

INDUSTRIAL
EDGE
SOLUTION
WITH
HARD REALTIME
CAPABILITIES

RT-edge

Micronet.Co,

マイクロネット
Micronet

INDUSTRIAL REALTIME EDGE COMPUTERS

Container Creation Manual

RT-edge basic software Container Creation manual
コンテナ作成マニュアル






株式会社マイクロネット

<http://www.mnc.co.jp>

TEL: +81(0)299-90-1733

FAX: +81(0)299-92-8557

本書で使用するマークについて

	ノート: 操作方法や手順等の補足情報や注釈を説明しています。
	情報: 製品を利用する上で有効な豆知識となる説明をしています。
	警告: 製品仕様上注意が必要な事象について説明しています。

Windows、Visual Studio は、米国 Microsoft Corporation の米国およびその他の国における商標または登録商標です。

INtime は、米国 TenAsys Corporation の登録商標です。

TenAsys®, INtime®, eVM® and iRMX® are registered trademarks in USA of the TenAsys Corporation.

その他、本書に記載されている会社名、商品名は、各社の商標または登録商標です。

本書の内容を無断で転載することは禁止されています。

本書の内容に関しては、予告なしに変更することがあります。あらかじめご了承ください。

目次

はじめに	2
用語解説	2
関連資料	3
1. 概要	4
1.1. RT-edge とサービスコンテナ	4
1.2. RT-edge の構成パターン	6
1.2.1. Windows スタンドアロン型	6
1.2.2. ハイブリッド型	6
1.3. 標準コンテナとカスタムコンテナ	7
2. 仕様	8
2.1. 必要条件	8
2.1.1. 動作環境	8
2.1.2. 開発環境	8
3. 開発環境設定	9
3.1. 開発用ファイル	9
3.2. 開発プロジェクト作成手順	10
3.2.1. INtime(C 言語)アプリケーションにおける方法	10
3.2.2. C#アプリケーションにおける方法	13
4. カスタムコンテナの実装項目	15
4.1. はじめに	15
4.2. RT-edge フレームワークの初期化・終了処理	15
4.3. コンテナステータスインジケータ/共通タグへのデータ追加実装	16
4.3.1. コンテナステータスインジケータ	16
4.3.2. コンテナ動作プロパティ	18
4.4. RT-edge フレームワーク変数へのアクセス	21
4.4.1. 参照用タグ(TagRef)	21
4.4.2. メールボックス(Mailbox)	22
4.4.3. 参照用コレクタ(CollectorRef)	23
4.4.4. タグトリガー(TagTrigger)	25
4.4.5. メッセージ(Message)	26
4.5. コレクタを用いたタグデータ収集と取得	27
4.5.1. コレクタによるデータ収集	27
4.5.2. コレクタによって収集されたデータの取得	28
4.6. タグトリガーによるタグ値変更通知の受信	29
4.6.1. タグトリガーによるタグ値変更通知	29
4.6.2. タグトリガーによって通知されたメッセージの受信	30
4.7. メッセージオブジェクトを利用した送信	32
4.7.1. 初期化完了後にメッセージ送信	32
4.7.2. 任意のタイミングでメッセージを送信	33

4.8. ユーザメッセージハンドラ	34
4.9. サービスコンテナ間メッセージ通信(送信・受信)	35
4.10. サービスコンテナ作成時の注意事項	36
4.10.1. エラーの現示	36
4.10.2. エラーの回復	36
4.10.3. データ更新の扱い	36
4.10.4. デバッグ実行について	36
4.10.5. エラーの特定	36
5. テンプレートを用いたサービスコンテナ作成	37
5.1. サービスコンテナ作成用テンプレートのインストール	37
5.2. テンプレートを用いた実装方法	38
5.2.1. INtime(C 言語)アプリケーションにおける方法	38
5.2.2. C#アプリケーションにおける方法	41
5.3. ECI ファイルの生成とタグの定義	43
5.4. カスタムコンテナ開発例: デジタル I/O ボードの制御	43
6. サンプルシステム	45
6.1. サンプルシステム実行ファイルのインストール	45
6.2. サンプルシステム構成	45
6.3. ソリューション構成	46
6.4. ECI ファイル構成	47
6.5. 実行方法	50
更新履歴	52

はじめに

この度は産業用リアルタイム・エッジソフトウェアプラットフォーム「RT-edge」をご利用いただきまして有難うございます。本マニュアルは、RT-edge システムを構成するコンテナ機能内のインターフェースや、プロセス(rta/exe)である、サービスコンテナの作成方法について説明しています。RT-edge の仕様・セットアップ方法や、RT-edge API の詳細等は「関連資料」に記載の別マニュアルを参照してください。

用語解説

本ドキュメントにおいて使用される用語・略称について説明します:

表 1. 用語集

用語	説明
RT-edge	エッジコンピューティングを軸とする IT の情報処理と、FA における装置・機器の制御を融合し、密度の高い高頻度データ利用を可能とするソフトウェアプラットフォームです。 FA で要求されるハードリアルタイム制御を組み込むことで、情報処理と機器・装置制御を可能とするエッジコントローラを構成することができます。
RT-edge 基本ソフトウェア	RT-edge 機能の核となる機能・ライブラリを実装するパッケージソフトウェア製品です。
INtime	INtime for Windows: Windows と協調動作可能なリアルタイムカーネル拡張ソフトウェアです(RTOS ソフトウェア)。 INtime Distributed RTOS(dRTOS): Windows OS を必要とせず、スタンドアロンで動作するリアルタイム OS です。
RTA	RealTime Application: リアルタイムアプリケーションの略称。INtime 上で動作するロードブルプロセスの拡張子です。INtime 上で動作するロードブルアプリケーションは、RTA という拡張子を持ちます。
RSL	Realtime Shared Library: リアルタイム共有ライブラリの略称。INtime 上でアプリケーションがロード可能なライブラリです。Windows 上で使用される DLL(Dynamic Link Library)のようなものです。RTA から使用されるライブラリインタフェース等は、こちらを使用して作成することができます。
API	Application Programming Interface: アプリケーションプログラミングインタフェースの略称。RT-edge ではデバイスへのアクセスインタフェースとして API ライブラリを提供しています。
NTX	INtime's Windows NT extension API: INtime 用 Windows NT 拡張 API の略称。NTX 関数は Windows プログラムが INtime リアルタイム環境上で実行するリアルタイムプログラムと通信を可能とする関数セットです。
産業機器通信インターフェース	各種フィールドバス経由で機器、装置との通信、もしくは直接入出力デバイスの制御を行うインターフェースです。本インターフェースを介し、センサー値の参照やアクチュエータ制御が可能です。
サービスコンテナ/EgService	RT-edge システムを構成する機能プロセス(rta/exe)です。
タグ/EgTag	瞬時値データ値 1 つを示すオブジェクトです。ユニーク名とグローバルなスコープを持ち、全ての EgService から読み書きが許されたオブジェクトです。タグは生成時にデータ型が確定され変更はできません。
リンクタグ	同一名称のタグを重複生成した場合に自動的に別名称で生成されるタグを指します。 通常のタグと同様、グローバルなスコープを持ち、全ての EgService から読み書きが許されたオブジェクトです。一つのタグに対し、異なるプロパティ情報を定義したい場合に使用します。
データセット/EgDataset	タグ 1 つ以上の組み合わせでデータ並び順(データ構造)を定義する名前付きオブジェクトです。
コレクタ/EgCollector	データセットに定義されたデータ構造に従って、同時刻のバイナリデータ列で生成し、デー

