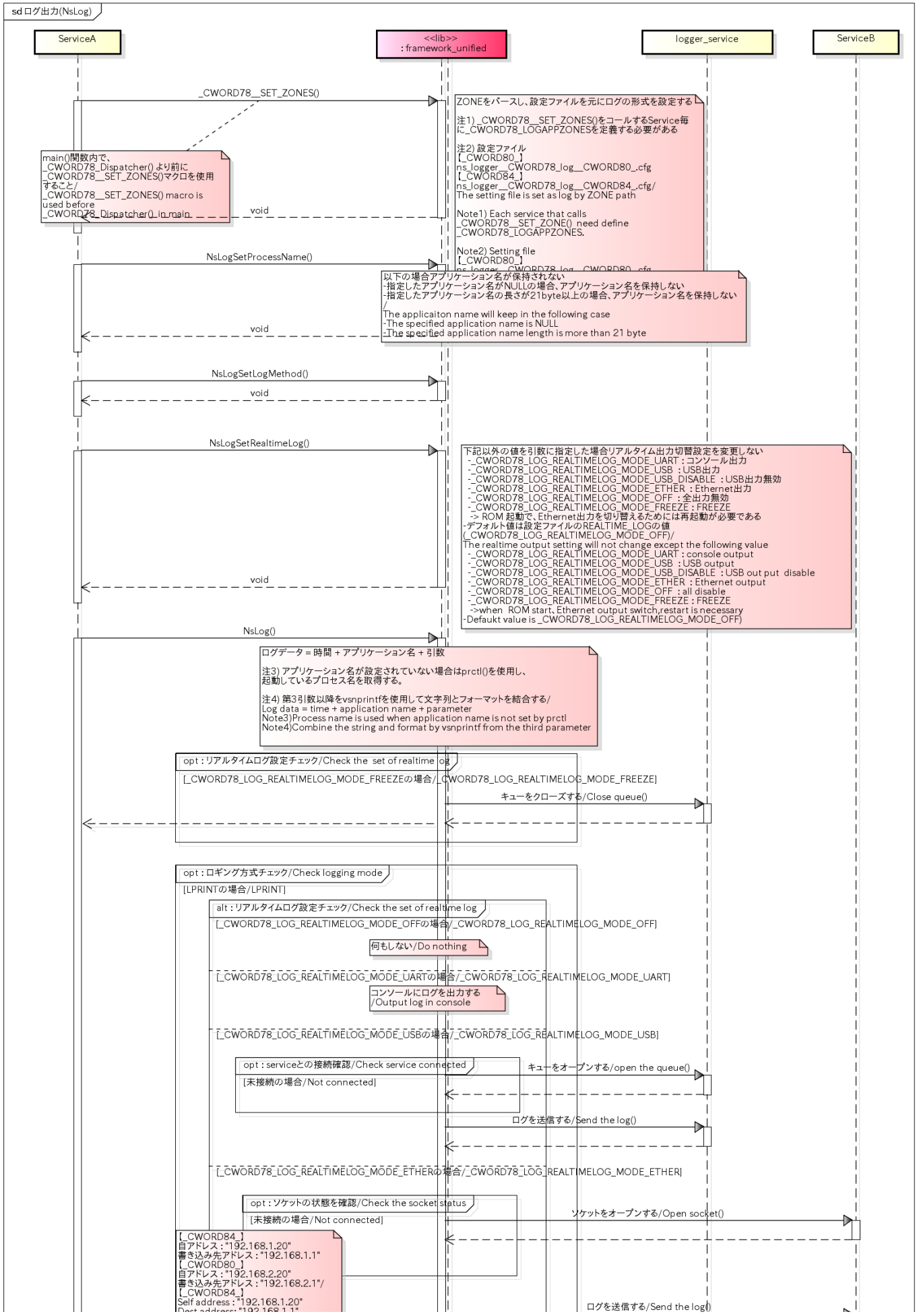


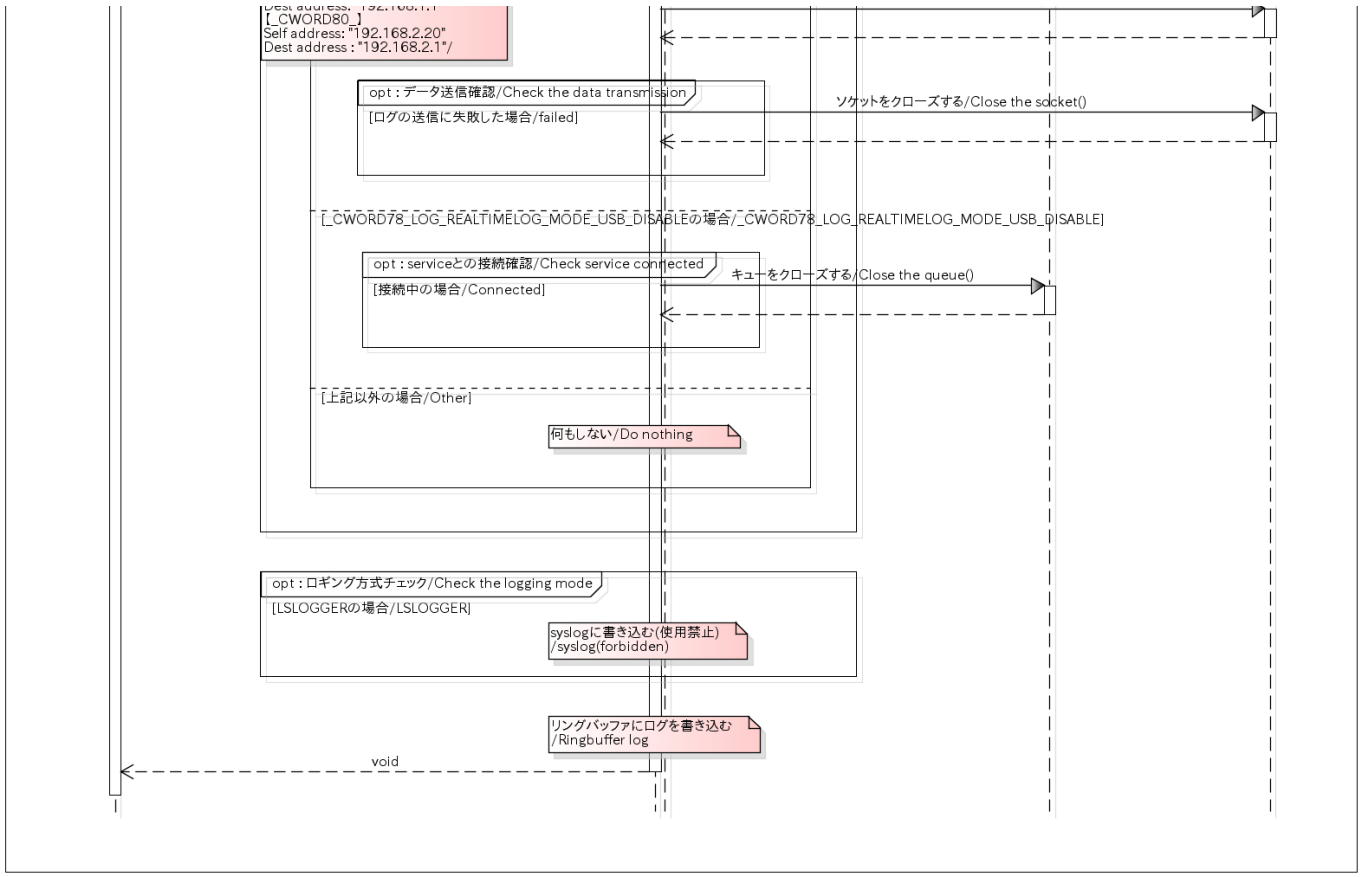


ログ出力(NsLogData) */log output(NsLogData)*

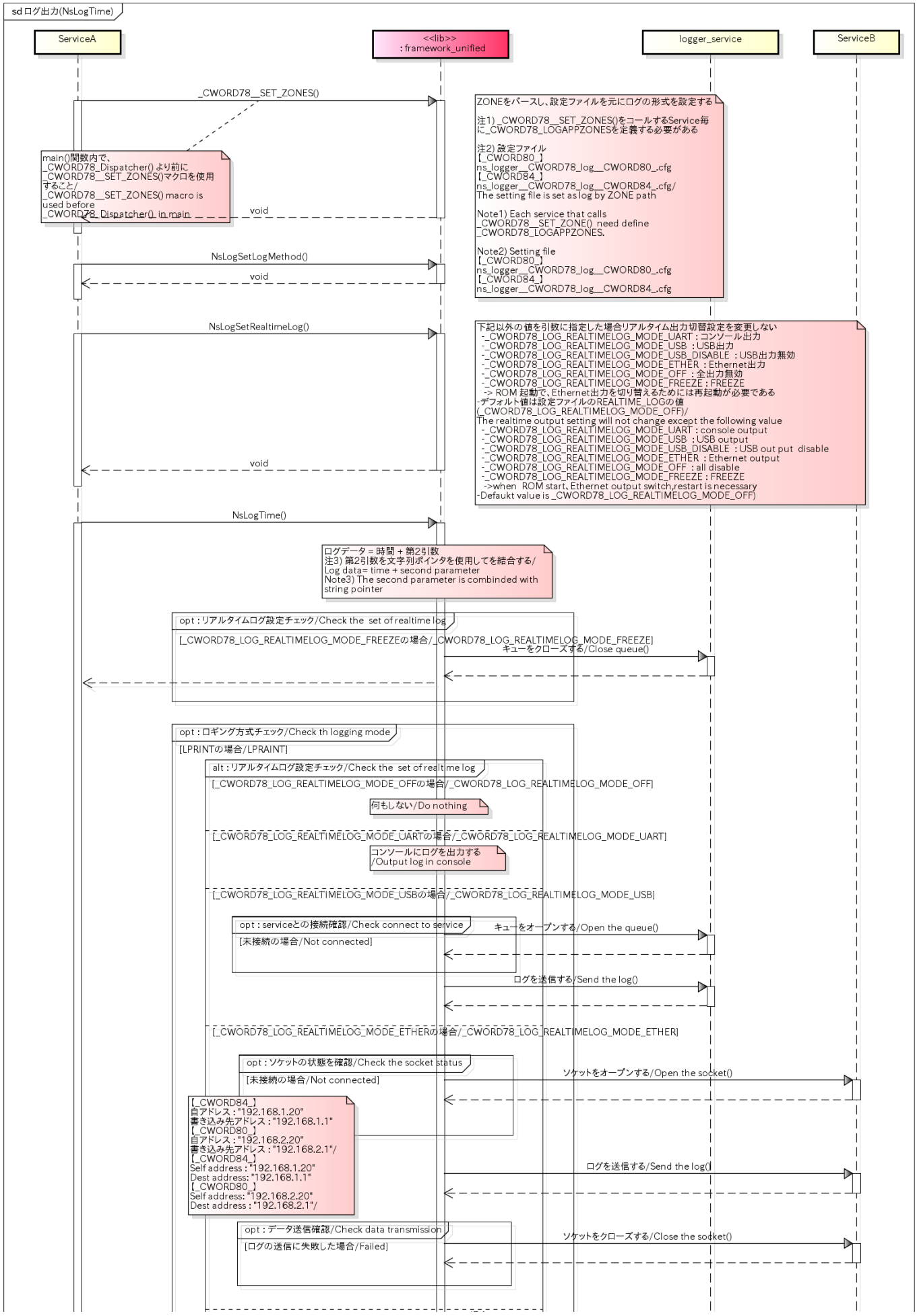


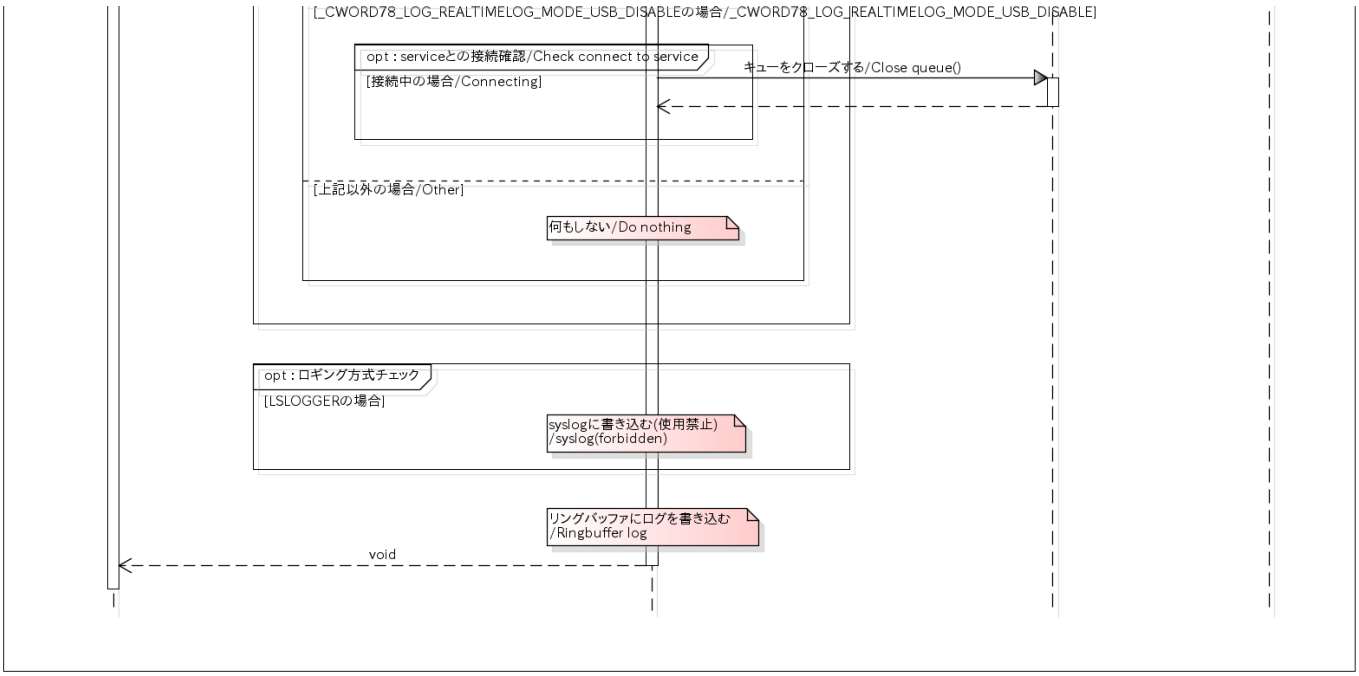
ログ出力(NsLog) */log output(NsLog)*



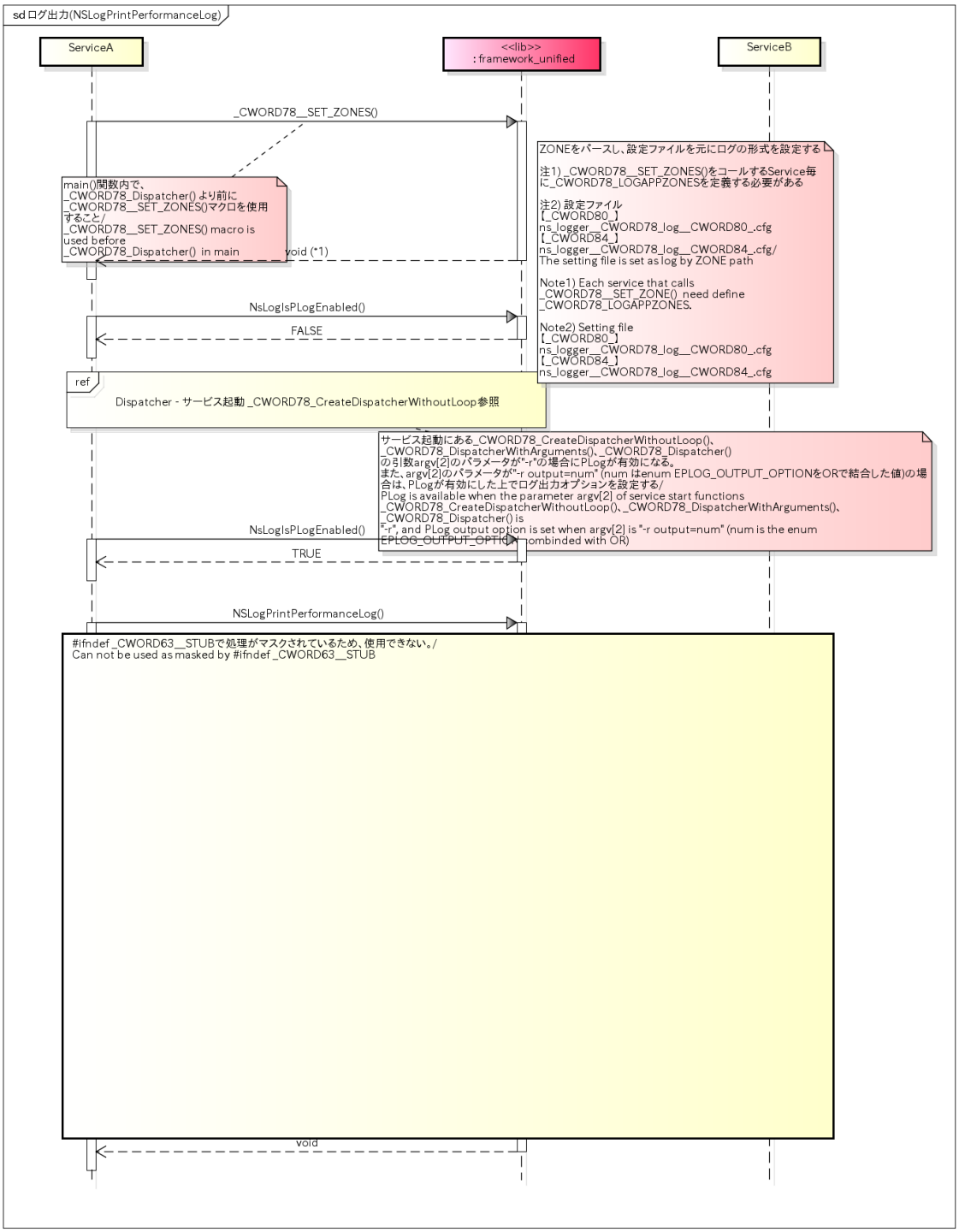


ログ出力(NsLogTime) ***/log output(NsLogTime)***

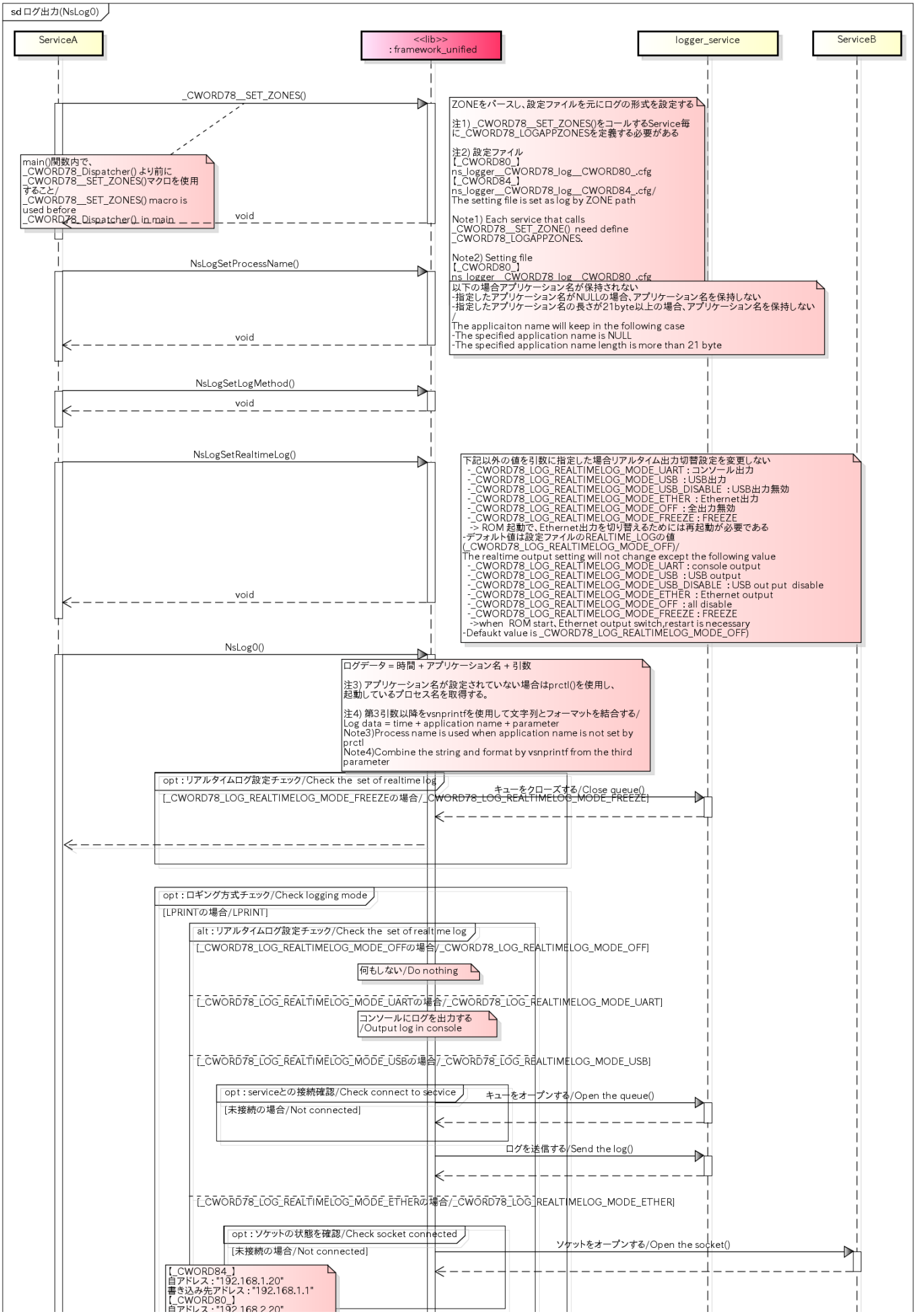


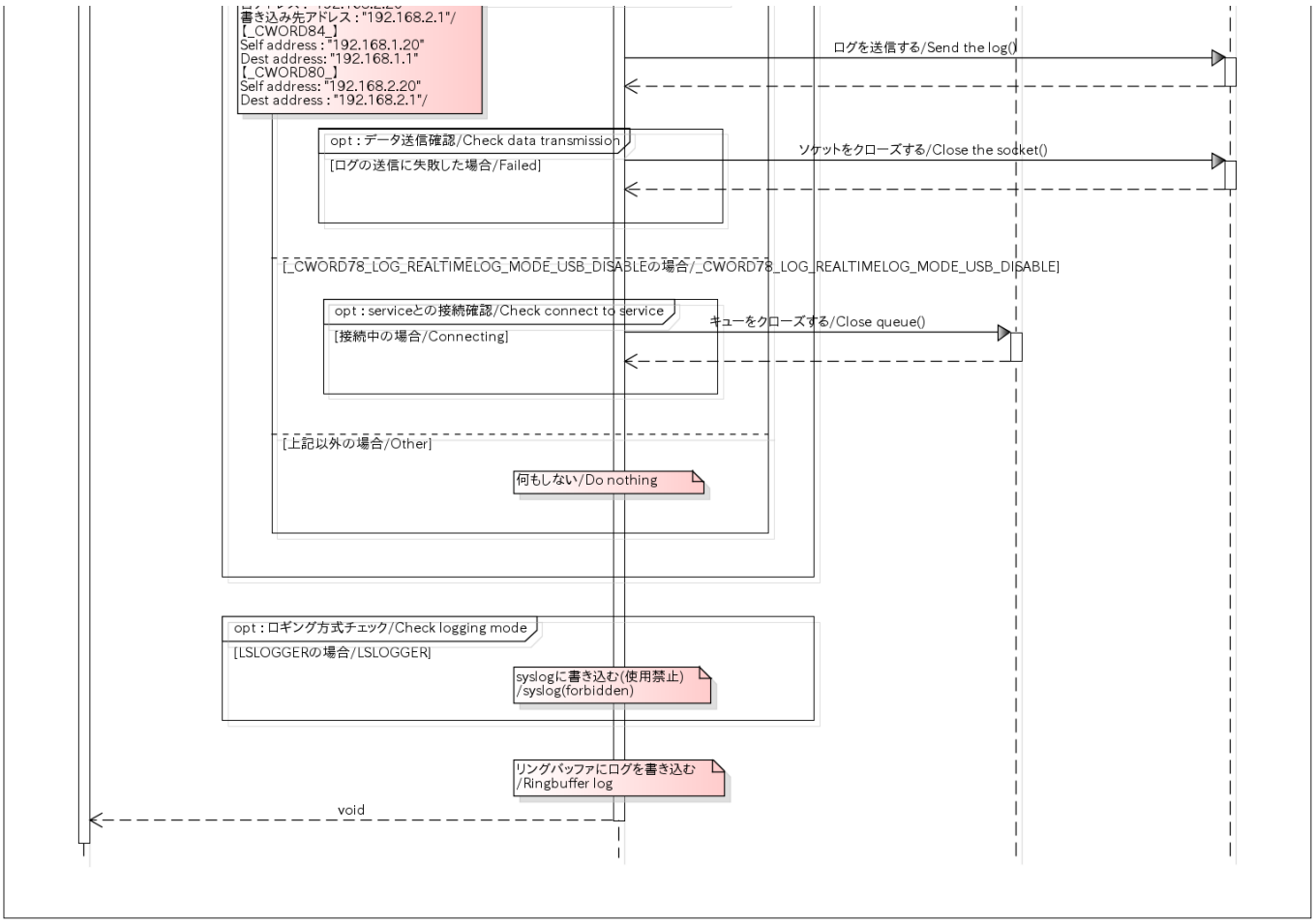


ログ出力(NSLogPrintPerformanceLog) ***/log output(NSLogPrintPerformanceLog)***



ログ出力(NsLog0) */log output(NsLog0)*





強制クリア */ force clear*

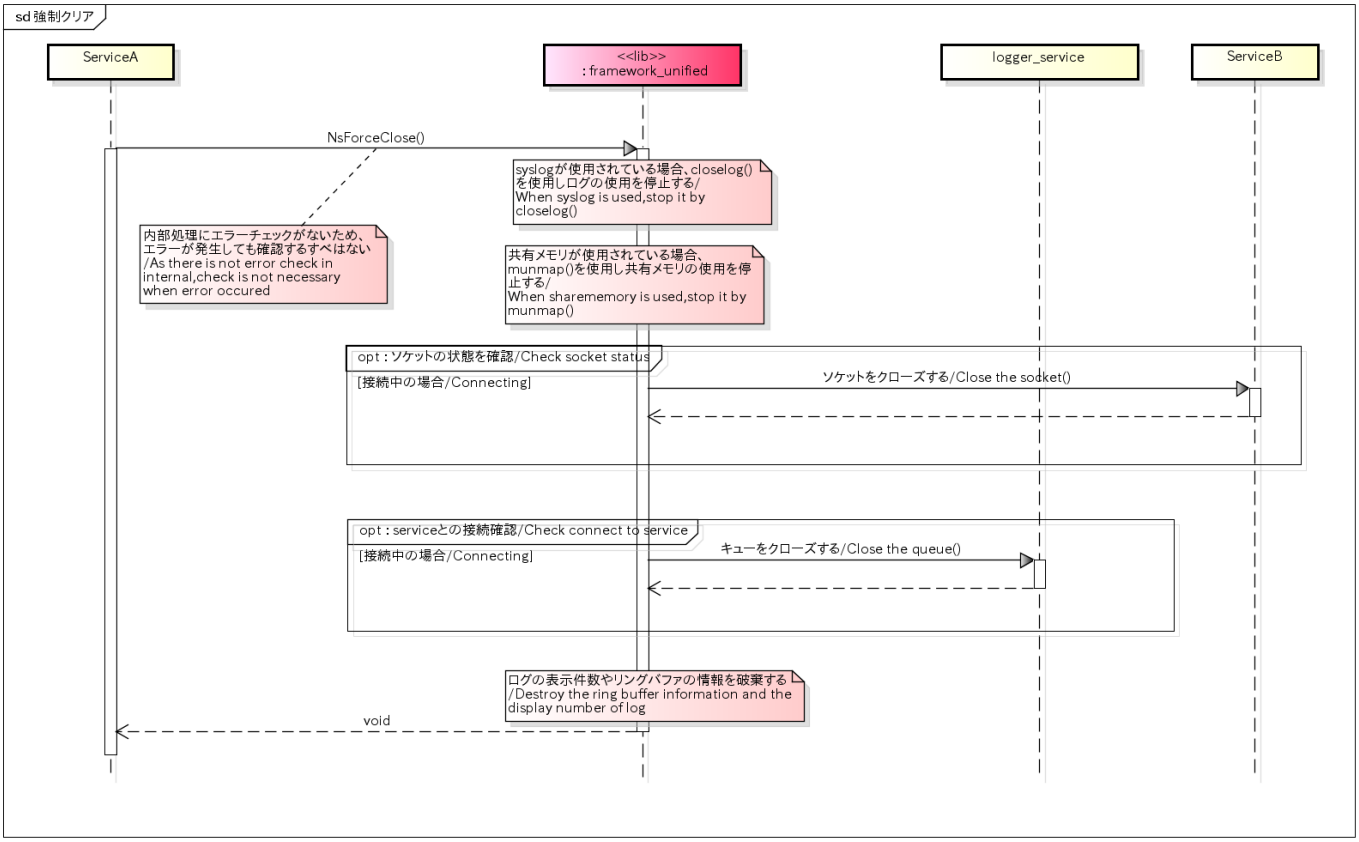
概要 *[Overview]*

ログの設定やメモリ上のログをクリアする際に使用する。

Clear log configuration or log that be saved.

シーケンス *[Sequence]*

強制クリア */ force clear*



ログ出力の制御ワード（設定・取得） */control word of log output(set・get)*

概要 *[Overview]*

シーケンスの順番でコールすることは必須ではない。制御ワードを設定、取得する際に使用するAPIである。

※ログを出力する際に使用する_CWORD78_LOG()内部でもZONEMASK()とIS_ZONE_SET()がコールされている。

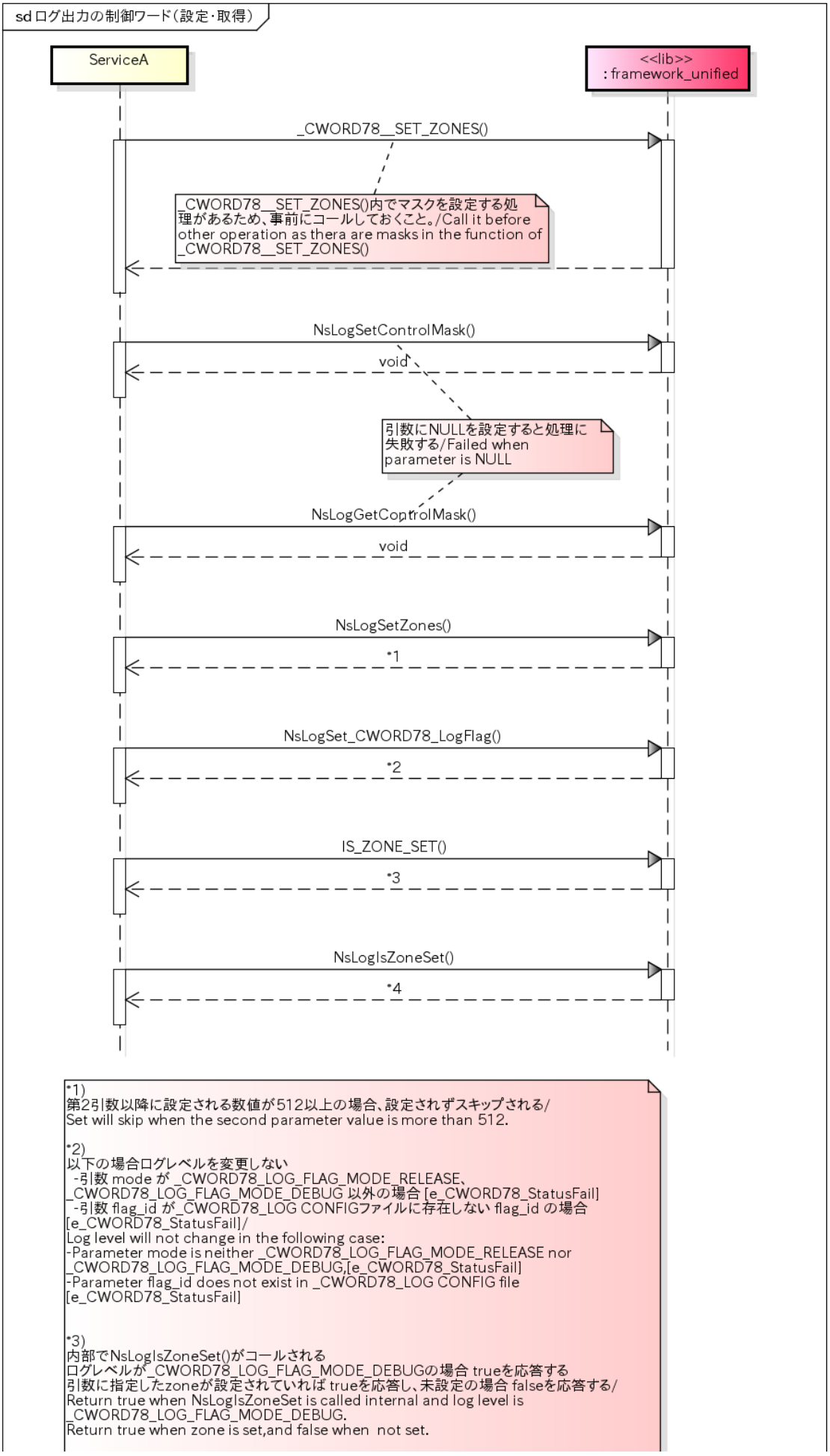
Do not need call the APIs by the order as the sequence describes.

This API is used to set/get control word

※ZONEMASK() and IS_ZONE_SET() is called even if inside _CWORD78_LOG().

シーケンス *[Sequence]*

ログ出力の制御ワード（設定・取得） */control word of log output(set・get)*



*4)
ログレベルが_CWORD78_LOG_FLAG_MODE_DEBUGの場合 trueを応答する
引数に指定したzoneが設定されていれば trueを応答し、未設定の場合 falseを応答する/
Return true when log level is set as _CWORD78_LOG_FLAG_MODE_DEBUG.
Return true when zone is set,and false when not set

ログファイル操作 */logfile handle*

概要 *[Overview]*

ログファイルの数や名前を設定する際に使用する。

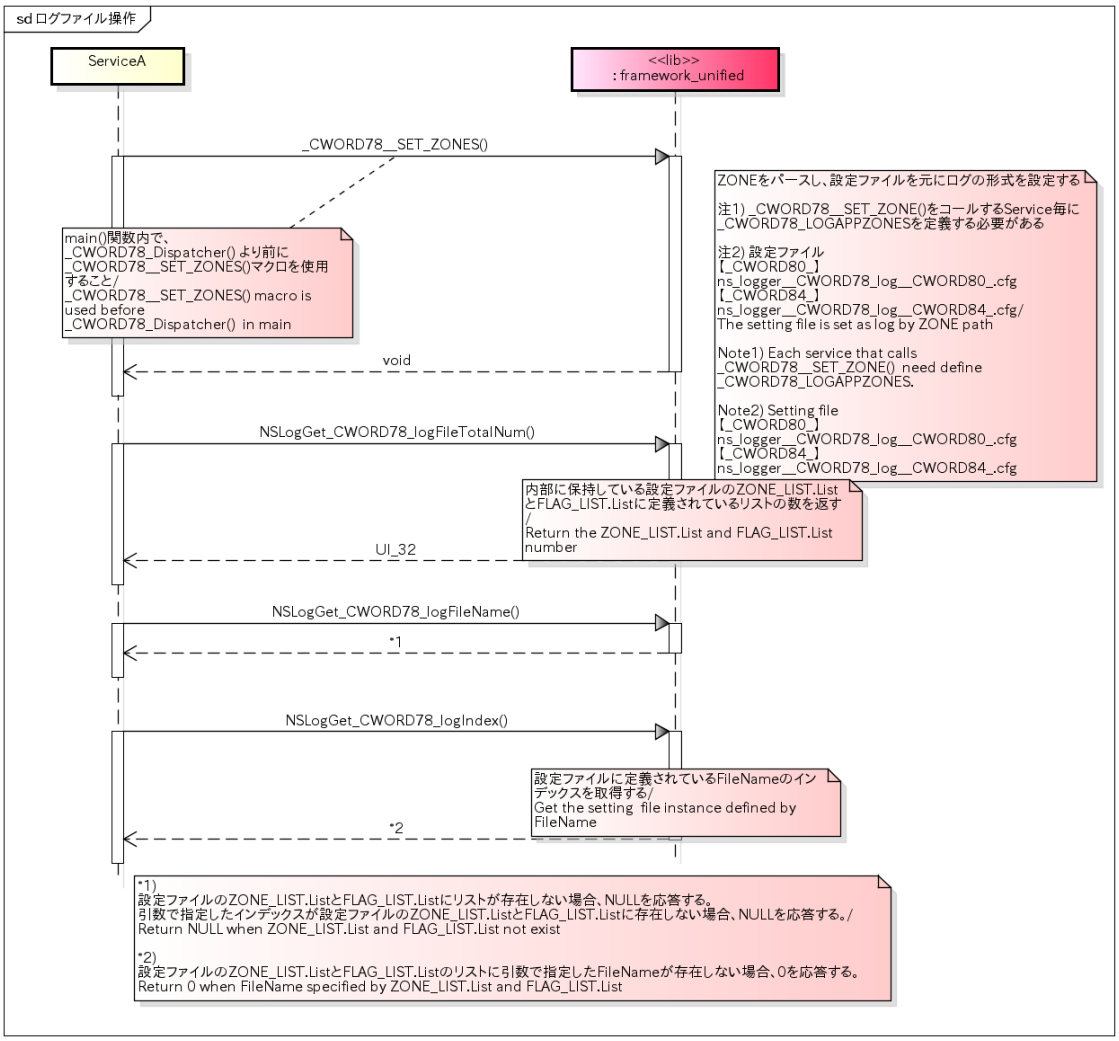
※本処理を実行する前に_CWORD78__SET_ZONES()をコールしておく必要がある。

Set log file number and name.

※_CWORD78__SET_ZONES() should be called before this action.

シーケンス *[Sequence]*

ログファイル操作 */logfile handle*



LogLevel（設定・取得） */LogLevel(Set・Get)*

概要 *[Overview]*

ログレベルを切り替える際に使用する。

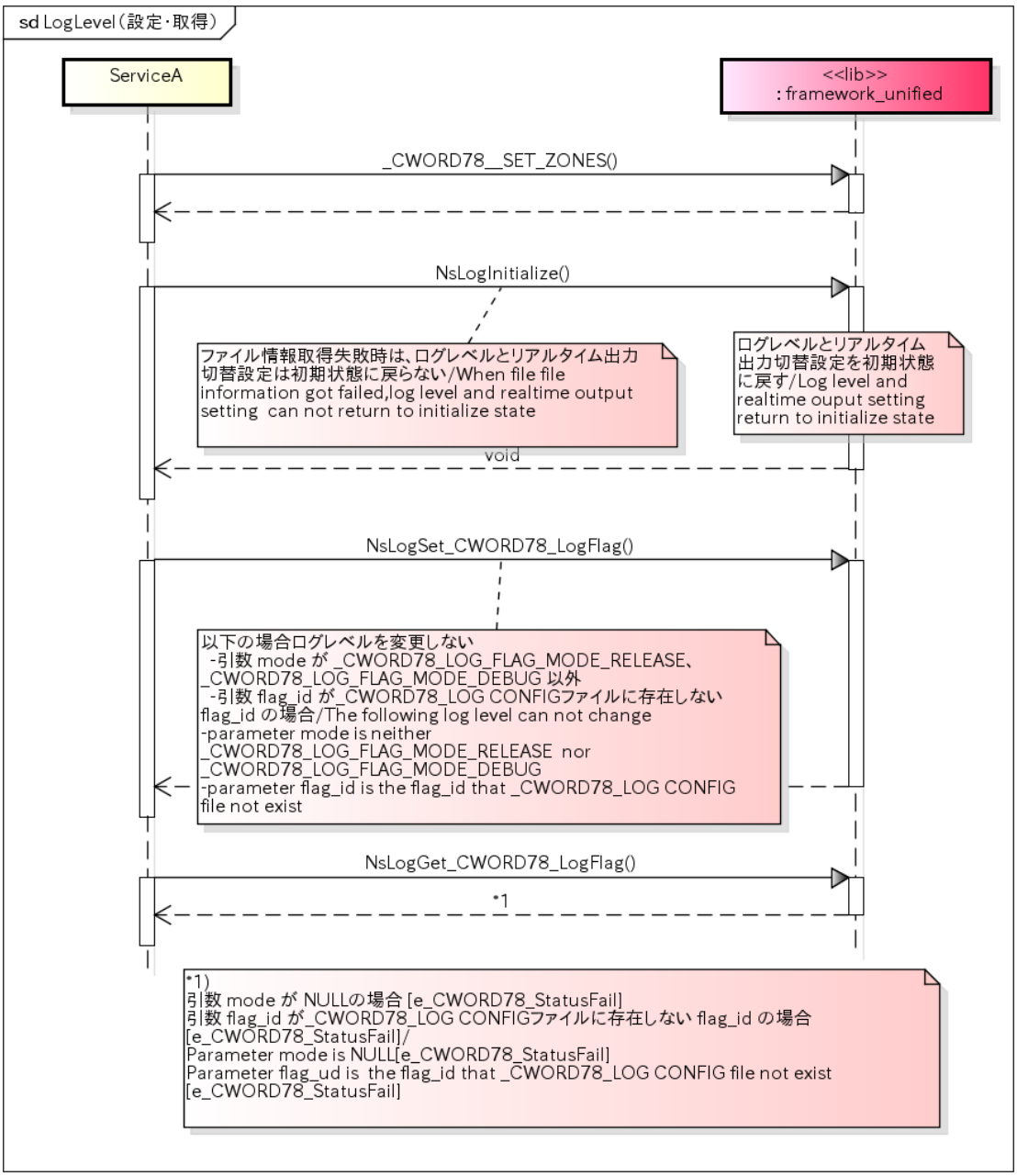
※本処理を実行する前に_CWORD78__SET_ZONES()をコールしておく必要がある。

Change log level.

※_CWORD78__SET_ZONES() should be called before this action.