用語	説明
	タレコードとしてメールボックスに送信するオブジェクト（スレッド）です。
メールボックス/EgMailBox	時系列なデータセット、または時系列メッセージを FIFO で蓄えることができ、また受信イベントとして処理できるオブジェクトです。
タグ参照/TagRef	タグの参照として使用するオブジェクトです。タグの名前を保持し値は保持しません。RT-edge コンテナ設定情報(ECI)ファイルでデータセットの収集用タグとして定義することや、サービスコンテナ内のオブジェクトとして定義することでサービスコンテナのメンバ変数として使用することができます。
コレクタ参照/CollectorRef	コレクタの参照として使用するオブジェクトです。コレクタの名前を保持しそれ以外のオブジェクトは保持しません。RT-edge コンテナ設定情報(ECI)ファイルでサービス内のオブジェクトとして定義することでサービス内のメンバ変数として使用することができます。
タグトリガー/TagTrigger	タグトリガーとは特定のタグの値が変化した場合に、サービス側でメッセージ通知を受けることができる機能です。サービスの ECI ファイルでタグトリガーとして登録したいタグ名を記載することで、タグ値の変更通知を受け取ることができます。
メッセージ	メッセージとは、一般的にある人や機器、ソフトウェアなどから、別の主体へ伝えようとする内容や、それを一定の形式のデータとして表現したものをさします。 RT-edge では、メールボックスで扱われる 1 レコード分のデータ、またはサービス間のコマンド、応答の電文のことを意味します。
RT-edge フレームワーク	フレームワークとは、一般的にアプリケーション開発時の土台として機能させるソフトウェアや仕組みのことをさします。 RT-edge では、アプリケーションが API を組み合わせて実装する中で、よく利用する処理についてマクロ化、自動化したもので RT-edge コンテナ設定情報(ECI)ファイルの記述により自動処理させることができます。
RT-edge コンテナ設定情報 (ECI)	RT-edge コンテナがタグに展開する入出力データ定義の他、RT-edge フレームワークが、オブジェクト生成やコンテナサービス等自動処理するための定義設定情報(XML 型式)。
入力	RT-edge システムを中心に見た場合、外部の情報を RT-edge システムへ取り込む方向性のデータの流れを意味します。
出力	RT-edge システムを中心に見た場合、RT-edge システムが持つデータを外部に書き出す方向性のデータの流れを意味します。
RTCD	Realtime Common Data の略称。RT-edge システム上で最もベースとなる共有データ構造機能です。
RT-edge Object	RT-edge システム上で使用可能なオブジェクト群（機能群）の総称です。 例えば、センサーや装置から収集したデータをアプリケーション間で受け渡しを行う場合に使用するタグ、アプリケーション間でメッセージのやり取りを行う場合のメールボックス等、アプリケーション間でデータの受け渡しを行うケースにおいて利用されるオブジェクトです。 RT-edge Object は Windows アプリケーション間、INtime®アプリケーション間、Windows-INtime®アプリケーション間いずれの場合も利用可能です。

関連資料

RT-edge 製品に含まれる資料

表 2 .RT-edge 関連資料

名称	ファイル名	内容
RT-edge ユーザーズマニュアル	DOCRTEGEUSER.pdf	RT-edge システム全般的な内容の説明が記載されています。
RT-edge API リファレンス	DOCRTEGEAPI.pdf	RT-edge API の使用方法が記載されています。
インストール手順書	インストール手順書.pdf	RT-edge 実行/開発環境のインストール手順が記載されています。

1. 概要

1.1. RT-edge とサービスコンテナ

RT-edge とは、エッジコンピューティングを軸とする IT の情報処理と、FA における装置・機器の制御を融合し、密度の高い高頻度データ利用を可能とするソフトウェアプラットフォームです。

RT-edgeの利用により、装置やセンサーからの高密度なデータ収集、分析だけでなく、提供される開発ライブラリキットを使用し、タグデータをレジスタとした機器制御を行うハードリアルタイムエッジアプリケーションの開発が可能です。

サービスコンテナ

RT-edge の処理ターゲットは、エッジコンピューティングを軸とした IT 情報処理(IT-Process)と、ミリ秒精度のハードリアルタイム性を要求される FA 制御(FA-Control)に分類され、ターゲットの機能に特化した専門処理サービスをコンテナ(サービスコンテナ)と呼びます。

IT 情報処理ターゲットは上位層にあり、主に外部システムからの要求指示の受付や、外部システムへのデータ公開、通信等を担う要素となります。IT 情報処理サービスコンテナは、制御システムのコンソール画面や外部システムから WEB ブラウザ経由でのアクセス機能、制御データ情報を外部クラウドストレージに保存する機能等、上位システムとの接続・インターフェースを提供します。

一方、FA 制御ターゲットは下位層に位置し、主に通信やハードウェアへの直接 I/O 入出力等により装置・機器制御を担う要素です。FA 制御サービスコンテナは、産業用フィールドバスやコントローラ通信プロトコルによるロボット制御、計測機器からのデータロギング、デジタルパルス出力等、装置・機器へのアクセスを提供します。

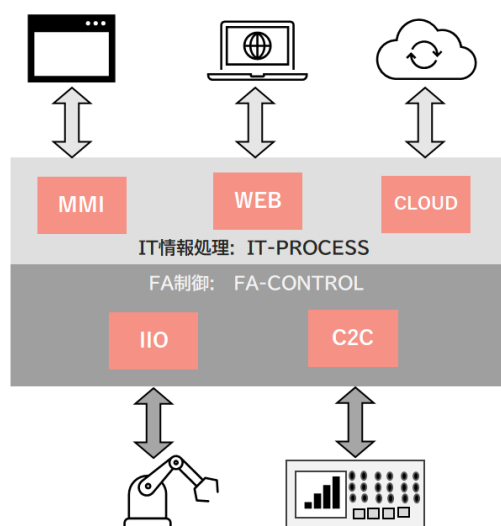


図 1. ターゲットとコンテナ

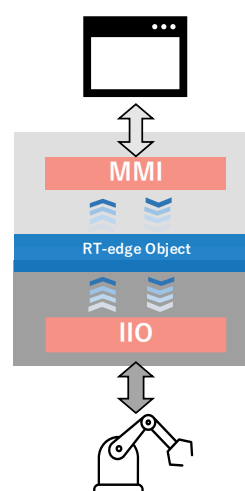


図 2. コンテナの役割



サービスコンテナ例

WEB: IT 情報処理ターゲット内には、IIS(Internet Information Service)を介し、制御情報をインターネット上に公開する WEB サービスコンテナ
 IIO: ロボットアーム制御に特化した産業 I/O サービスコンテナ