シーケンス *[Sequence]*

LogLevel（設定・取得） */LogLevel(Set・Get)*



リアルタイムログ出力切替設定 */Change to realtime log output*

概要 *[Overview]*

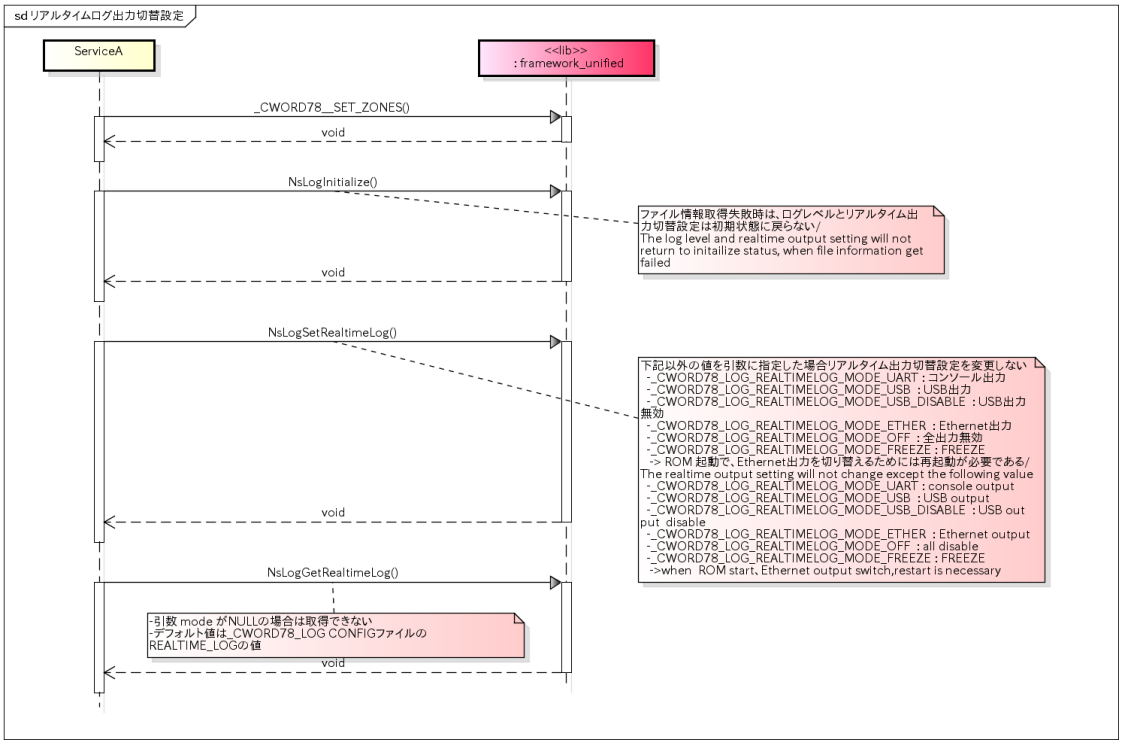
リアルタイムログの出力を切り替える際に使用する。

※本処理を実行する前に_CWORD78__SET_ZONES()をコールしておく必要がある。

Change to realtime log output.

シーケンス [Sequence]

リアルタイムログ出力切替設定



ログイベント送信 */send log event*

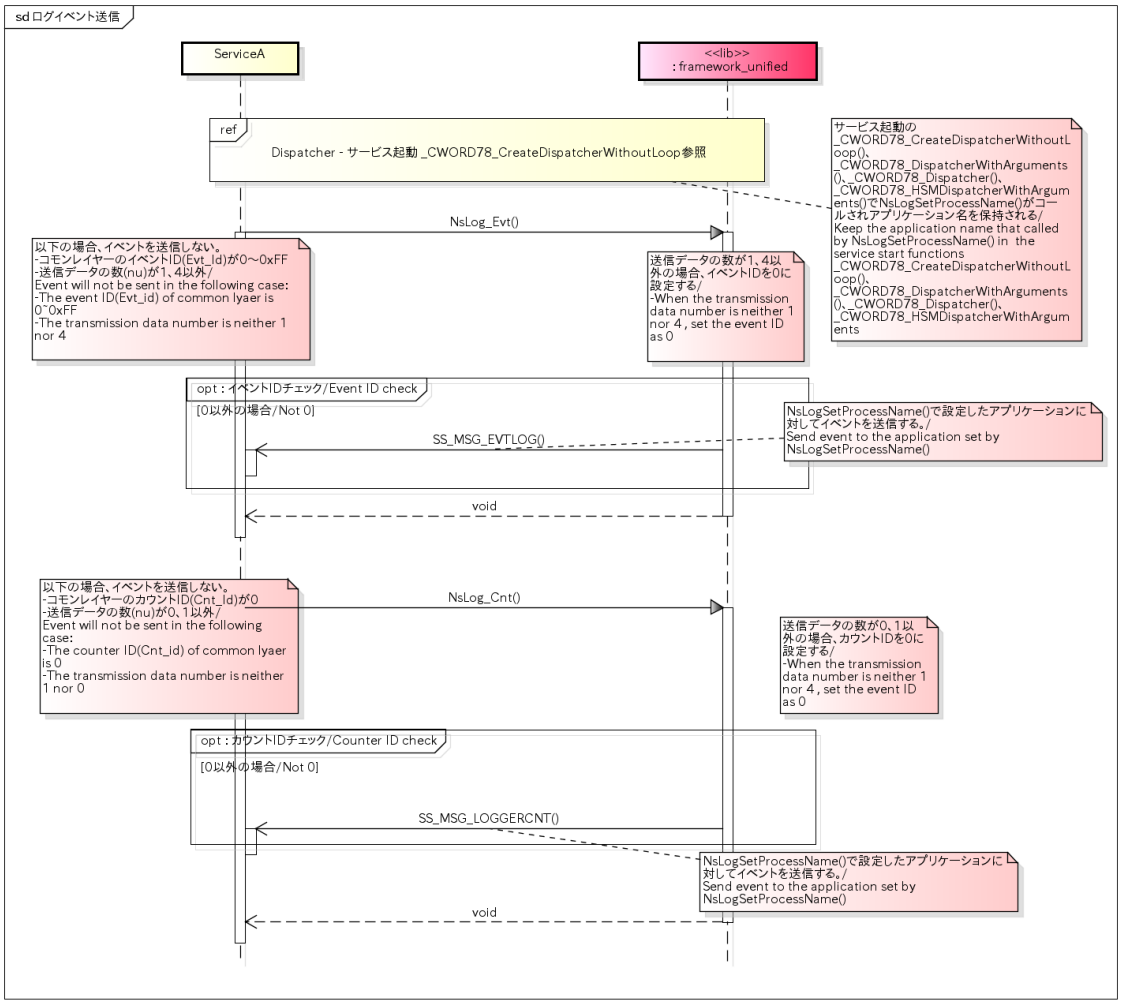
概要 *[Overview]*

本処理はログイベントを送信する機能を提供する。本APIがコールされると登録されているプリケーションに「SS_MSG_EVTLOG」、「SS_MSG_LOGGERCNT」イベントを通知する。

This API provides function to send log event.And notify 「SS_MSG_EVTLOG」、「SS_MSG_LOGGERCNT」event of Application who calls and register the API

シーケンス *[Sequence]*

ログイベント送信 */ send log event*



Mutex */Mutex*

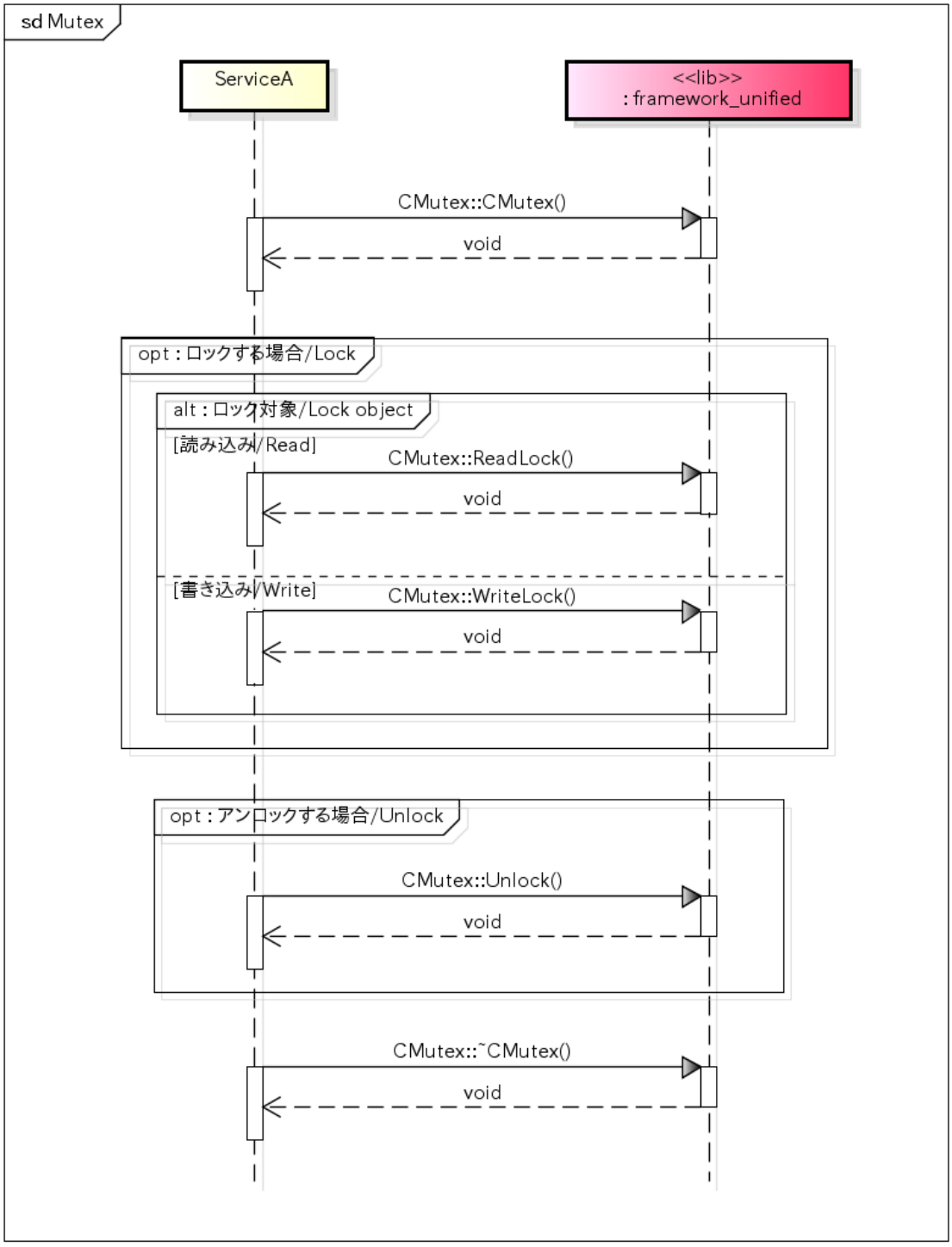
概要 *[Overview]*

ロック機能を担う。

Lock function.

シーケンス *[Sequence]*

Mutex */Mutex*



RWLock */RWLock*

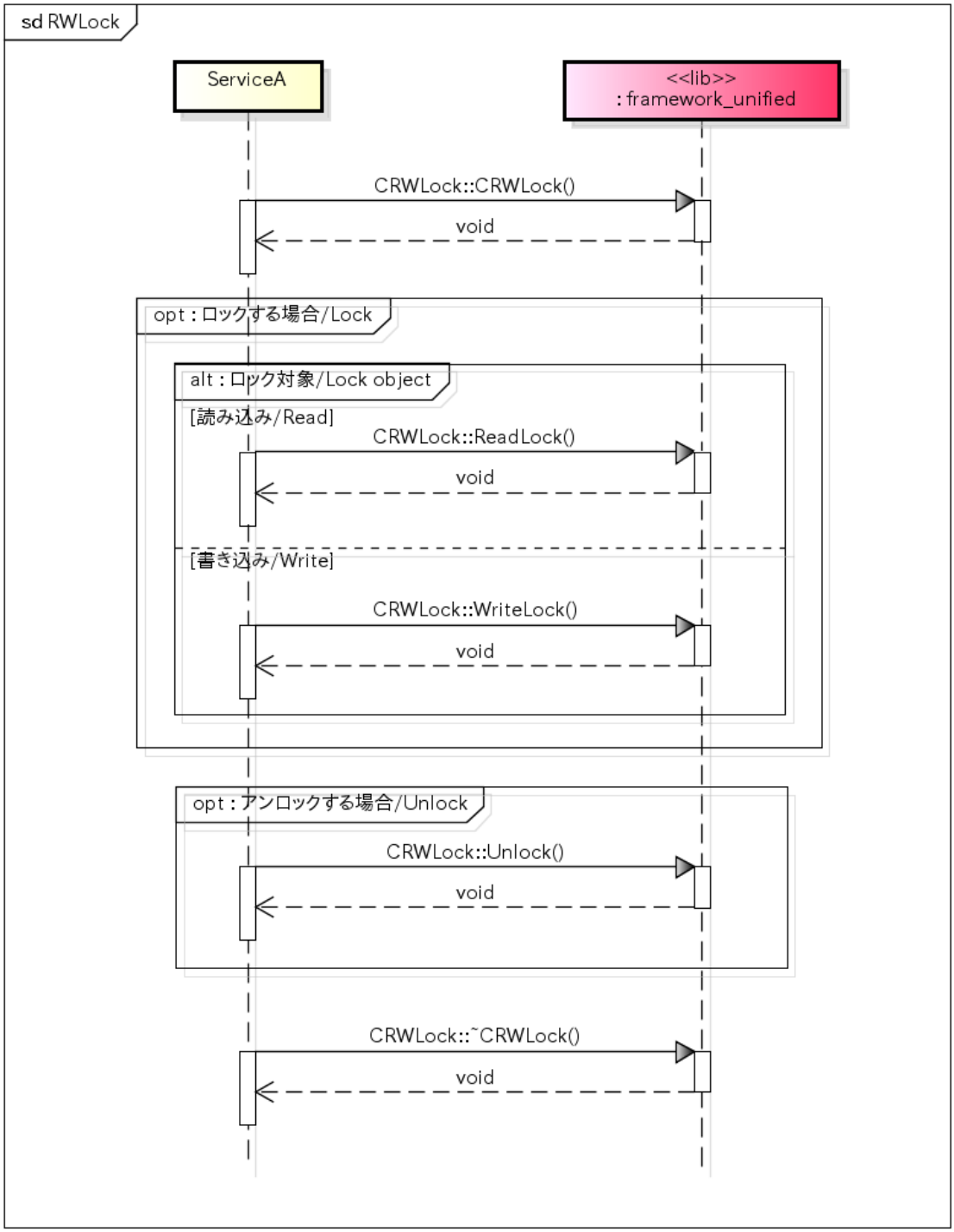
概要 *[Overview]*

リードロック、ライトロック機能を担う。

Read lock and write lock function

シーケンス *[Sequence]*

RWLock */RWLock*



Version */Version*

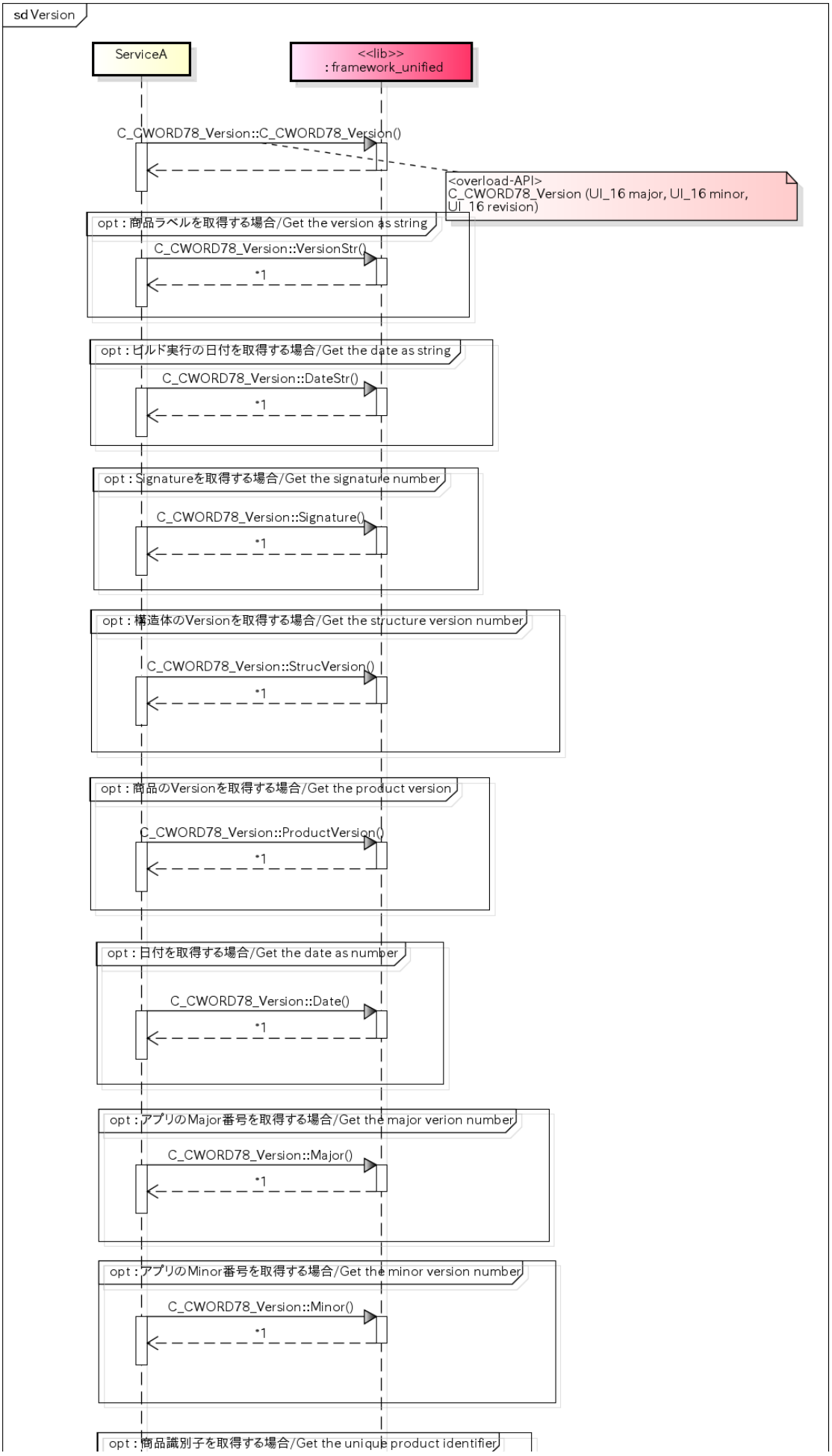
概要 *[Overview]*

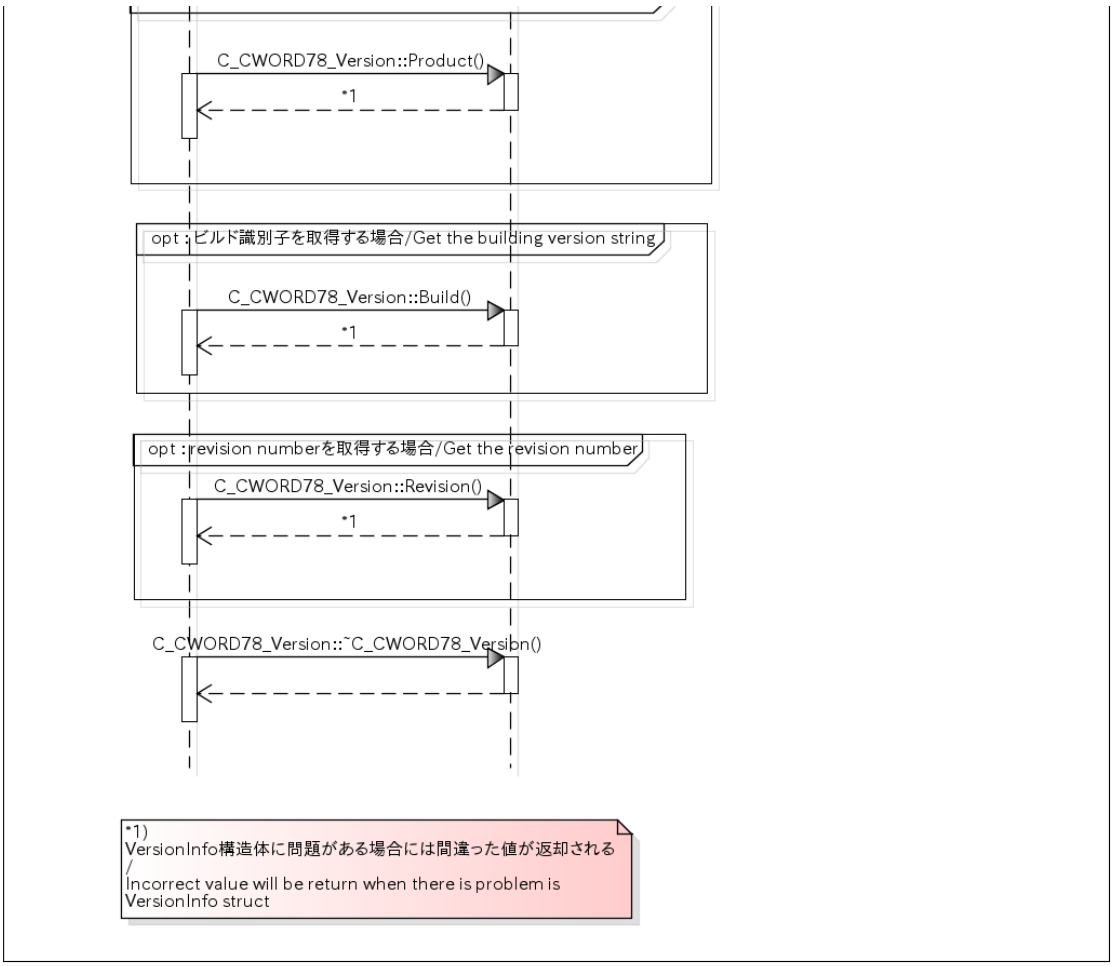
バージョン情報を取得する。

Get version information.