サービスコンテナはターゲットに特化した入出力データを RT-edge Object であるシステム内でグローバルにアクセス可能なタグ情報としてリンクし、このタグ情報のコレクションを公開します。

サービスコンテナは、タグ情報コレクションや、動作・挙動を決定するパラメータ設定と、ターゲット処理に特化した一つ以上の実行処理の集合体です:

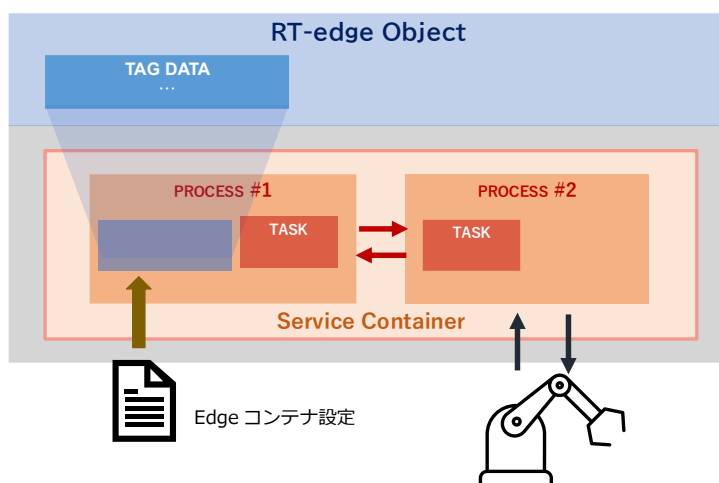


図 3. サービスコンテナの構造

サービスコンテナ例

PROCESS #1

RT-edge Object を使用する処理プロセス (サービスハンドラ)

PROCESS #2

ターゲット特化処理プロセス

Edge コンテナ設定情報

タグデータコレクション等コンテナ設定

1.2. RT-edge の構成パターン

RT-edge の構成パターンとして、Windows のみで構成されるスタンドアロンと Windows と INtime で構成されるハイブリッド型の 2 つがあります。

1.2.1. Windows スタンドアロン型

Windows 上で IT-Process と FA-Control の両方を行う構成です。特徴として以下のものが挙げられます。

- 1) ソフトリアルタイム処理向き
- 2) 簡易的な開発・実行が可能
- 3) INtime 不要

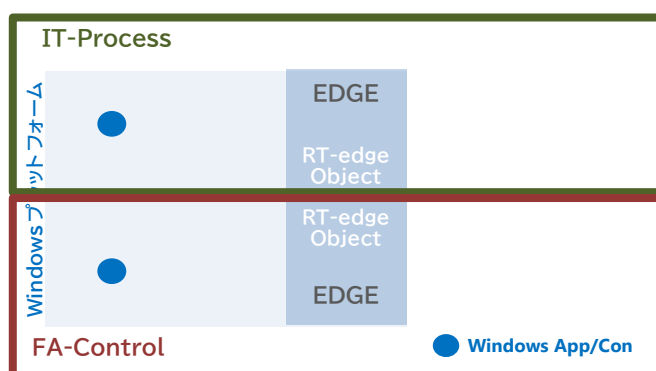


図 4 Windows スタンドアロン型構成イメージ

1.2.2. ハイブリッド型

Windows と INtime を組み合わせで成され、IT-Process を Windows にて、リアルタイム性が重視される FA-Control を INtime にて行います。特徴としては以下のものが挙げられます。

- 1) ハードリアルタイム処理向き
- 2) RTA/RSI と連携可能
- 3) 直接デバイス制御が可能

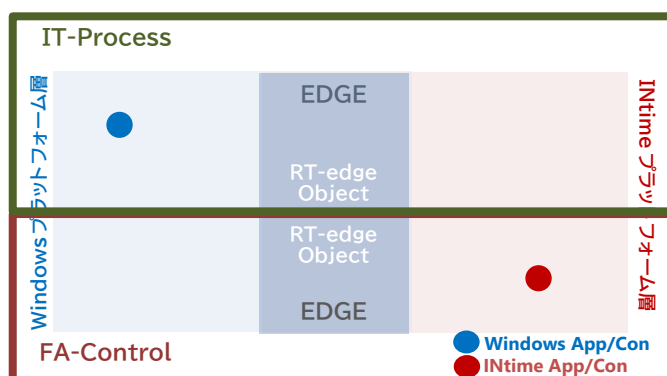


図 5 ハイブリッド型構成イメージ

1.3. 標準コンテナとカスタムコンテナ

RT-edge のサービスコンテナには標準コンテナとカスタムコンテナがあります。

標準コンテナは特定の機能を持つ、あらかじめ用意されたサービスコンテナを指し、マイクロネットが提供しています。設定ファイルを変更することで利用できるものから、API の形で提供され、開発要素を含むものがあります。

カスタムコンテナはユーザが開発するサービスコンテナであり、edge API を利用して作成するコンテナを指します。

カスタムコンテナの開発や、必要なサービスコンテナの組み合わせで任意のアプリケーションシステムの構築が可能です。

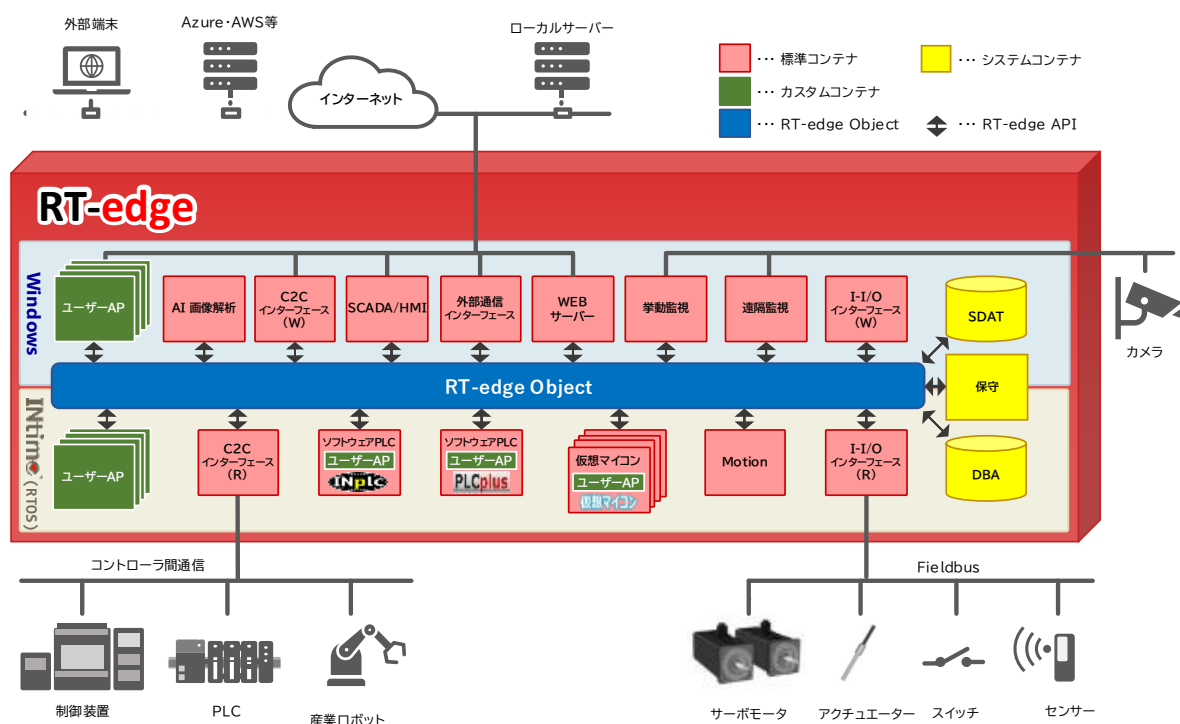


図 6 RT-edge 全体構成

本マニュアルでは、カスタムコンテナを作成する手順について説明します。

2. 仕様

2.1. 必要条件

サービスコンテナの動作・開発環境は、以下のとおりです。

2.1.1. 動作環境

Windows:

Windows 10 以降

INtime: (INtime for Windows 上で動作するサービスコンテナ利用時) :

- 1) INtime version 6.4.21125.1 以降
- 2) トレーサブルコントローラ 6.4.21125.1 以降

その他:

.NET Framework 4.6

2.1.2. 開発環境

Visual Studio

Visual Studio 2017

INtime SDK (INtime for Windows 上で動作するサービスコンテナ作成時) :

- 1) INtime version 6.4.21125.1 以降
- 2) トレーサブルコントローラ 6.4.21125.1 以降

その他:

.NET Framework 4.6

3. 開発環境設定

本章では、RT-edge 基本ソフトウェアを使用したサービスコンテナ開発環境の設定からプロジェクト作成手順について説明します。本章の設定を実施することでカスタムコンテナ作成の基本となるプロジェクトが作成されます。作成したプロジェクトに対して、4 章で紹介するカスタムコンテナに必要な項目の実装、または 5 章で説明するテンプレートの適用を実施します。

3.1. 開発用ファイル

カスタムコンテナの開発には、RT-edge インストーラによって展開された RT-edge 基本ソフトウェアを使用します。



必要に応じて開発プロジェクト配下にコピー、または下記展開ディレクトリを環境変数に登録し、参照できるようにします。本マニュアルでは C ドライブ直下に開発環境コンポーネントを展開し、下記の環境変数を登録した例を記載します。

環境変数 : 「RTEDGE」 パス : 「C¥RTedge¥」

表 3 開発環境コンポーネント

フォルダ階層	ファイル名	説明
RTedge¥Library¥	edgeAPI_RTCD.h	RT-edge 開発環境 API ヘッダ
RTedge¥Library¥	egAPI.h	RT-edge 開発環境 API ヘッダ
RTedge¥Library¥Rt¥	eghgapi.lib	RT-edge 開発環境 API ライブラリ(INtime)
RTedge¥Library¥Rt¥	egosdep.lib	RT-edge 開発環境 API ライブラリ(INtime)
RTedge¥Library¥Rt¥	egRTCD.lib	RT-edge 開発環境 API ライブラリ(INtime)
RTedge¥Library¥Win¥	eghgapiWin.lib	RT-edge 開発環境 API ライブラリ(Windows)
RTedge¥Library¥Win¥	egosdepntx.lib	RT-edge 開発環境 API ライブラリ(Windows)
RTedge¥Library¥Win¥	egRTCDntx.lib	RT-edge 開発環境 API ライブラリ(Windows)

3.2. 開発プロジェクト作成手順

RT-edge 開発ライブラリを使用しカスタムコンテナを作成するには、アプリケーションビルド用の開発プロジェクトを作成後、ライブラリリンク設定を行う必要があります。カスタムコンテナには、従来のコンピュータ言語で記述されるネイティブコード(Windows C/C++言語 MFC, INtime C/C++言語)で作成する方法と.NET Framework 上で動作するオブジェクトコード(C#)で作成する方法があります。以下に INtime アプリケーション(C 言語)を使用した開発プロジェクト、C#(.NET)を使用した開発プロジェクト作成方法について説明します。

3.2.1. INtime(C 言語)アプリケーションにおける方法

以下の手順に沿ってプロジェクト設定をおこなってください（例では Visual Studio 2017 を用いています）。

手順 1

Microsoft Visual Studio を使用し、プロジェクトを作成します。

プロジェクト生成には、INtime Projects を選択後、Application Wizard を指定しプロジェクトワークスペースを生成します：

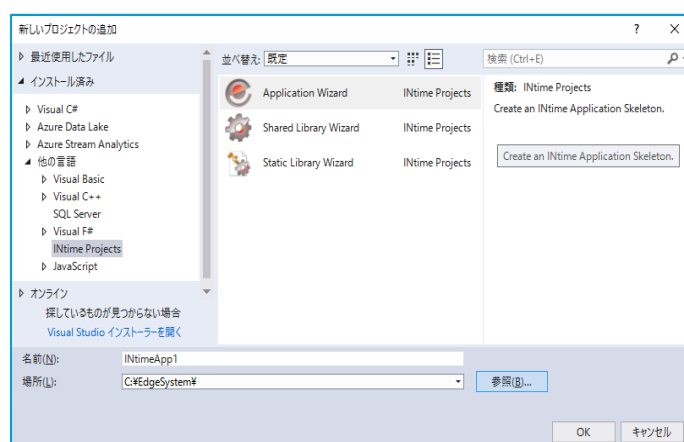


図 7. プロジェクトワークスペース生成(C/C++)

手順 2

ヘッダファイルの参照設定を行います。

[プロジェクト]メニューから[プロパティ]を選択し、プロジェクトのプロパティダイアログを開きます。[構成プロパティ]-[C/C++]-[全般]の[追加のインクルードディレクトリ]に、開発用ヘッダ定義ファイルのパスを指定します。

例: 開発ライブラリのパス (「RTEDGE」が環境変数に登録してある場合)

`$(RTEDGE)Library`

[追加のインクルードディレクトリ]欄に直接追加するか、右端リストのボタンから[編集]を選択し、追加のインクルードディレクトリダイアログからパスを設定してください:

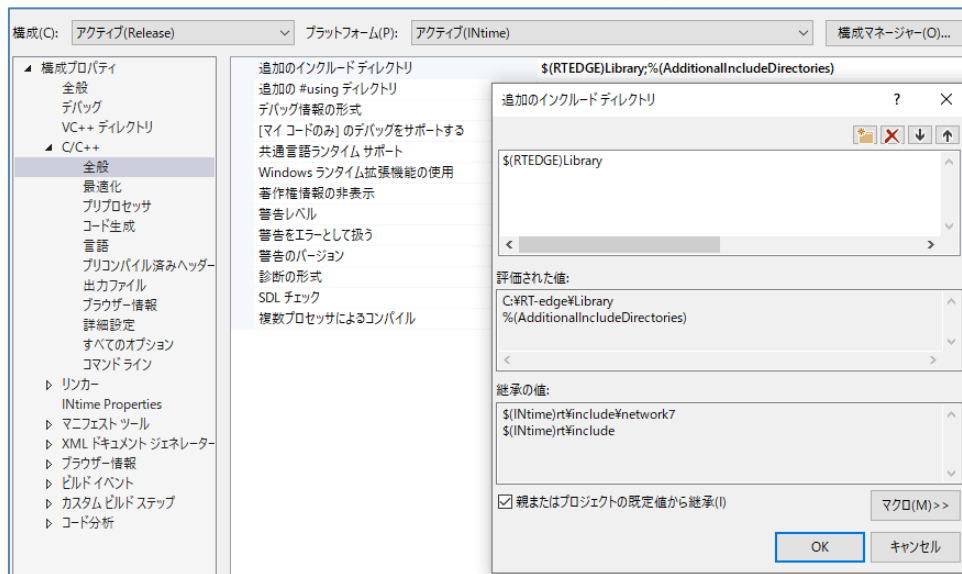


図 8 インクルードディレクトリ設定



参照パスを追加する場合、追加のインクルードディレクトリには、あらかじめウィザードにより設定されている設定情報があります。この設定は消さないでください。インクルードパスの設定は、最後に追加し、既に設定されているパスの間に設定したり、先頭に配置したりしないでください。



RT-Edge API を使用する場合は、ライブラリ関数定義が行われている egAPI.h をユーザープログラムの先頭で #include してください。

手順 3

リンクライブラリ設定を行います。

ヘッダ参照設定と同様、[プロジェクト]メニューから[プロパティ]を選択し、プロジェクトのプロパティダイアログを開きます。[構成プロパティ]-[リンカー]-[全般]の[追加のライブラリディレクトリ]に、開発リンクライブラリのパスを指定します。

例: (「RTEDGE」が環境変数に登録してある場合)

`$(RTEDGE)Library¥Rt`

[追加のライブラリディレクトリ]欄に直接入力するか、右端リストのボタンから[編集]を選択し、追加のライブラリディレクトリダイアログからパスを設定してください:

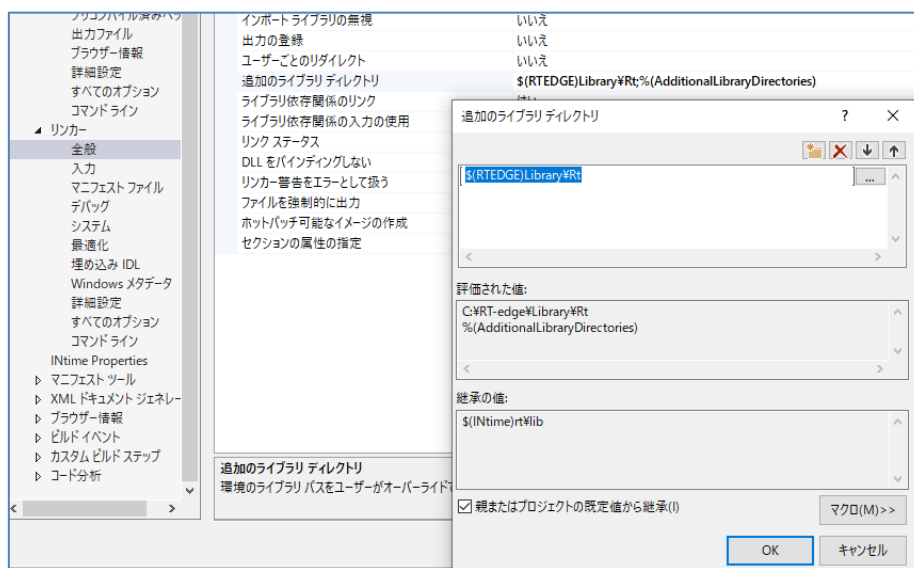


図 9 リンクライブラリ設定



参照パスを追加する場合、追加のライブラリディレクトリには、あらかじめウィザードにより設定されている設定情報があります。この設定は消さないでください。ライブラリパスの設定は、最後に追加し、既に設定されているパスの中間に設定したり、先頭に配置したりしないでください。

次に、[構成プロパティ]-[リンカー]-[入力]の[追加の依存ファイル]に製品リンクライブラリ (eghgapi.lib)を設定します。



追加の依存ファイル欄には、既に追加されているエントリがあります。リンクライブラリ名を追加する際、先に追加されているエントリを削除せず、最後に追加するようにしてください。

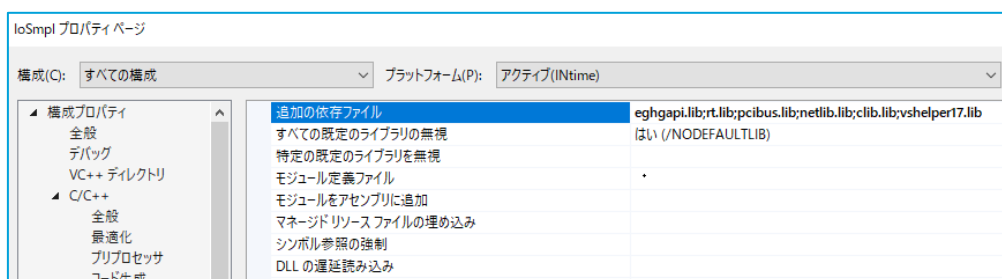


図 10 追加の依存ファイル入力

3.2.2. C#アプリケーションにおける方法

以下の手順に沿ってプロジェクト設定を行ってください。

手順 1

Microsoft Visual Studio を使用し、プロジェクトを作成します。Visual C#のプロジェクトから、プロジェクトワークスペースを生成します：

Visual C#

- Windows フォームアプリケーション
- WPF アプリケーション
- コンソールアプリケーション

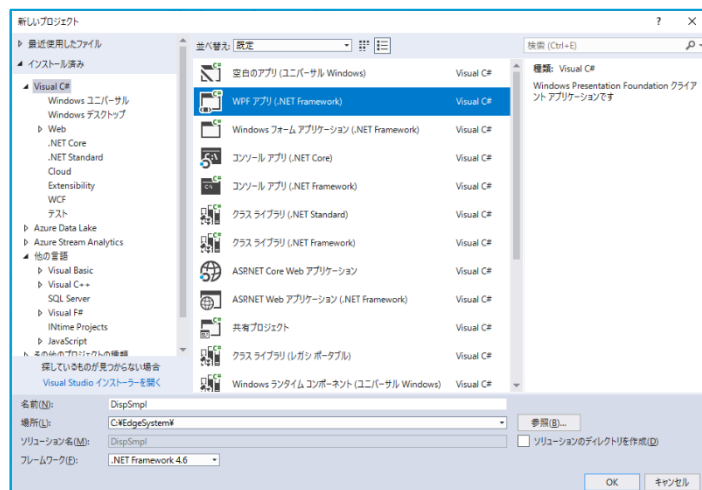


図 11 プロジェクトワークスペース生成(VC#/VB.NET)