シーケンス *[Sequence]*

Version */Version*





HSMEvent */HSMEvent*

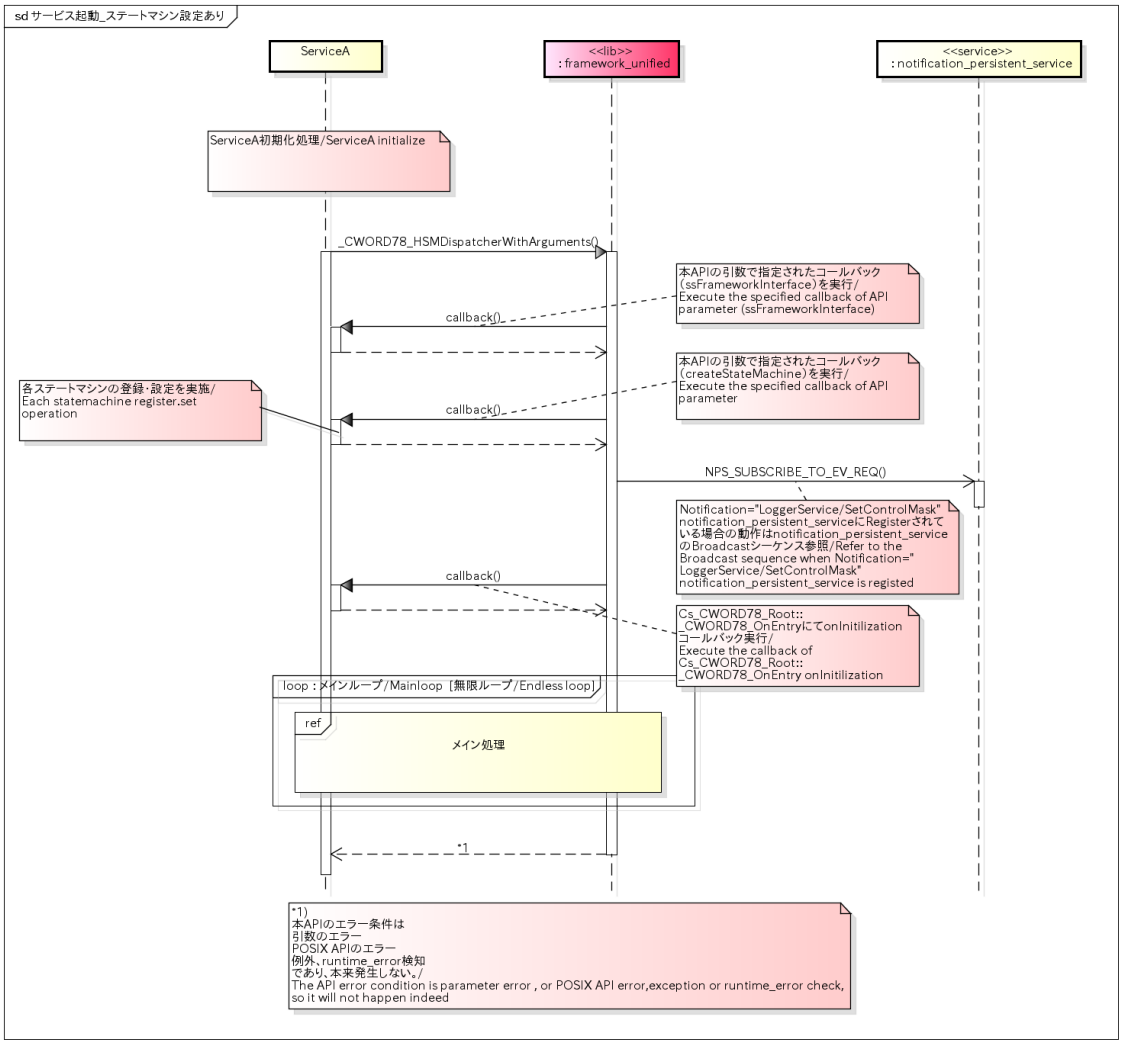
概要 *[Overview]*

本項にてステートマシンを保有するサービスについて、起動・子スレッド生成・コールバック登録を行うためのシーケンスを以下に記載する。

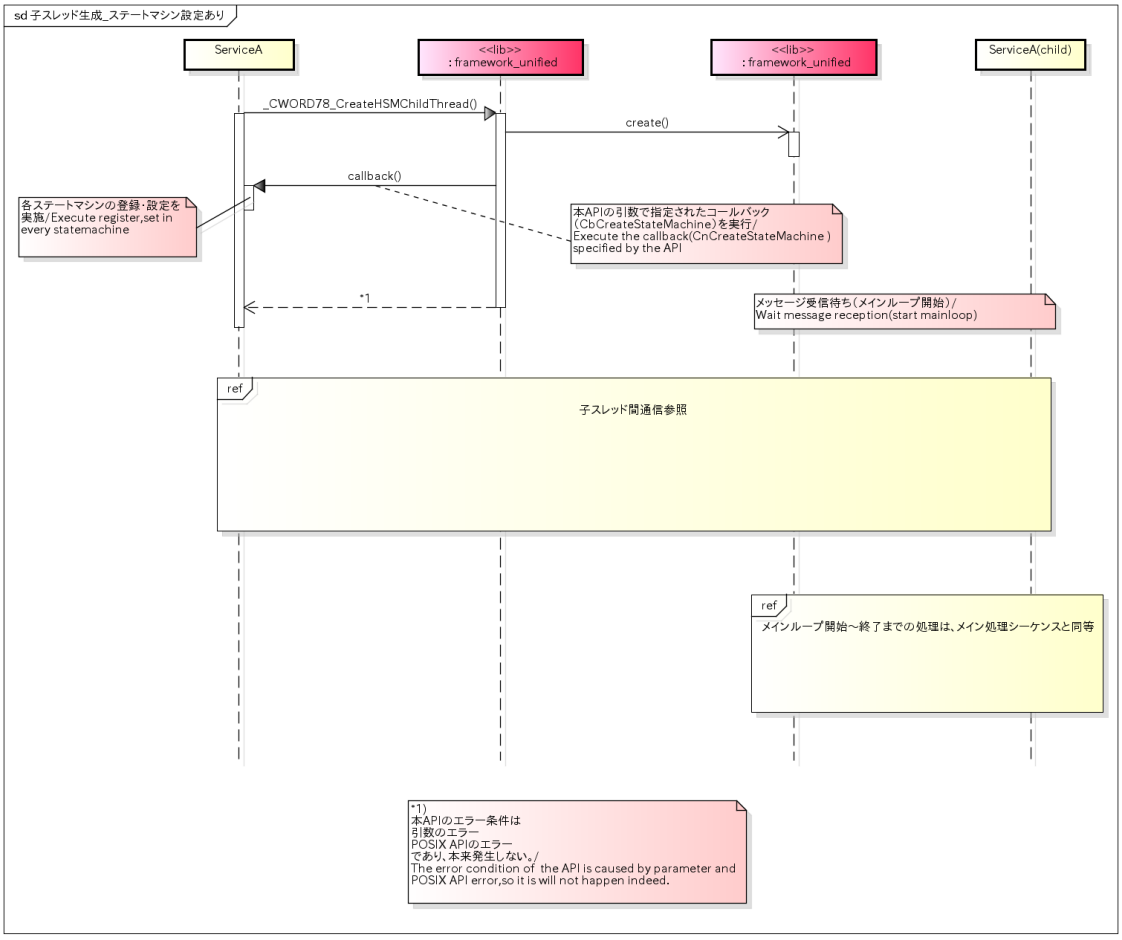
Sequences to start,create sub thread,register callback of service that has state machine,are as follows.

シーケンス *[Sequence]*

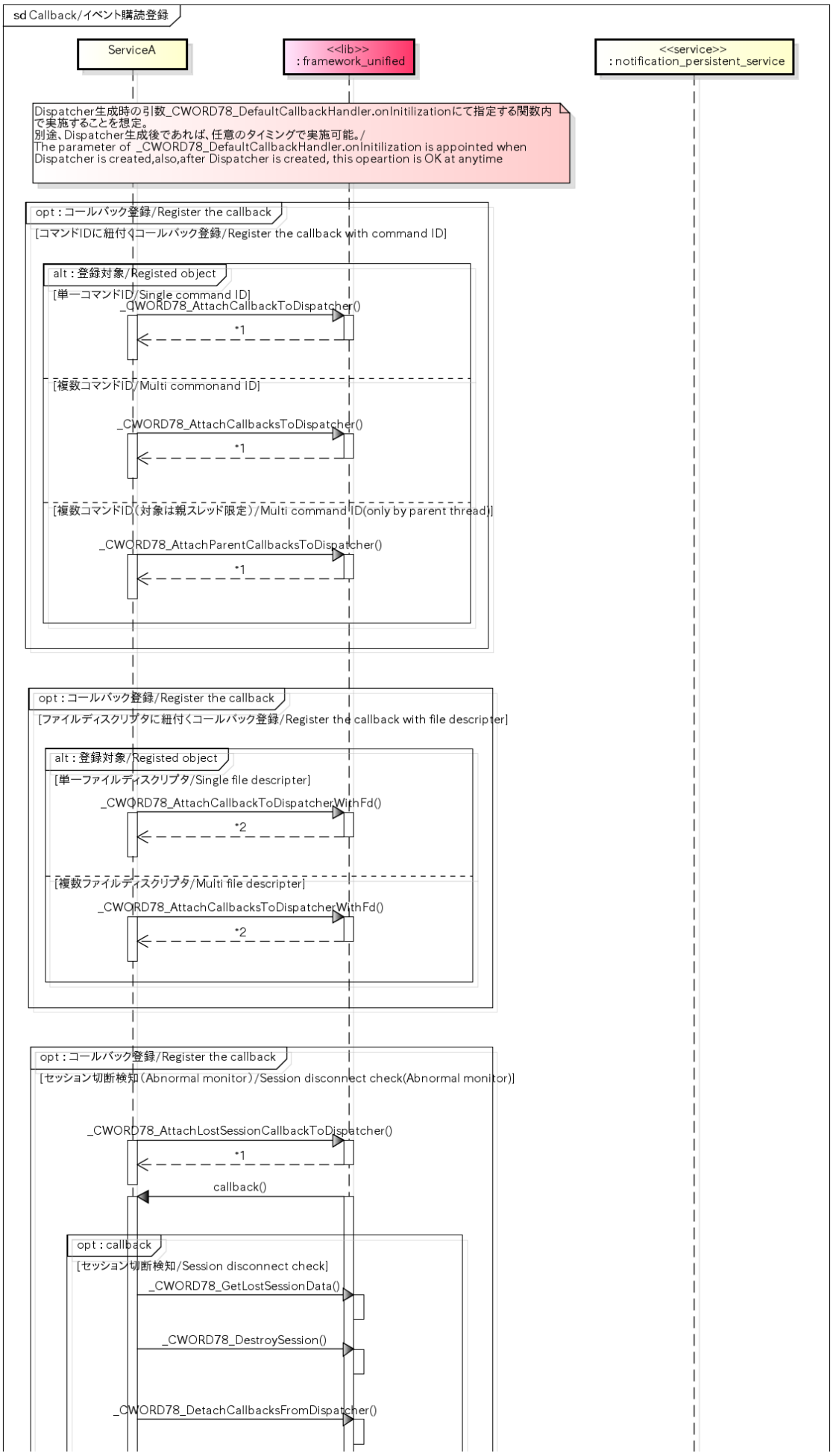
サービス起動_ステートマシン設定あり */start service(state machine configuration exists)*

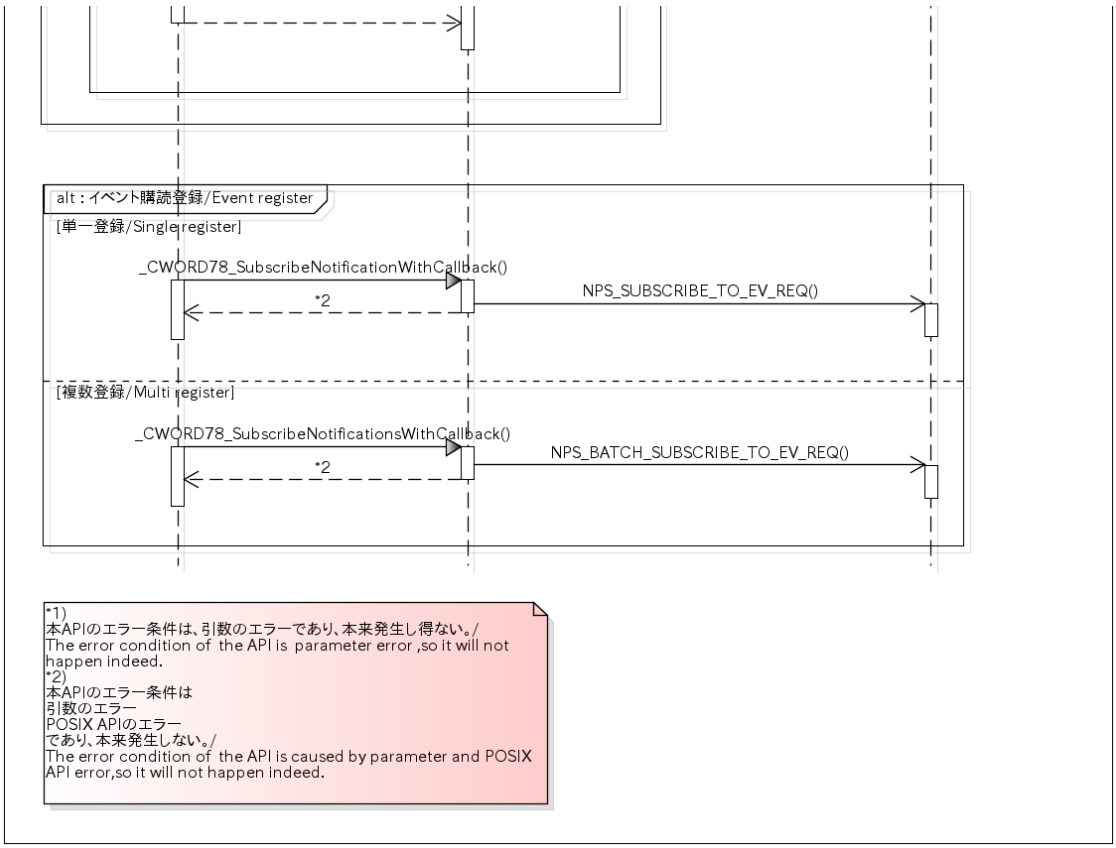


子スレッド生成_ステートマシン設定あり ***/create sub thread(state machine configuration exists)***

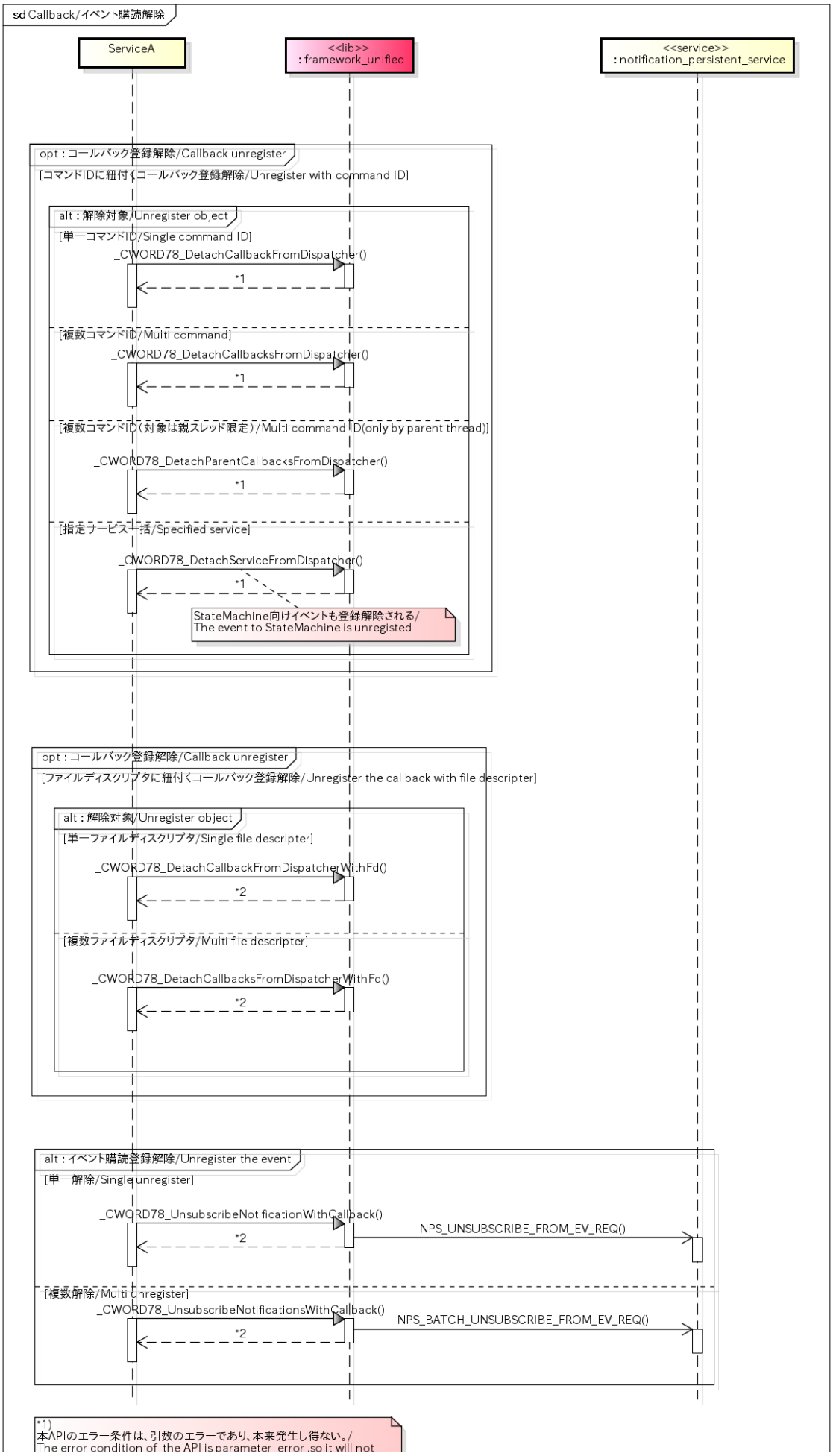


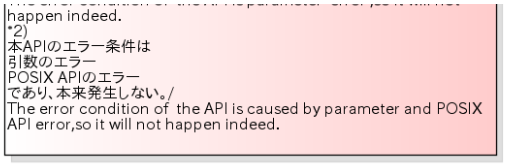
Callback及びイベント登録(ステートマシン) ***/register callback and event(statemachine)***





Callback及びイベント登録解除(ステートマシン) ***/ free callback and event register(statemachine)***





セッション管理(data pool) */session management(data pool)*

概要 *[Overview]*

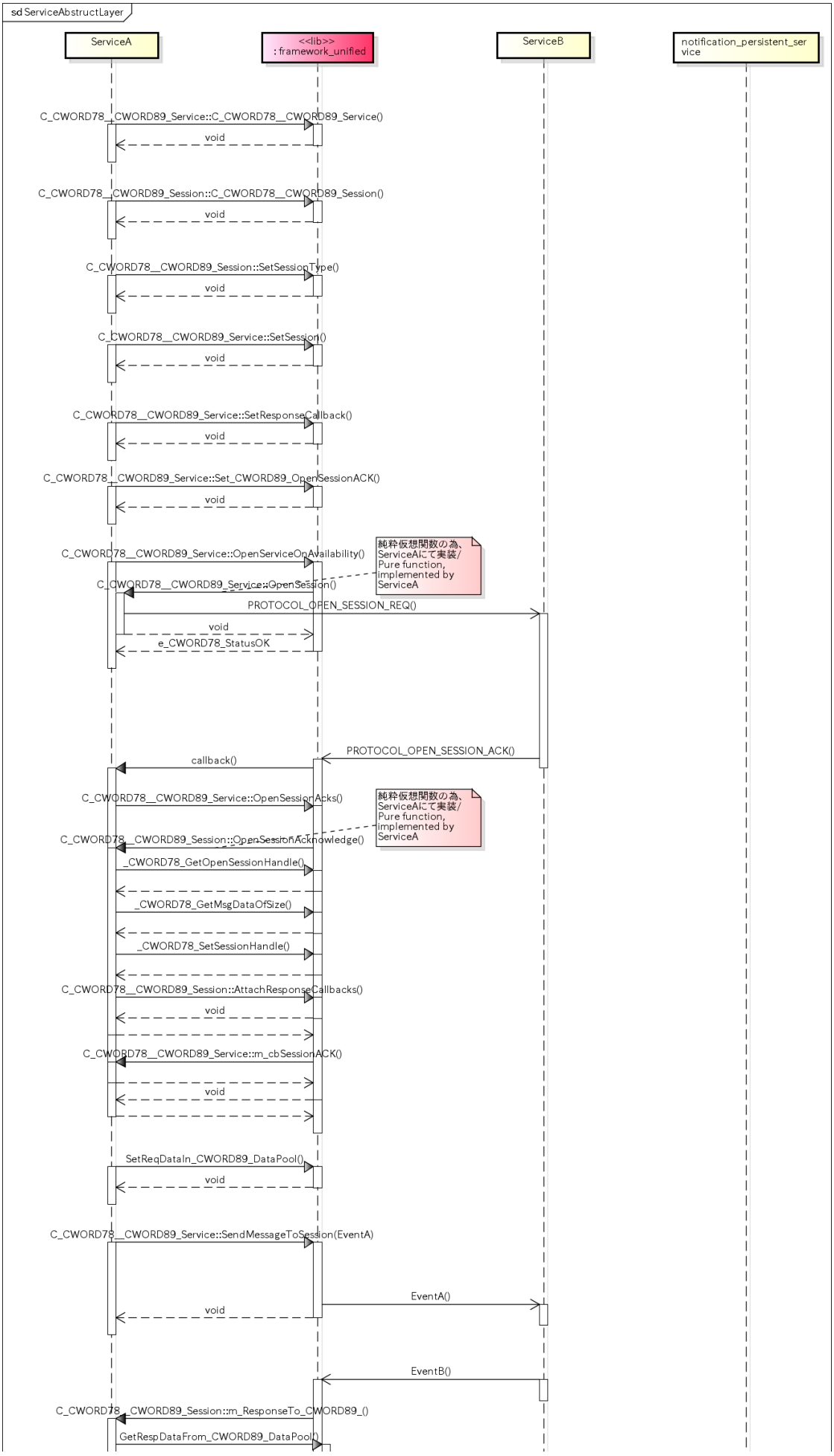
本処理は、セッションオープン/クローズの機能をクラス化したものとして提供する。
Publish-Subscribe方式の通信については、notification_persistent_serviceの機能を利用する。

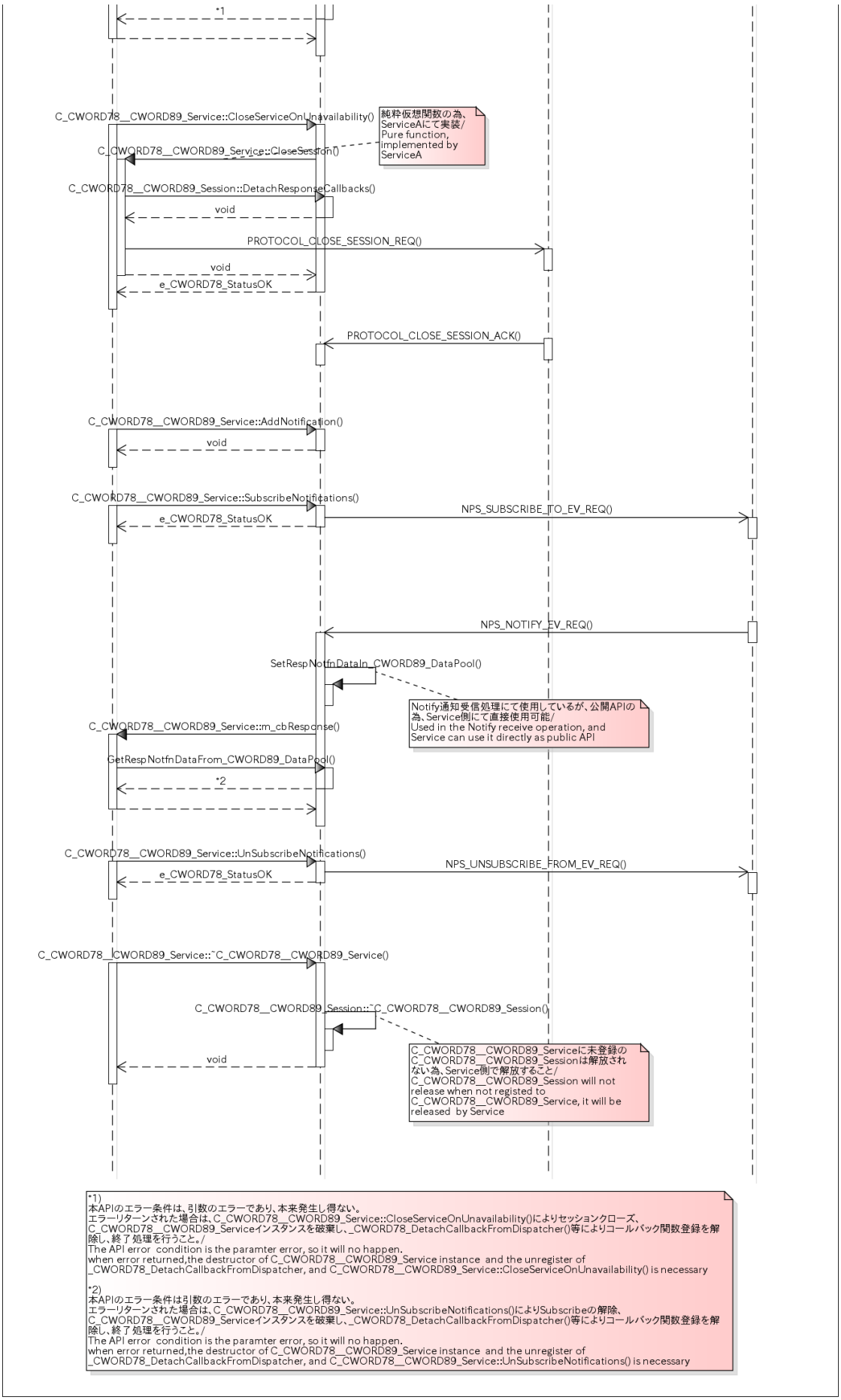
Provide session open/close function as a class.

Publish-Subscribe communication way will use function of notification_persistent_service.

シーケンス *[Sequence]*

ServiceAbstructLayer





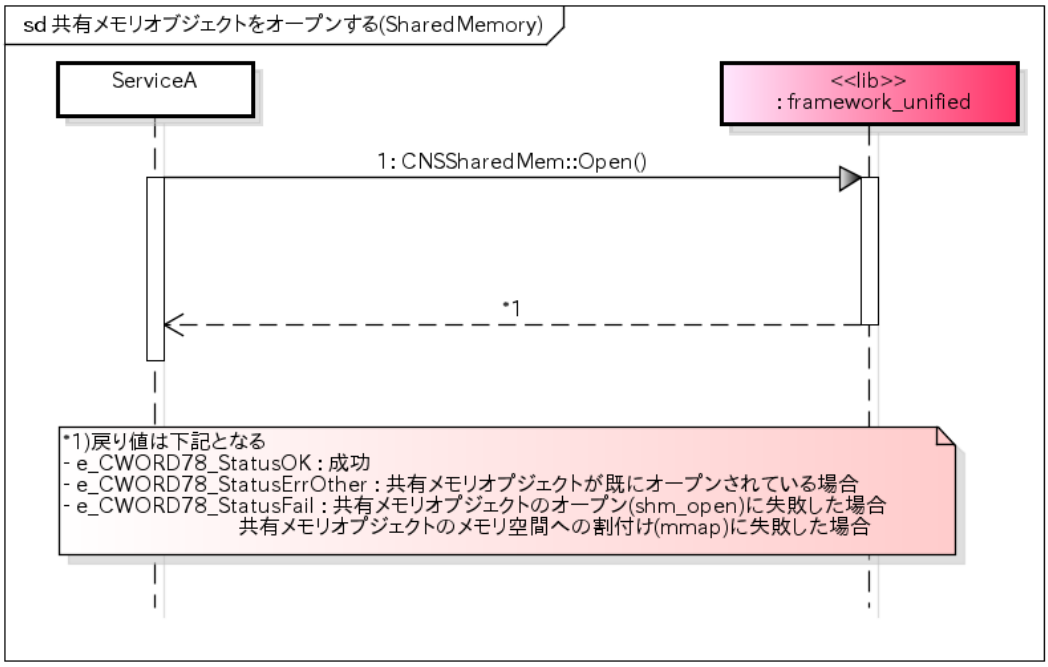
共有メモリオブジェクトをオープンする ***/Open shared memory object.***

概要 *[Overview]*

CNSSharedMemReaderクラスを用いて共有メモリオブジェクトをオープンする。
Open the shared memory object using the CNSSharedMemReader class.

シーケンス *[Sequence]*

SharedMemoryReader



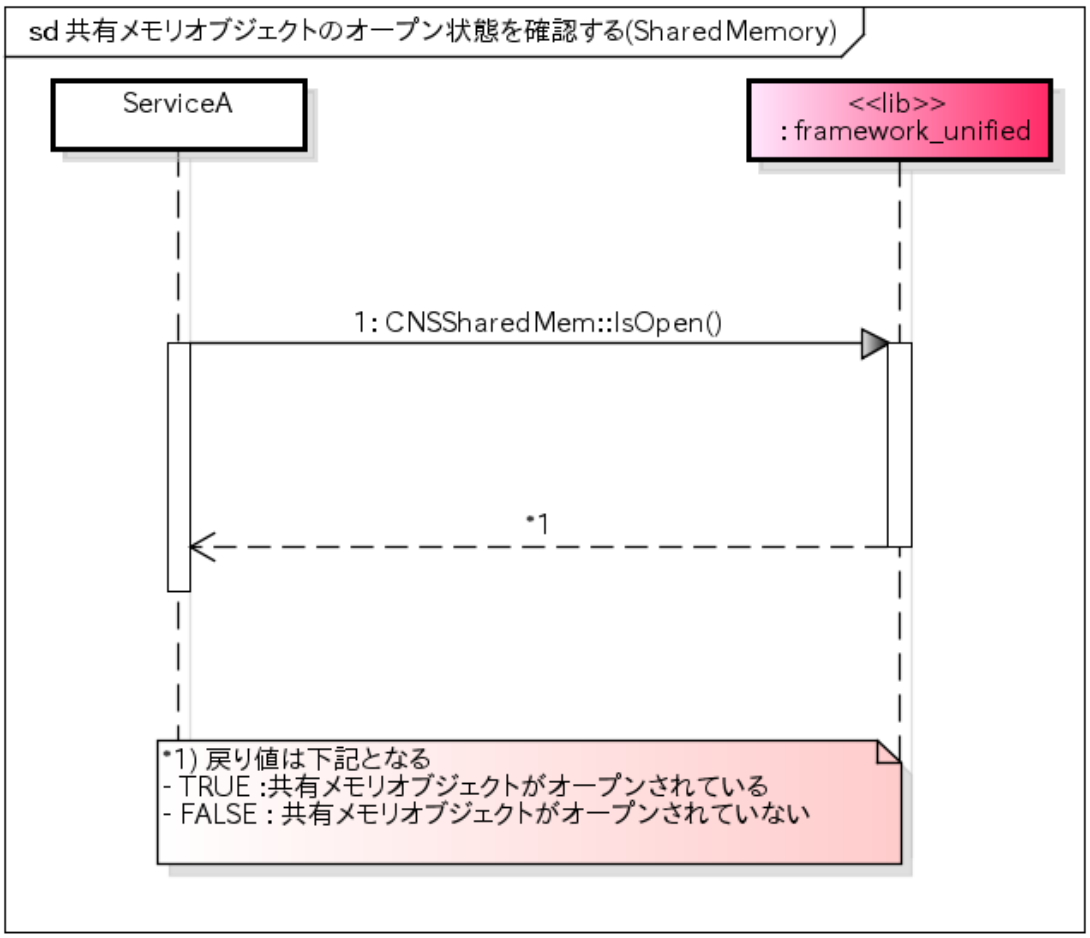
共有メモリオブジェクトのオープン状態を確認する */Check the open state of the shared memory object.*

概要 *[Overview]*

CNSSharedMemReaderクラスインスタンス内の共有メモリオブジェクトのオープン状態を確認する。
Check the open state of the shared memory object in the instance of the CNSSharedMemReader class.

シーケンス *[Sequence]*

SharedMemoryReader



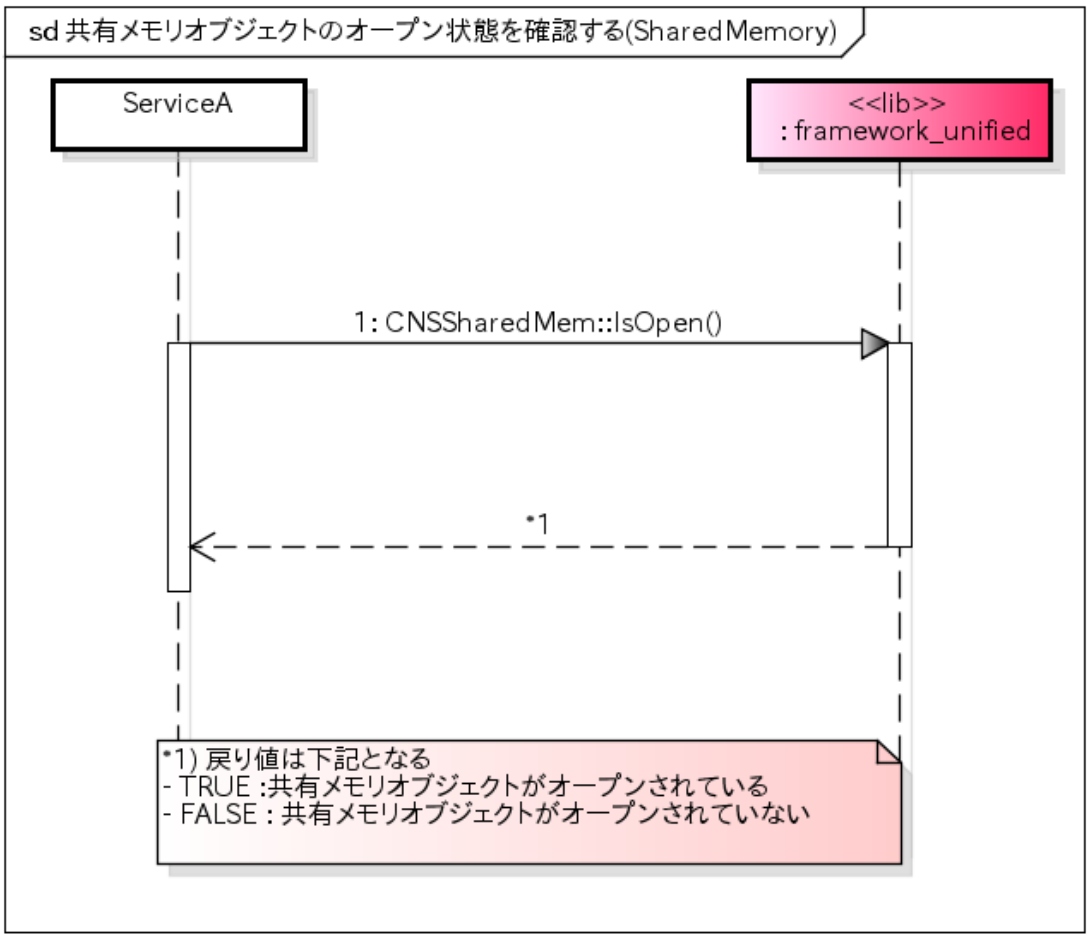
共有メモリオブジェクトをクローズする */Close shared memory object.*

概要 *[Overview]*

CNSSharedMemReaderクラスを用いて共有メモリオブジェクトをクローズする。
Close the shared memory object using the CNSSharedMemReader class.

シーケンス *[Sequence]*

SharedMemoryReader



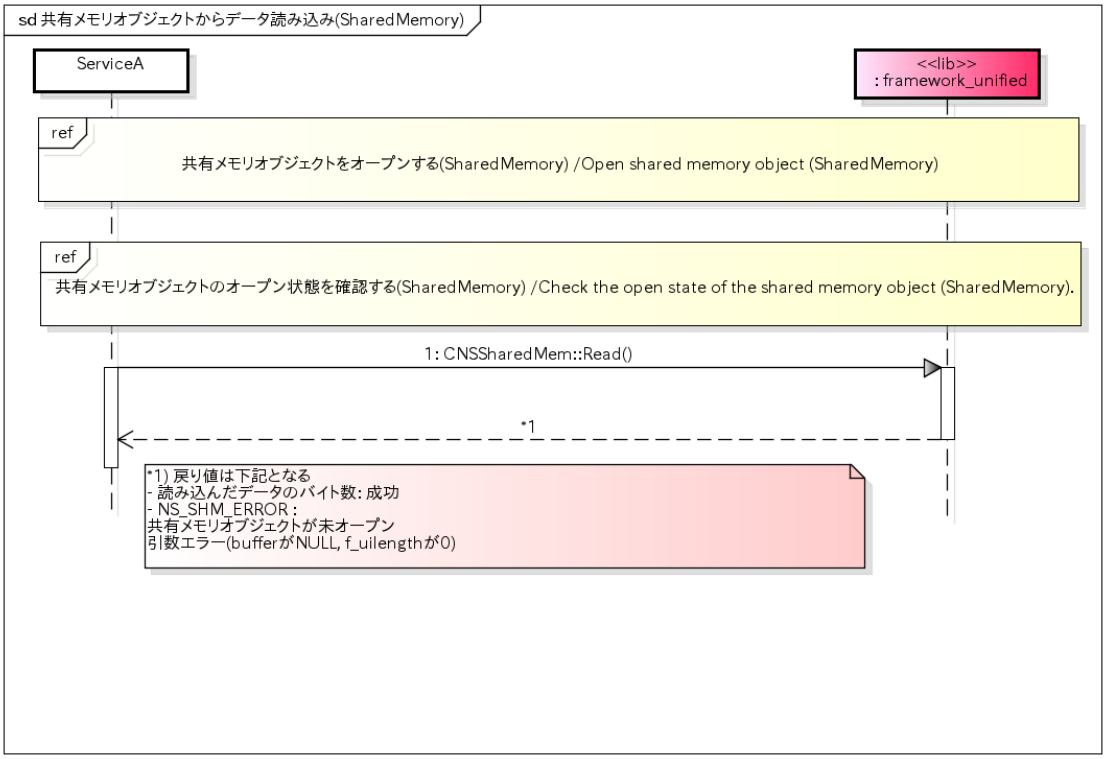
共有メモリオブジェクトからデータ読み込み */Read data from shared memory object.*

概要 *[Overview]*

共有メモリオブジェクトからデータを読み込む。
Read data from shared memory object.

シーケンス *[Sequence]*

SharedMemoryReader



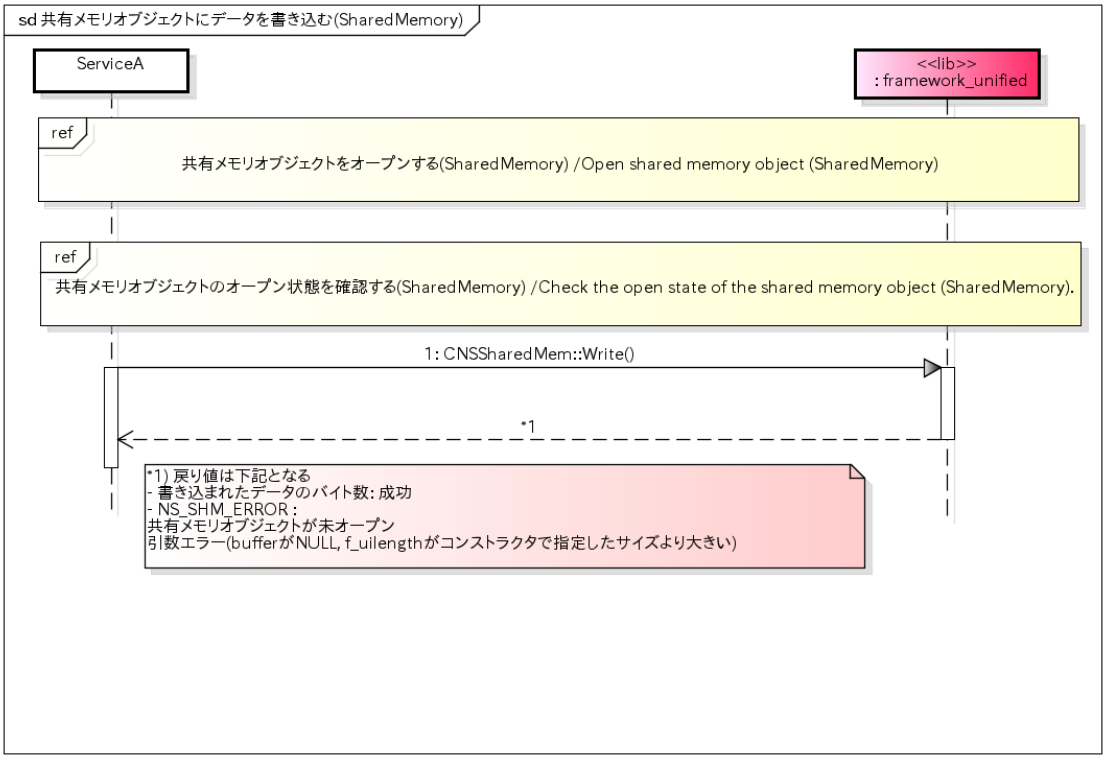
共有メモリオブジェクトのデータをファイルへ書き出し */Write the data of the shared memory object to a file.*

概要 [Overview]

共有メモリオブジェクトのデータをファイルへ書き出す。
Write the data of the shared memory object to a file.

シーケンス [Sequence]

SharedMemoryReader



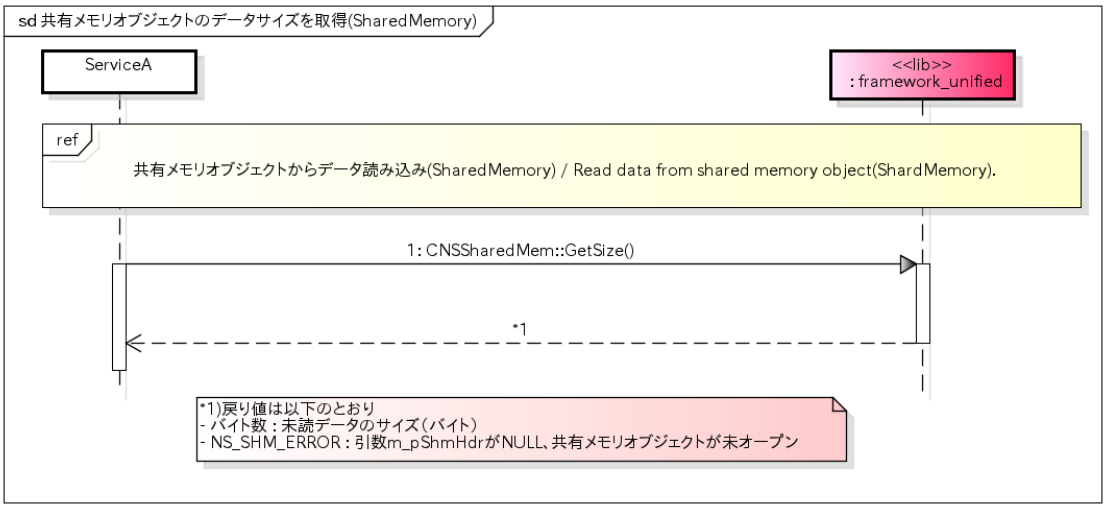
共有メモリオブジェクトのデータサイズを取得 */Get the data size of the shared memory object.*

概要 [Overview]

共有メモリオブジェクトのデータサイズを取得する。
Get the data size of the shared memory object.