手順 2

アセンブリの参照設定を行います。

[プロジェクト]メニューから[参照設定]を選択し、参照マネージャを起動します。

[ブラウザ]-[参照]から、ファイル選択ダイアログにて、インストールを行ったフォルダの「egapiWrap.dll」を選択します。

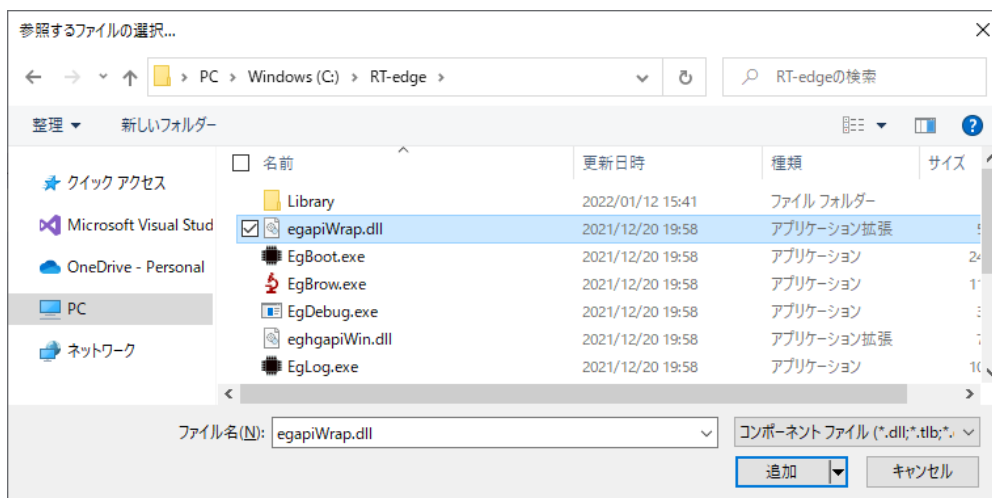


図 12 参照するアセンブリ選択

手順 3

アセンブリ追加の確認。

ソリューションエクスプローラから、参照設定を開き、追加したアセンブリが表示されていることを確認します。

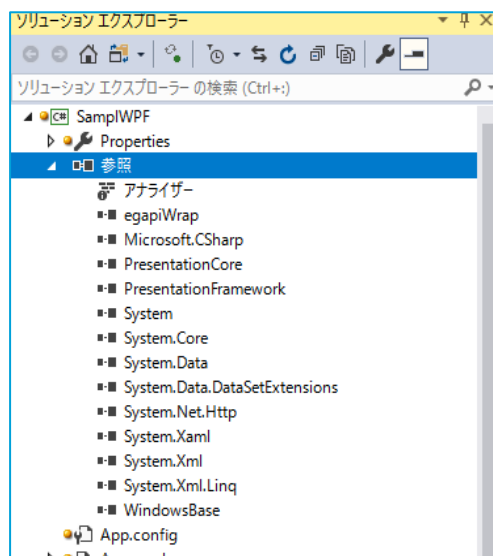


図 13 参照設定確認

手順 4

プラットフォームターゲットの設定。

プロジェクトのプロパティ設定を開き、プラットフォームターゲットを「x64」とします。

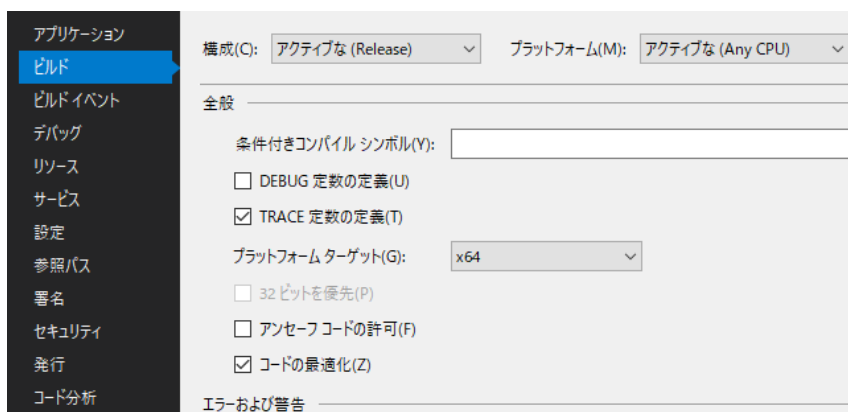


図 14 プラットフォームターゲットの設定

4. カスタムコンテナの実装項目

カスタムコンテナに実装する項目について説明します。

4.1. はじめに

サービスコンテナを容易に作成できるように、RT-edge インストーラにサービスコンテナ作成用テンプレートをご用意しています。サービスコンテナ作成用テンプレートには本章で説明する内容のうち、以下の機能を実装しています。

- 1) サービスコンテナの初期化・開始・停止・削除関数
- 2) Edge メッセージハンドラ処理関数
- 3) サービスコンテナインジケータタグ更新関数
- 4) 定周期スレッド処理ひな型

サービスコンテナ作成用テンプレートを用いることでサービスコンテナを作成する上でのコード生成短縮が見込め、上記機能を容易に使用することができます。詳細は「5. テンプレートを用いたサービスコンテナ作成」を参照してください。

4.2. RT-edge フレームワークの初期化・終了処理

サービスコンテナは EgInit API をコールすることで RT-edge フレームワークを初期化できます。RT-edge フレームワークを使用しない場合このコールを行う必要はありません。また、RT-edge フレームワークを使用した場合は、処理終了のタイミングで EgFinalize API をコールして終了処理を実行することができます。

表 4 RT-edge フレームワーク初期化/終了処理 API

API 関数	説明
EgInit	RT-edge フレームワーク初期化・開始処理
EgFinalize	RT-edge フレームワーク終了処理

上記 API の呼び出し方法は API マニュアルを、実際のコーディングにおける呼び出しのタイミングについては本書 5.2.1 手順 4 を参照してください。また、RT-edge フレームワークの概要や利点につきましてはユーザーズマニュアルを参照してください。



サンプルでは、InitService()関数内で EgInit 関数を、終了処理内で EgFinalize()関数を呼び出しています。サンプルを用いた実装の場合は、EgInit 関数に渡す設定情報ファイルを運用に沿った値に変更してください。

4.3. コンテナステータスインジケータ/共通タグへのデータ追加実装

サービスコンテナは動作状態を示すインジケータと、コンテナの動作を設定するプロパティを持っています。下図に RT-edge フレームワーク利用時のインジケータとプロパティの構成について示します。コンテナ#1の RT-edge フレームワークが ECI ファイルを読み込みこむと、記載されている通りにコンテナ動作プロパティタグを生成します。なお、コンテナステータスインジケータは ECI ファイルに記載されていなくても自動生成されます。生成されたタグは他コンテナであるコンテナ#2から参照するなどして利用可能です。