シーケンス [Sequence]

SharedMemoryReader



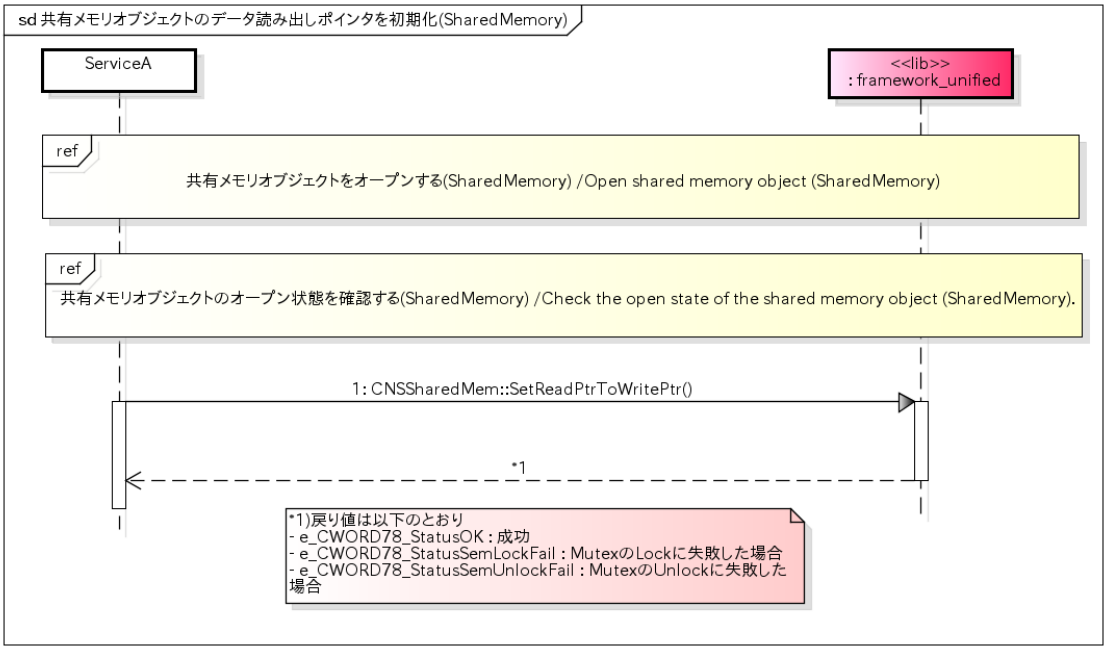
共有メモリオブジェクトのデータ読み出しポインタを初期化 */Initializes the data read pointer of the shared memory object.*

概要 [Overview]

共有メモリオブジェクトの読み出し位置ポインタを書き込み位置ポインタと同一に初期化する。
Initializes the read position pointer of the shared memory object as same as the write position pointer.

シーケンス [Sequence]

SharedMemoryReader



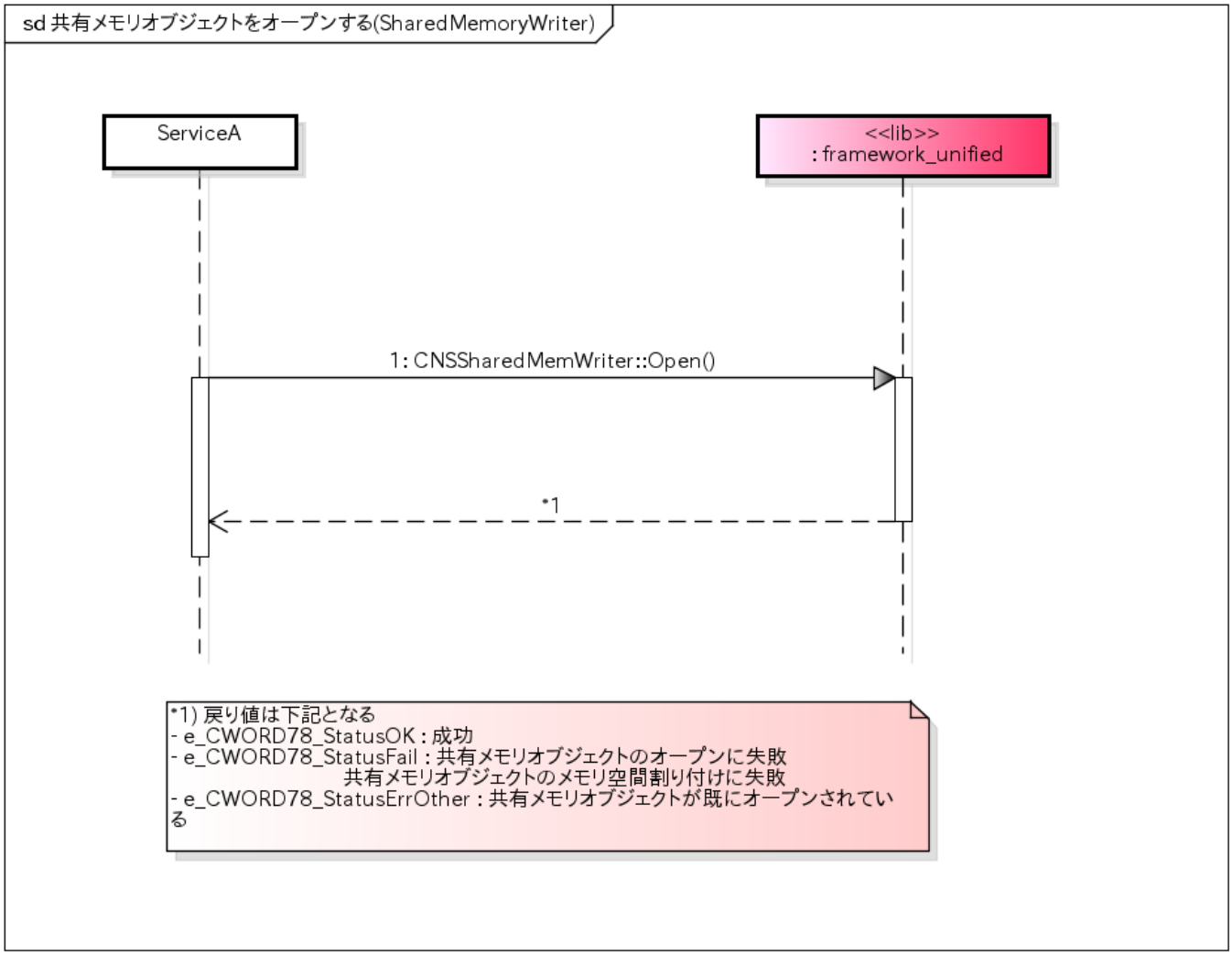
共有メモリオブジェクトをオープンする(SharedMemoryWriter) */Open shared memory object (SharedMemoryWriter).*

概要 *[Overview]*

CNSSharedMemWriterクラスを用いて共有メモリオブジェクトをオープンする。
Open the shared memory object using the CNSSharedMemWriter class.

シーケンス *[Sequence]*

SharedMemoryWriter



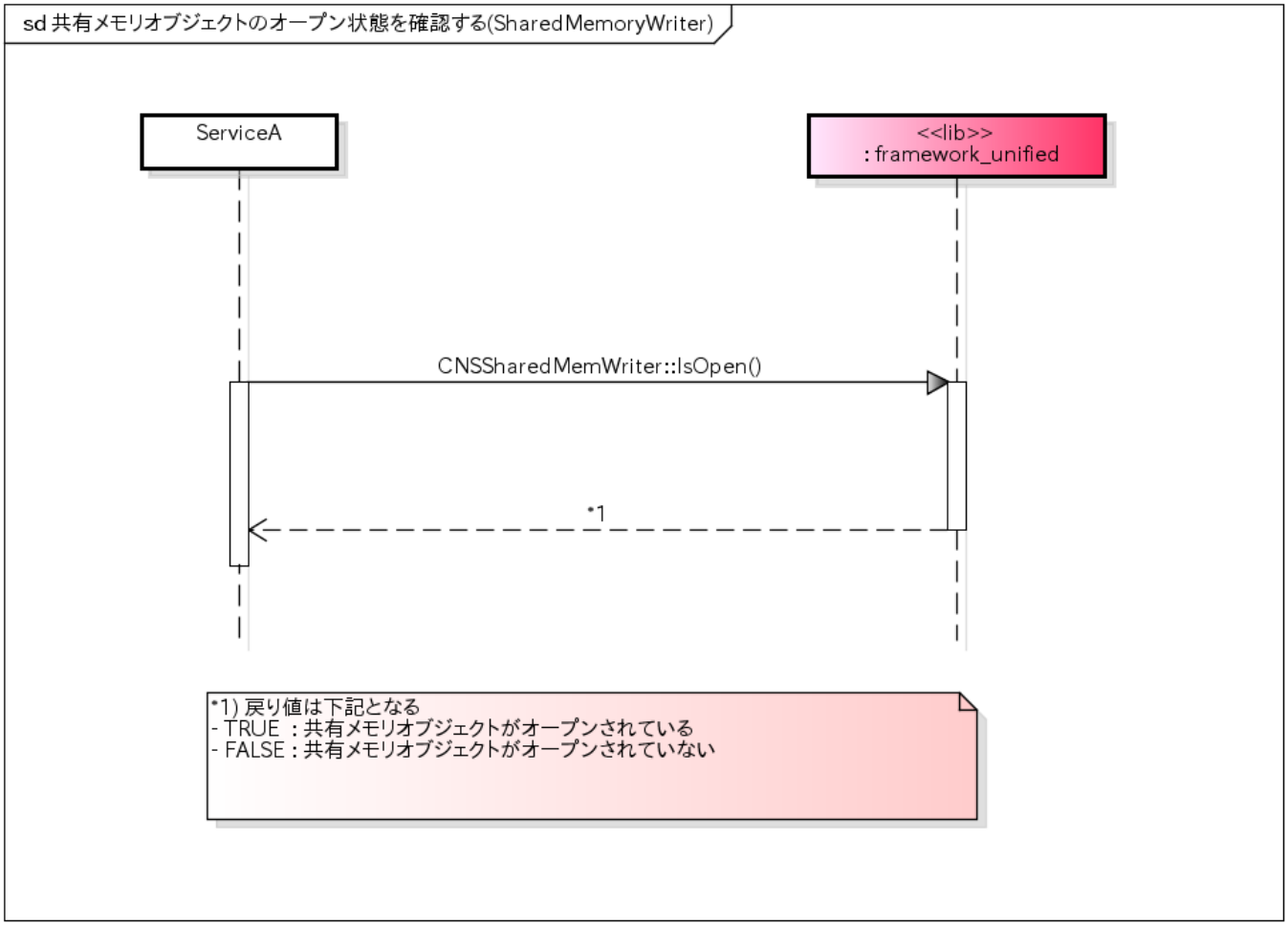
共有メモリオブジェクトのオープン状態を確認する(SharedMemoryWriter) */Check the open state of the shared memory object (SharedMemoryWriter).*

概要 *[Overview]*

CNSSharedMemWriterクラスインスタンス内の共有メモリオブジェクトのオープン状態を確認する。
Check the open state of the shared memory object in the instance of the CNSSharedMemWriter class.

シーケンス *[Sequence]*

SharedMemoryWriter



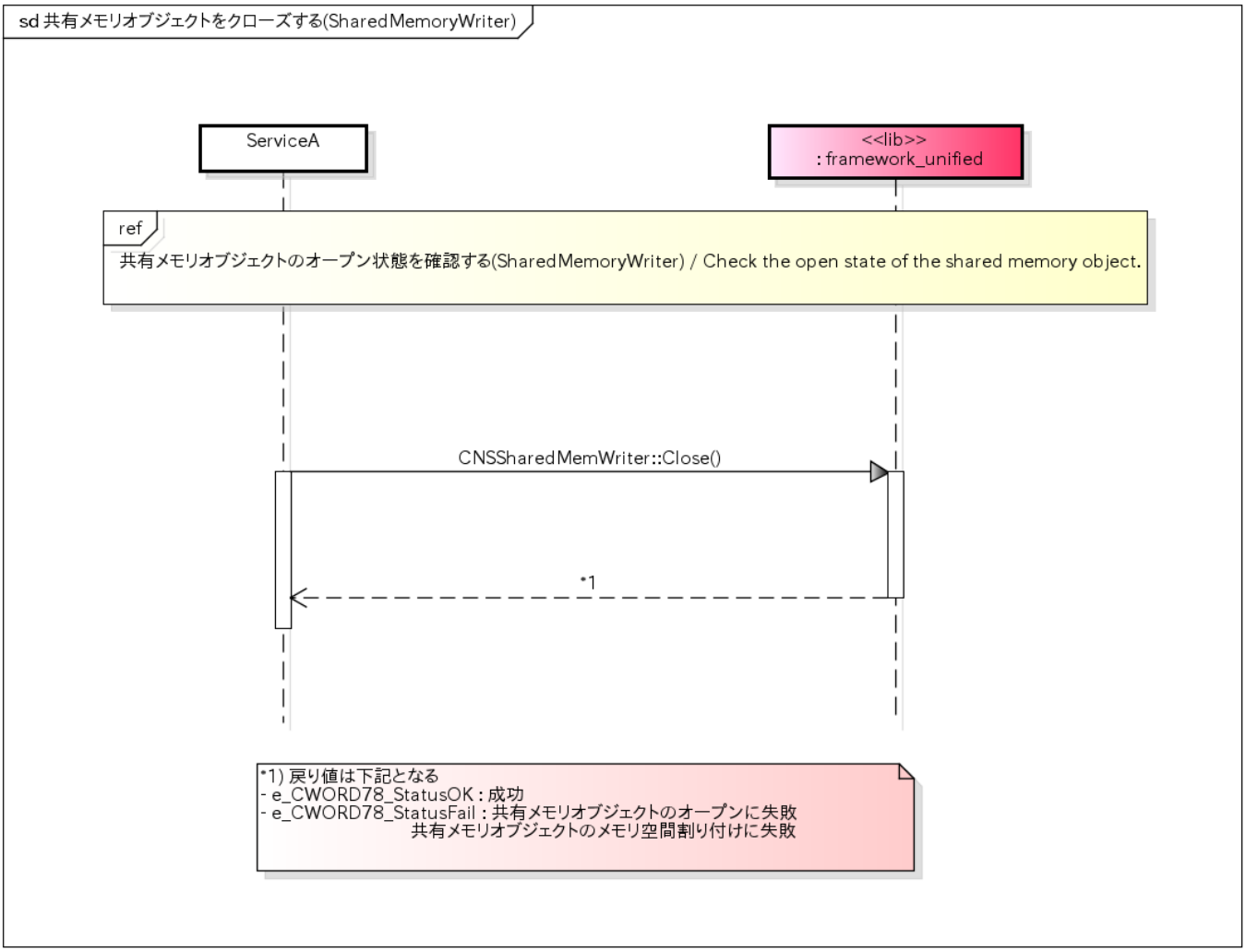
共有メモリオブジェクトをクローズする(SharedMemoryWriter) */Close shared memory object (SharedMemoryWriter).*

概要 *[Overview]*

CNSSharedMemWriterクラスを用いて共有メモリオブジェクトをクローズする。
Close the shared memory object using the CNSSharedMemWriter class.

シーケンス *[Sequence]*

SharedMemoryWriter



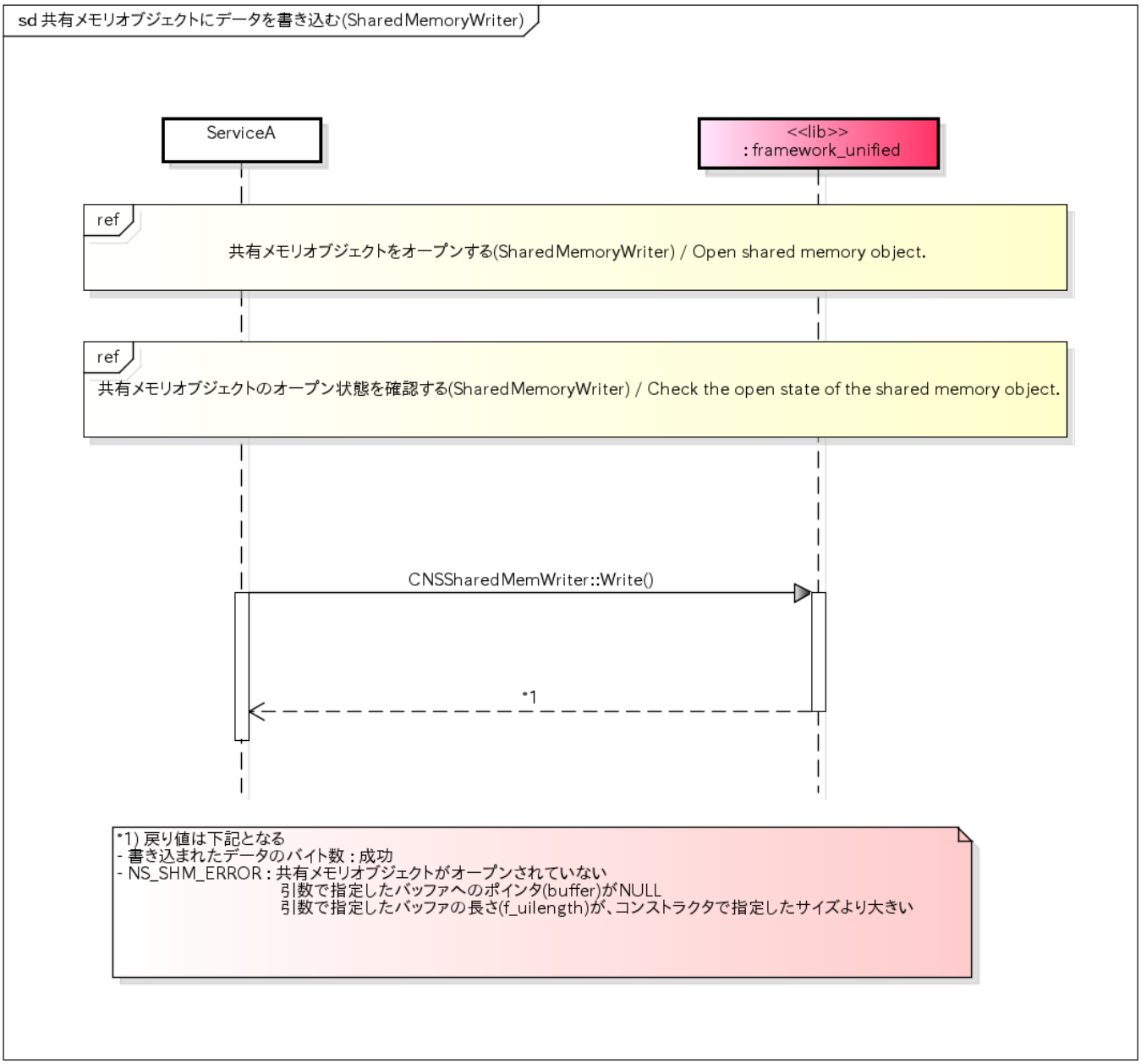
共有メモリオブジェクトにデータを書き込む */Write the data to shared memory object.*

概要 *[Overview]*

CNSSharedMemWriterクラスを用いて共有メモリオブジェクトにデータを書き込む。
Write the data to shared memory object using the CNSSharedMemWriter class.

シーケンス *[Sequence]*

SharedMemoryWriter



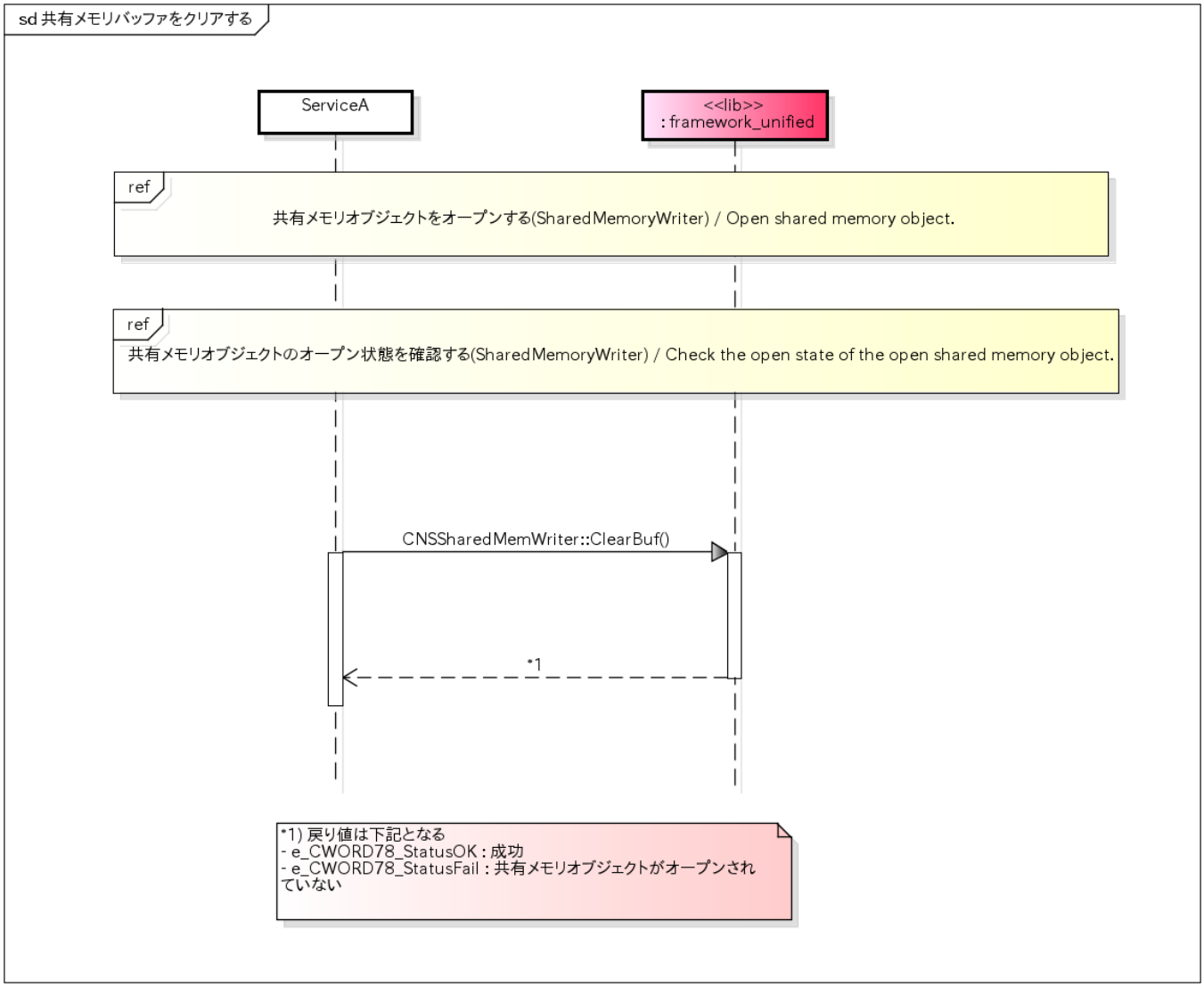
共有メモリバッファーをクリアする */Clear the shared memory buffer.*

概要 *[Overview]*

CNSSharedMemWriterクラスを用いて共有メモリバッファーをクリアする。
Clear the shared memory buffer using CNSSharedMemWriter class.