次項からはコンテナステータスインジケータおよびコンテナ動作プロパティについて説明します。

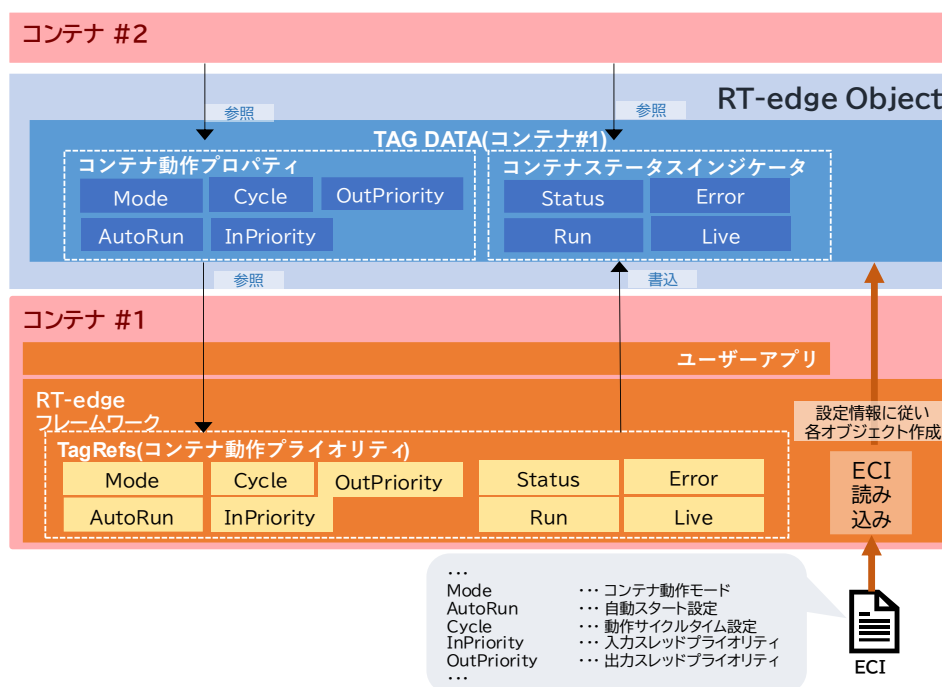


図 15 RT-edge フレームワーク利用時のインジケータ/プロパティの構成

4.3.1. コンテナステータスインジケータ

サービスコンテナは、状態を Edge システムに公開する目的で標準的に表 5 に示す Edge タグを実装します。これらはサービスコンテナ間の連携時に対象コンテナの死活判断などに利用できます。これらのタグは RT-edge フレームワークを利用することで、ECI ファイルに記述がなくても自動生成され、一部は RT-edge フレームワークが自動処理します。RT-edge フレームワークを用いないカスタムコンテナの場合も、極力以下仕様に沿うように実装することを推奨します：



RT-edge フレームワークでコンテナステータスインジケータタグを自動生成しますが、値の更新（書き込み）は自動で行いません。サービスコンテナごとに値を更新する処理を実装する必要があります。

テンプレートでは、UpdateIndicator()関数を呼び出すことでコンテナステータスインジケータタグの更新を行います。サービスコンテナのステータスが変化した場合は MYSTATE 構造体の値を変更後、UpdateIndicator()関数を呼ぶようにしてください。

表 5 コンテナステータスインジケータ Tag の一覧

インジケータ Tag 名	備考
SERVICE.servicename.Status	現在のサービス起動状態を示します
SERVICE.servicename.Error	現在のサービスエラー状態を示します
SERVICE.servicename.Run	現在のデータ更新動作の状態を示します
SERVICE.servicename.Live	サービスが健全であることを示すカウンタ



Tag 名に記載している「servicename」は、サービスコンテナ名を指します。サービスコンテナ名は、ECI ファイルの Name プロパティで指定した名前です。初期化関数 EgInit 呼び出し後、ServiceRealName 変数を用いることで取得できます。

表 5 に示した Edge タグの内容と設定値について表 6 から表 9 に示します。

表 6 サービスコンテナ起動状態ステータス (.Status)

Tag		備考
Name	SERVICE.servicename.Status	現在のサービスコンテナ起動状態を示します
Type	uint8	
値	意味	備考
0	サービスコンテナ未起動	EgFinalize コールにより RT-edge フレームワークが設定します
1	サービスコンテナ起動済	EgInit コールにより RT-edge フレームワークが設定します

表 7 サービスコンテナ異常状態ステータス (.Error)

Tag		備考
Name	SERVICE.servicename.Error	現在のサービスコンテナの状態を示します
Type	bool	
値	意味	備考
0 (false)	エラー無し	
1 (true)	エラー発生中	サービスコンテナに API エラー、通信エラー等データ更新障害が生じているときに設定します

表 8 サービスコンテナ実行状態ステータス (.Run)

Tag		備考
Name	SERVICE.servicename.Run	現在のデータ更新動作の状態を示します
Type	bool	
値	意味	備考
0 (false)	データ更新停止中	データ更新を行わない状態。サービスコンテナはメッセージ EM_SERVICE_PAUSE 受信時に設定する必要があります
1 (true)	データ更新活性化中	データ更新可能な状態。サービスコンテナはメッセージ EM_SERVICE_RUN 受信時に設定する必要があります

表 9 サービスコンテナ実行カウンタ (.Live)

Tag		備考
Name	SERVICE.servicename.Live	サービスコンテナが健全であることを示すカウンタ
Type	uint32	
値	意味	備考
0～0xffffffff	サービスコンテナ動作	サービスコンテナプロセスの健全性確認手段として用意されています。この値を変化させることでサービスコンテナの動作が正常に行われていることを表現します。

表 11 サービスコンテナデータ更新動作自動スタート指定 (.AutoRun)

Tag		備考
Name	SERVICE.servicename.AutoRun	サービスコンテナ起動時にデータ更新を自動開始します
Type	Bool	
値	意味	備考
0 (false)	データ更新自動開始無効設定	初期値
1 (true)	データ更新自動開始有効設定	RT-edge フレームワークからメッセージ EM_SERVICE_RUN が届きます。サービスコンテナはメッセージハンドラを実装してデータ更新状態(.Run = true)となるよう処理します

表 12 サービスコンテナ動作サイクルタイム指定 (.Cycle)

Tag		備考
Name	SERVICE.servicename.Cycle	サイクル駆動型サービスコンテナの動作周期の設定値
Type	uint32	
値	意味	備考
0~0xffffffff	サービスコンテナ動作周期設定値	初期値:0 単位に規定はなくサービスコンテナ固有の定義になります

表 13 サービスコンテナ入力動作スレッドプライオリティ指定 (.InPriority)

Tag		備考
Name	SERVICE.servicename.InPriority	データ更新スレッド(入力)の優先度設定値
Type	uint8	
Value	0	初期値
値	意味	備考
0~255	スレッド入力処理優先順位	数値が小さいほど優先的になります

表 14 サービスコンテナ出力動作スレッドプライオリティ指定 (.OutPriority)

Tag		備考
Name	SERVICE.servicename.OutPriority	データ更新スレッド(出力)の優先度設定値
Type	uint8	
Value	0	初期値
値	意味	備考
0~255	スレッド出力処理優先順位	数値が小さいほど優先的になります



プロパティ値の読み込み処理

サービスコンテナは、ECI ファイルに記載された通りにプロパティ値の読み込みを実行します。ECI ファイルにプロパティ値を定義することで読み込んだ値による状態制御を行うことができます。なお、プロパティの読み込みに失敗した場合に備えて、デフォルト動作をサービスコンテナ内に記述することを推奨します。

EgInit API をコールすることで、サービスコンテナに読み込まれたプロパティタグが自動生成されます。

下の例ではサービスコンテナの動作モードを読み込む処理について記載します。

■ECI ファイル (myService.xml)