シーケンス *[Sequence]*

SharedMemoryWriter



Operation of Shared Memory /Operation of Shared Memory

概要 [Overview]

Shared Memory操作についてシーケンスを以下に記載する。

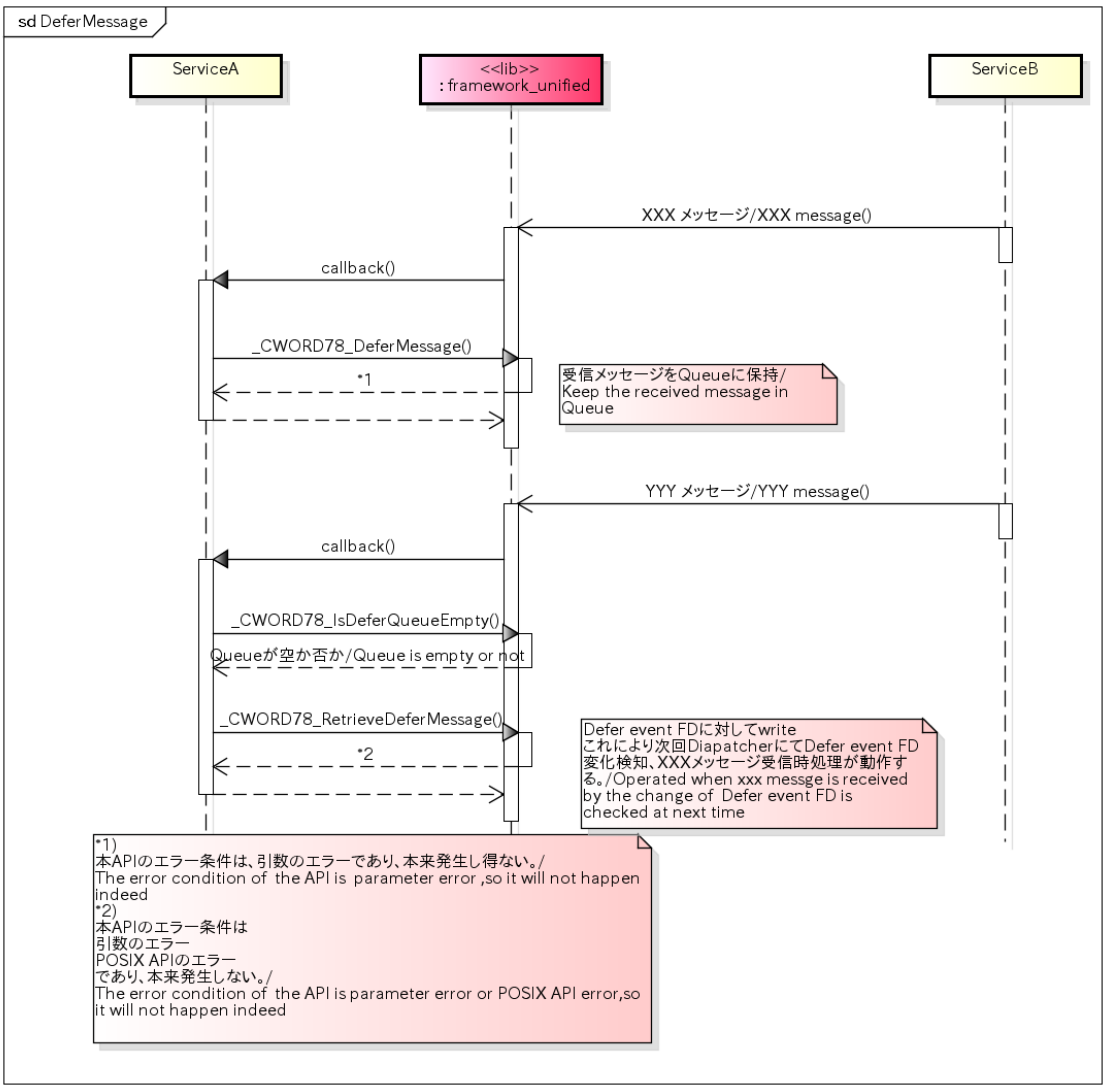
Getter,Setterを用いることで、任意のデータを取り扱うことが出来る。

Sequence of operation of shared memory is as follows.

Using Getter API and Setter API, It can handle any data.

シーケンス [Sequence]

DeferMessage /DeferMessage



共有メモリオブジェクトをオープンする(SharedMemory) / *Open shared memory object (SharedMemory).*

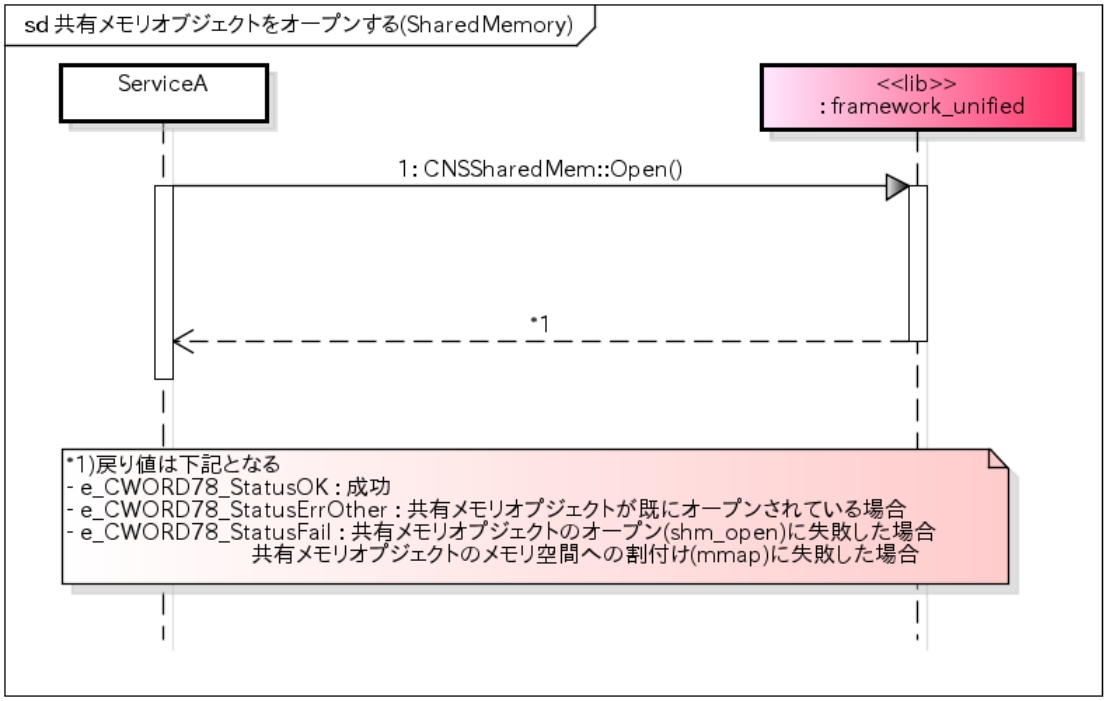
概要 *[Overview]*

CNSSharedMemクラスを用いて、共有メモリオブジェクトをオープンする

Open the shared memory object using the CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



共有メモリオブジェクトのオープン状態を確認する(SharedMemory) / *Check the open state of the shared memory object (SharedMemory).*

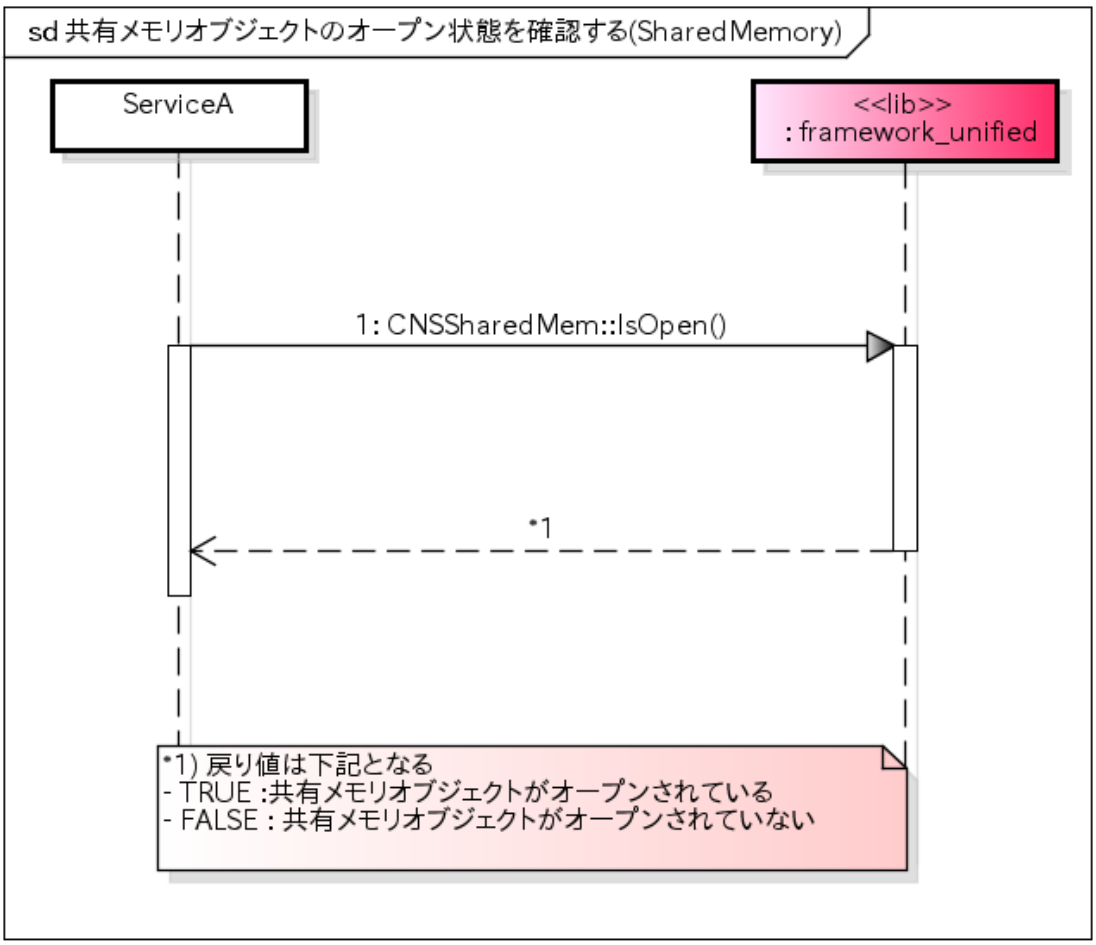
概要 *[Overview]*

CNSSharedMemクラスインスタンス内の共有メモリオブジェクトのオープン状態を確認する。

Check the open state of the shared memory object in the instance of the CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



共有メモリオブジェクトをクローズする(SharedMemory) / *Close shared memory object (SharedMemory).*

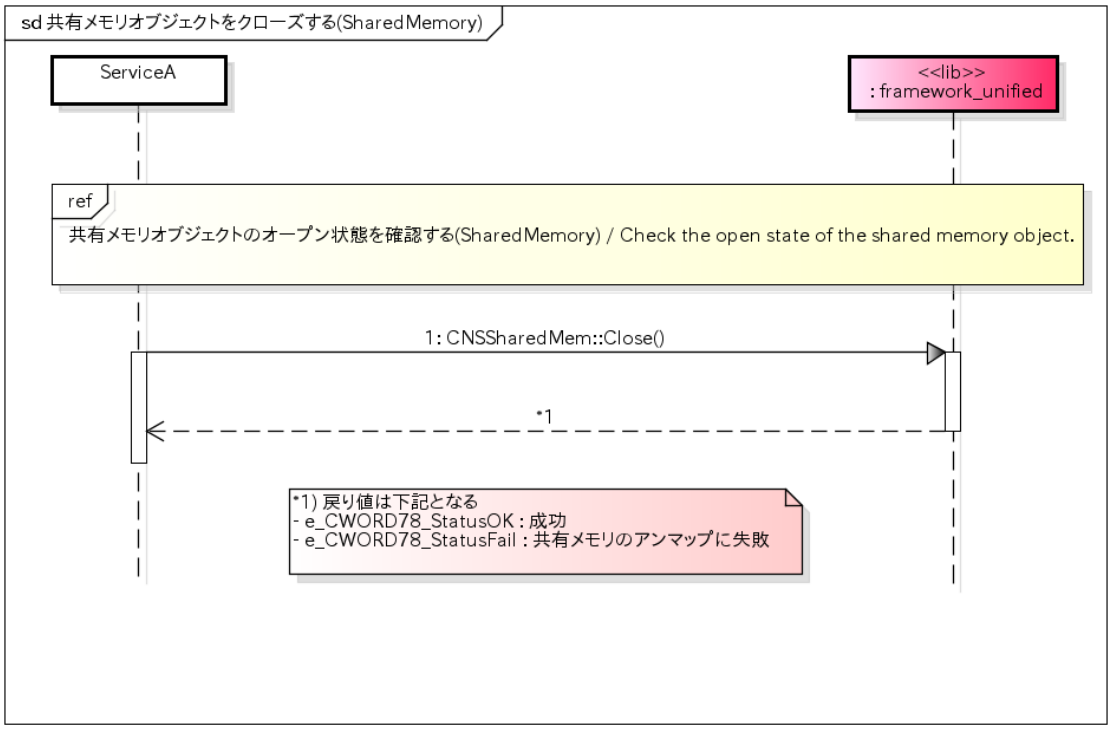
概要 *[Overview]*

CNSSharedMemクラスを用いてオープンした共有メモリオブジェクトをクローズする。

Close the shared memory object by using the CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



共有メモリオブジェクトからデータ読み込み(SharedMemory) / *Read data from shared memory object(ShardMemory).*

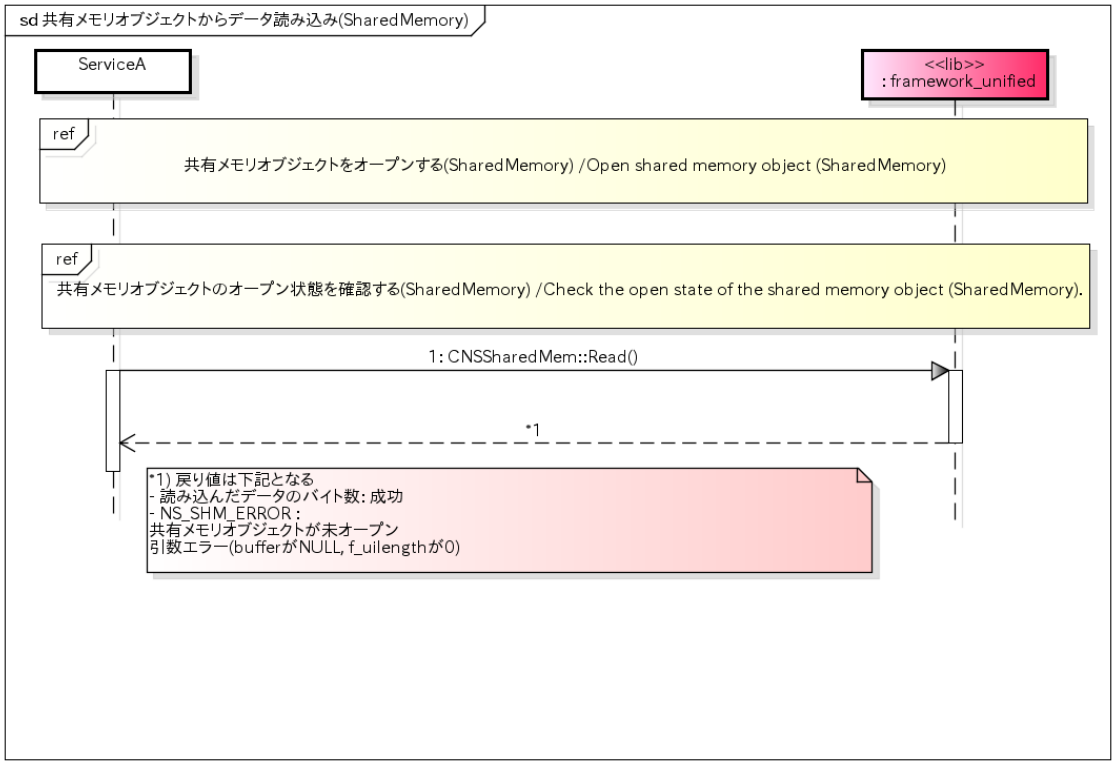
概要 *[Overview]*

CNSSharedMemクラスのインスタンスを用いて、共有メモリオブジェクトからデータを読み込む

Read the data from shared memory object by using the instance of the CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



共有メモリオブジェクトのデータをファイルへ書き出し(SharedMemory) / *Write the data of the shared memory object to a file(SharedMemory).*

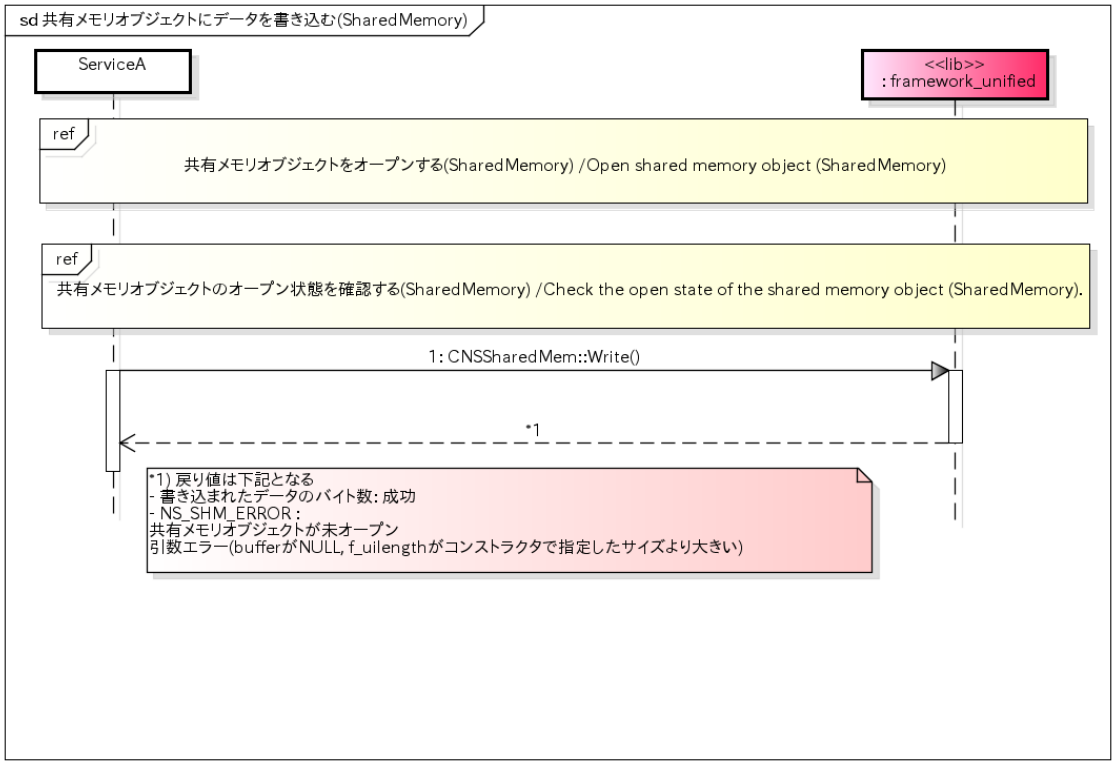
概要 *[Overview]*

CNSSharedMemクラスのインスタンスを用いて、読み込んだ共有メモリオブジェクトのデータをファイルへ書き出す。

Write the read data of the shared memory object to a file by using the instance of CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



共有メモリオブジェクトのデータサイズを取得(ShraedMemory) / *Get the data size of the shared memory object(SharedMemory).*

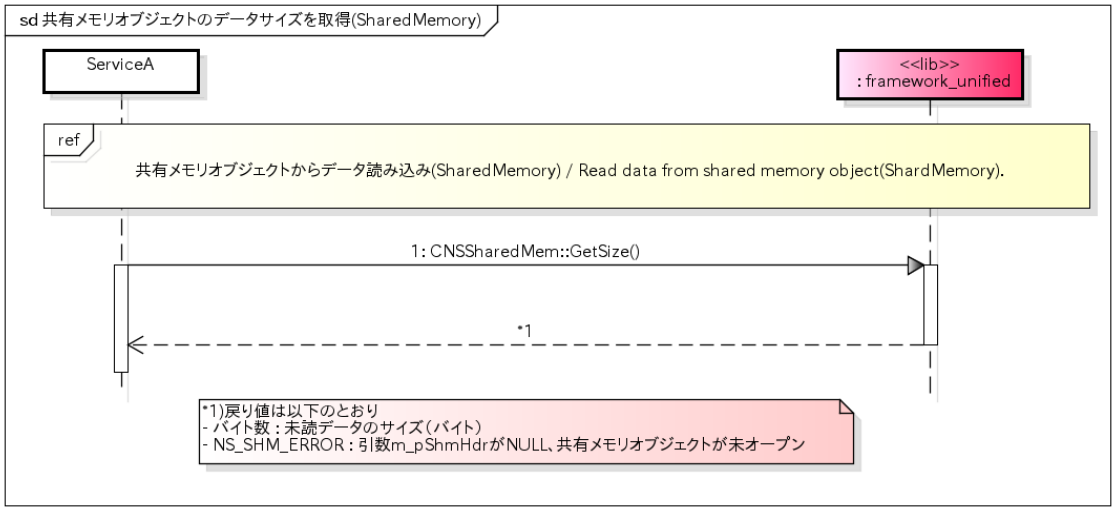
概要 *[Overview]*

CNSShraedMemクラスのインスタンスを用いて、読み込んだ共有メモリオブジェクトのデータサイズを取得する。

Get the read data size of the shared memory object by using the instance of CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



共有メモリオブジェクトのデータ読み出しポインタを初期化(SharedMemory) / *Initializes the data read pointer of the shared memory object(SharedMemory).*

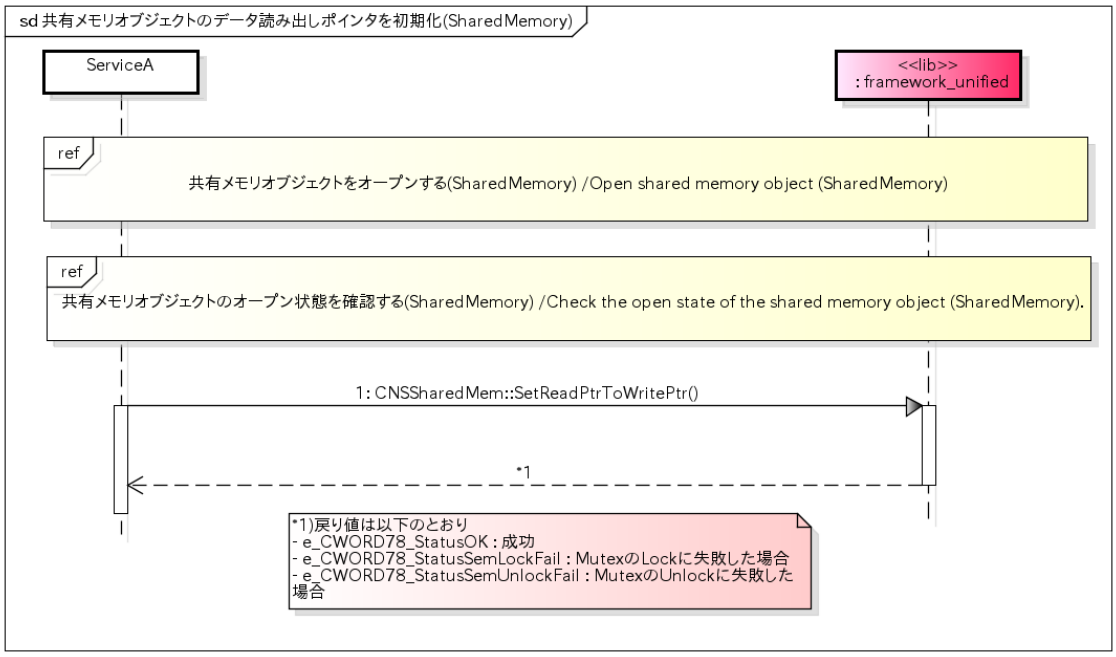
概要 *[Overview]*

共有メモリオブジェクトの読み出し位置ポインタを書き込み位置ポインタと同一に初期化する。

Initializes the read position pointer of the shared memory object as same as the write position pointer.

シーケンス *[Sequence]*

SharedMemory



共有メモリオブジェクトにデータを書き込む(SharedMemory) / *Write the data to shared memory object(SharedMemory).*

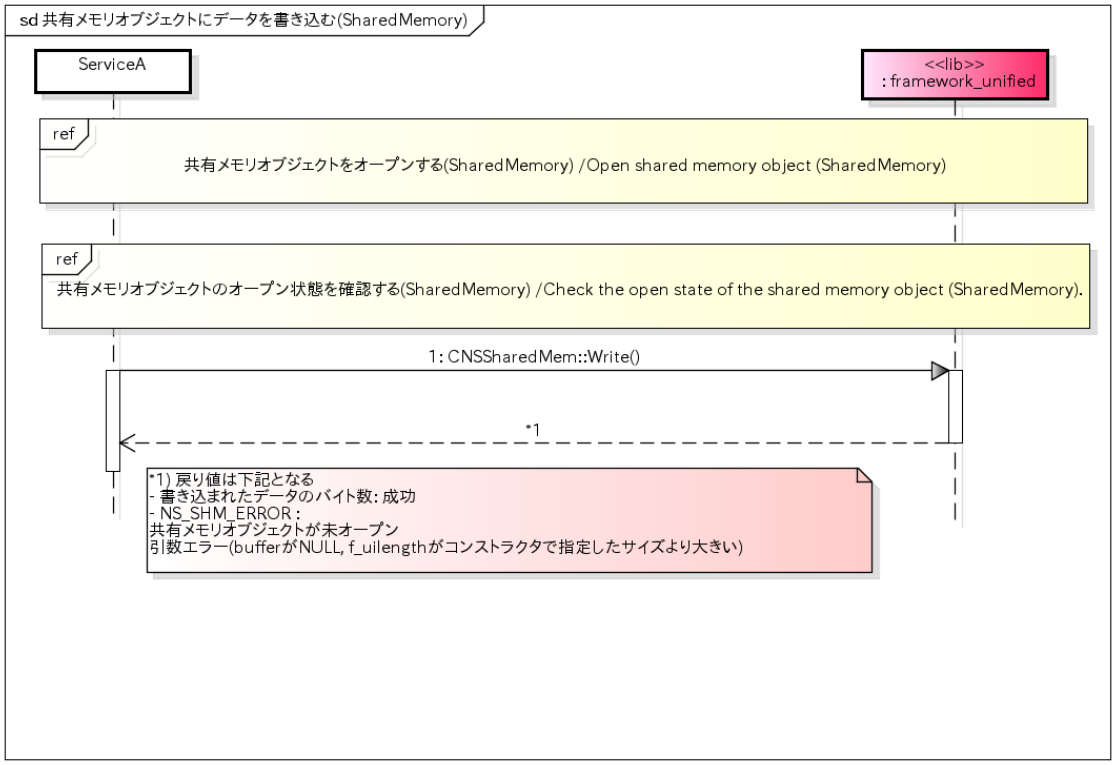
概要 *[Overview]*

CNSSharedMemクラスを用いて共有メモリオブジェクトにデータを書き込む。

Write the data to shared memory object by using the CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



共有メモリバッファーをクリアする(SharedMemory) / *Clear the shared memory buffer (SharedMemory).*

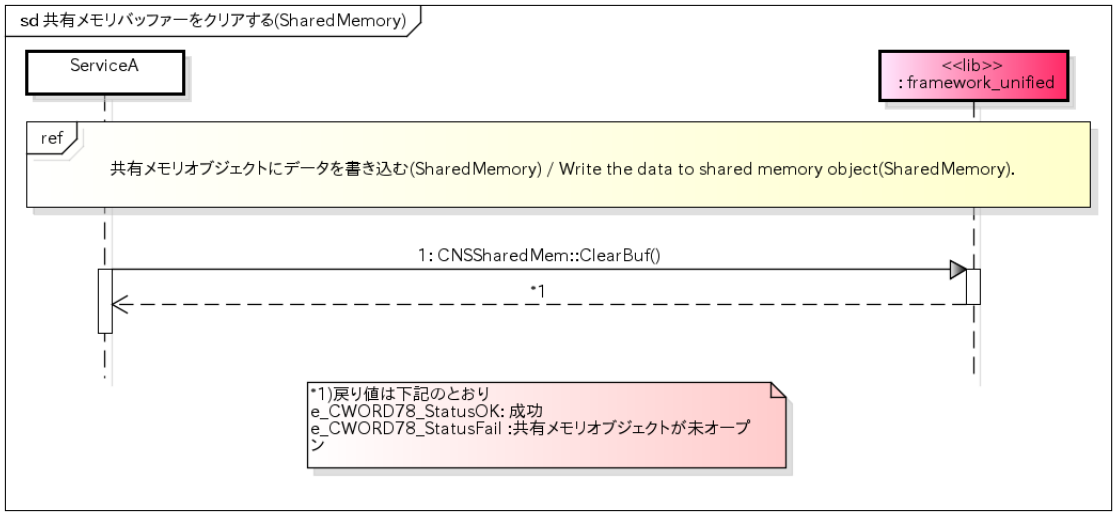
概要 *[Overview]*

CNSSharedMemクラスを用いて共有メモリバッファーをクリアする。

Clear the shared memory buffer by using CNSSharedMem class.

シーケンス *[Sequence]*

SharedMemory



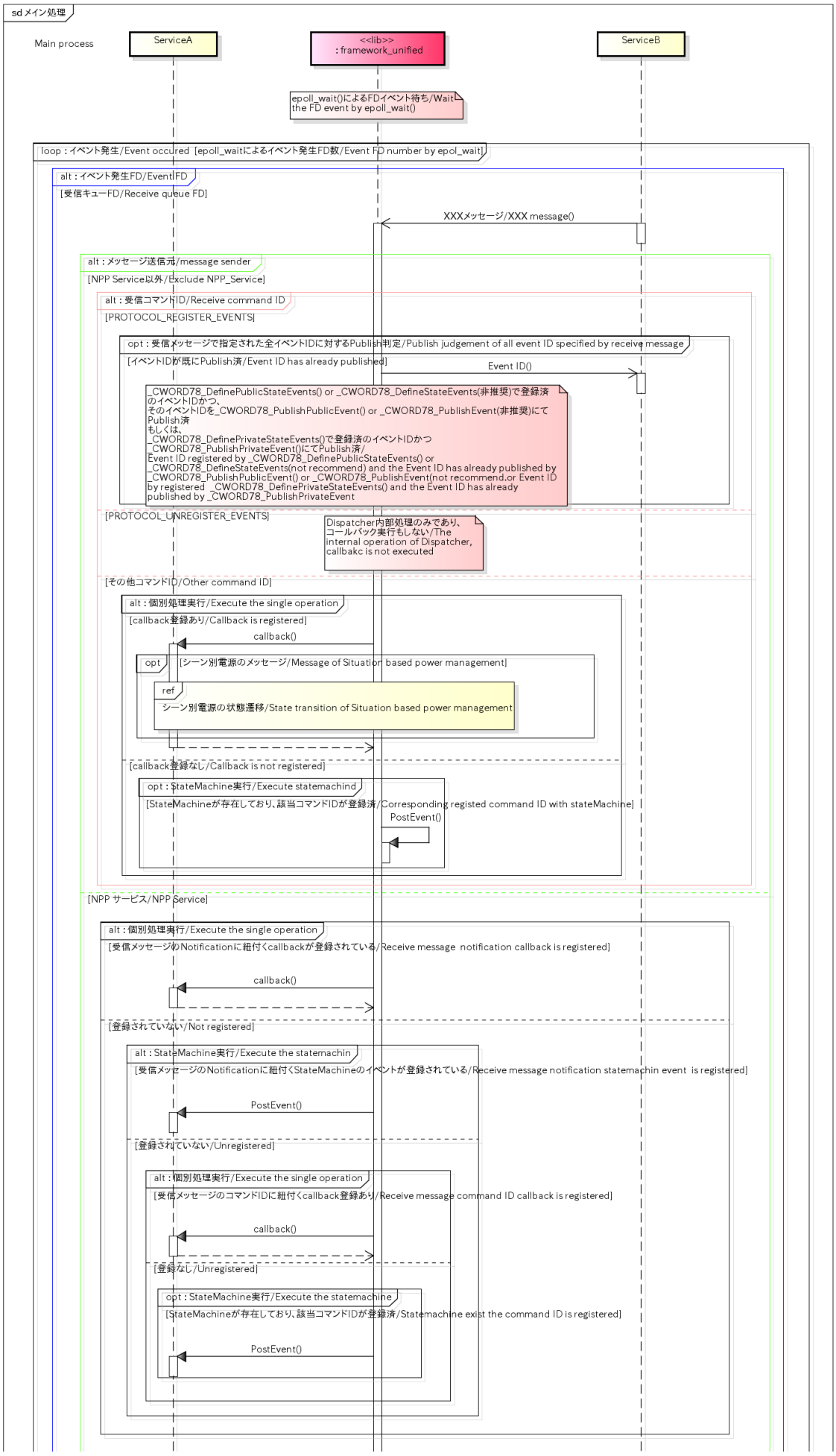
メイン処理 / *Main process*

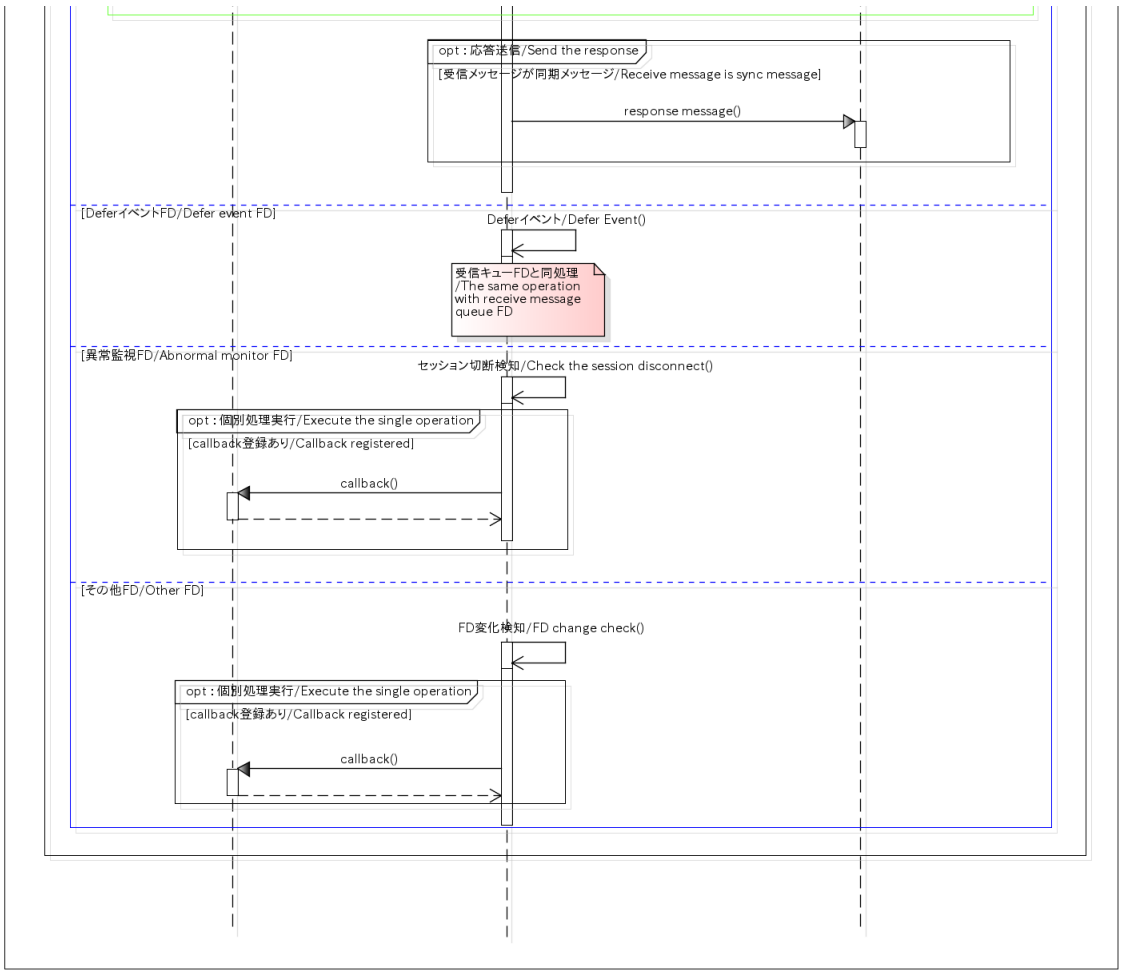
概要 *[Overview]*

メッセージを受信し、コールバックの実行を行う。

receive the message and execute register callback.

シーケンス *[Sequence]*





参照先シーケンス *[Referring sequence]*

シーン別電源の状態遷移 */ State transition of Situation based power management*

参照元シーケンス *[Referred sequence]*

サービス起動（framework_unified_Dispatcher_001）

子スレッド起動_ループなし（framework_unified_Thread_001）

子スレッド起動_ループあり(子スレッド間通信)(framework_unified_Thread_002)

HSMEvent（framework_unified_StateMachine_001）

シーン別電源の状態遷移 */ State transition of Situation based power management*

概要 *[Overview]*

シーン別電源のメッセージ受信(通常起動、乗車中起動、乗車中状態への遷移、駐乗車起動、駐乗車状態への遷移、 終了要求）とコールバックファンクションの呼び出しを行う。

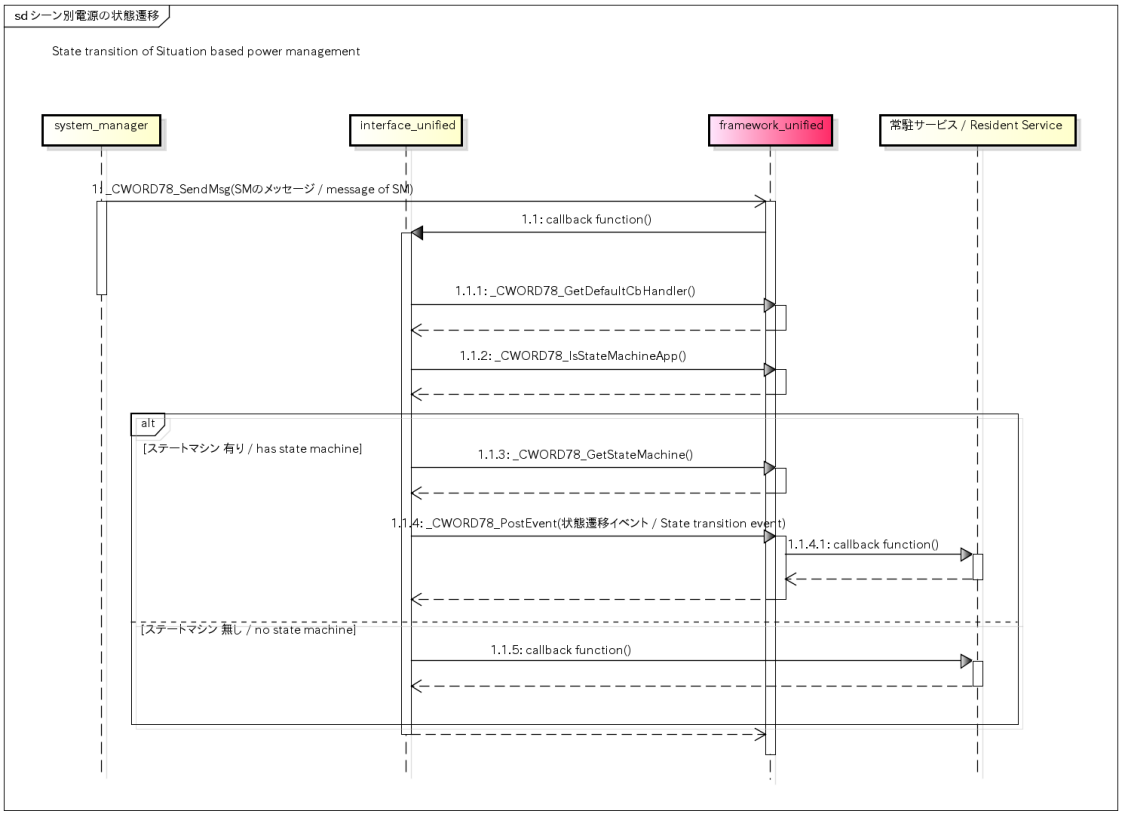
Receive power messege by scene (Nomal-startup, Riding startup, Transition for riding state, Parking and riding startup, Transition for parking and riding state and Stop request) and call callback-function.

シーン別電源のメッセージとコールバック関数の一覧 *[Message and callback function lists of Situation based power management]*

1: SMのメッセージ *Message of SM*	1.1.4: 状態遷移イベント *State transition event*	1.1.4.1: callback function 1.1.5: callback function	内容 *Contents*
SS_SM_START	ev_CWORD78_Start	_CWORD78_OnStart()	通常起動要求

			Nomal-startu
SS_SM_STOP	ev_CWORD78_Stop	_CWORD78_OnStop()	終了要求 *Stop request*
SS_SM_PRE_START	ev_CWORD78_PreStart	_CWORD78_OnPreStart()	乗車中起動要 *Riding startu*
SS_SM_PRE_STOP	ev_CWORD78_PreStop	_CWORD78_OnPreStop()	乗車中状態へ *Transition re*
SS_SM_BACKGROUND_START	ev_CWORD78_BackgroundStart	_CWORD78_OnBackgroundStart()	駐乗車起動要 *Parking and*
SS_SM_BACKGROUND_STOP	ev_CWORD78_BackgroundStop	_CWORD78_OnBackgroundStop()	駐乗車状態へ *Transition re riding state*

シーケンス *[Sequence]*



参照元シーケンス *[Referred sequence]*

メイン処理 */ Main process*