Tag エlement、Value プロパティが設定値になります

```
<Tags>
  <Tag Name="SERVICE.myService.Mode" Type="3" Value="2" Comment="myService Operation mode" />
</Tags>
```

■サービスコンテナ処理の実装(myService.c)

```
#define SERVICE_PROP_MODE "SERVICE.myService.Mode"
#define DEFAULT_MODE 1
BYTE byMode;
int32_t status; // サービスコンテナの動作モード

status = EgTagRead(SERVICE_PROP_MODE, &byMode, sizeof(byMode)); // 動作モードを ECI ファイルから読み込
if( status!=EDGE_SUCCESS ) byMode=DEFAULT_MODE; // 動作モード読み込み失敗時のデフォルト値
```



上記例では、#define にて“myService”というサービスコンテナ名を固定値で記述しておりますが、初期化関数(EgInit)呼び出し後、ServiceRealName 変数を用いることで、ECI ファイルに記載のサービスコンテナ名の取得が可能です。

4.4. RT-edge フレームワーク変数へのアクセス

RT-edge フレームワーク変数とは、ECI ファイルに定義された参照用タグ (①)・メールボックス (②)・参照用コレクタ (③)・タグトリガー(④)・メッセージ(⑤)オブジェクトを、サービスコンテナからアクセスできるように、グローバル変数として公開したものです。EgInit API をコールし RT-edge フレームワークを初期化することで、ECI ファイルに定義された各オブジェクトを RT-edge フレームワーク共通変数へ展開します。

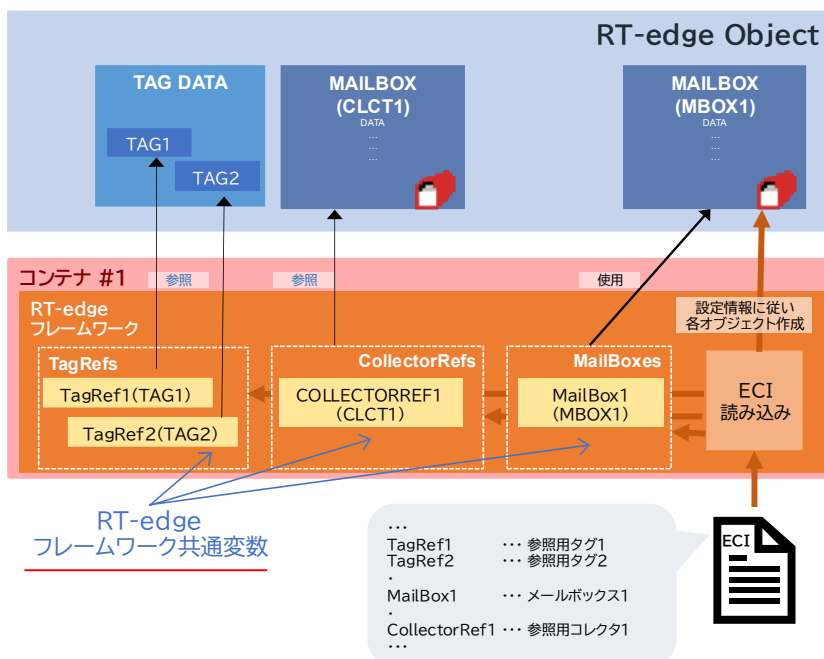


図 17 RT-edge フレームワーク変数イメージ図

各オブジェクトに対する RT-edge フレームワーク変数の詳細について説明します:

4.4.1. 参照用タグ(TagRef)

ECI ファイルで定義した参照用タグ(TagRef)オブジェクトは、以下の RT-edge フレームワーク変数としてアクセスすることができます:

1) C/C++

- ① int32_t egTagRefs_OUT_Count : Output 用 EgTagRef の配列数
- ② int32_t egTagRefs_IN_Count : Input 用 EgTagRef の配列数
- ③ EGTAGREF egTagRefs_OUT : Output 用 EgTagRef の配列
- ④ EGTAGREF egTagRefs_IN : Input 用 EgTagRef の配列

EDGECONFIG 構造体

```
typedef struct
{
    char    Name[EG_MAX_TAGNAME+1];        // EgTag 名
    char    Address[EG_MAX_TAGADDRESS+1];  // アドレス(ソース)
    uint16_t Type;                          // データ型
    uint16_t Size;                          // データサイズ
} EGTagRef;                                // 構造体定義
```

2) .NET(C#)

- ① List<EgHgTagRef> tagRef_OUT_List : Output 用 EgTagRef のリスト
- ② List<EgHgTagRef> tagRef_IN_List : Input 用 EgTagRef のリスト

EgHgTagRef 構造体

```
public class EgHgObject
{
    public string Name;        //名前
    ...
}
/// <summary>
/// Service で保持する Tag の参照
/// </summary>
public class EgHgTagRef : EgHgObject
{
    public ushort Size;        //データサイズ
    public ushort Type;        //データタイプ
    public string Address;     //アドレス情報
    ...default
}
```

4.4.2. メールボックス(Mailbox)

ECI ファイルで定義したメールボックス(Mailbox)オブジェクトは、以下の RT-edge フレームワーク変数としてアクセスすることができます:

1) C/C++

- ① int32_t : egMailboxes_Count : Output 用 egMailboxes の配列数
- ② EGMAILBOX : egMailboxes : メールボックス のリスト

EDGECONFIG 構造体

```
typedef struct
{
    char    MailBoxName[EG_MAX_MAILBOXNAME+1];    // メールボックス名
    uint32_t dwColSize;                            // 1レコード長
    uint32_t dwRowSize;                            // レコード数
    RTCDMEMINFO MemInfo;                          // RTCD の構造体 (RTCDMEMINFO)
} EGMAILBOX;
```

2).NET(C#)

List<EgHgMailBox> mailBoxList : EgHgMailBox のリスト

EgHgObject/EgHgMailBox 構造体

```
public class EgHgObject
{
    public string Name;           //名前
    ...
}
public class EgHgMailBox
{
    public uint ColSize;          //1 データ列数
    public uint RowSize;          //1 データ行数
    public edge_API.RTCMEMINFO stMemInfo;
    public EgHgMailBox(){...}
}
```

4.4.3. 参照用コレクタ(CollectorRef)

ECI ファイルで定義した CollectorRef オブジェクトは、以下の変数としてアクセスすることができます:

1) C/C++

- ① int32_t : egCollectors_Count : egCollector (実態) の配列数
- ② int32_t : egCollectorsRefs_Count : egCollectorRef (参照) の配列数
- ③ EGLOCALCOLLECTOR egCollectors[] : egCollector (実態) の配列
- ④ EGCOLLECTORREF egCollectorsRefs[] : egCollectorRef (参照) の配列

egCollector (実態) は、スレッド内でデータ収集処理を行います。

EGLOCALCOLLECTOR/EGCOLLECTORREF 構造体

```
// Collector オブジェクト
typedef struct
{
    char    CollectorName[EG_MAX_CLCTNAME+1];    // コレクタ名
    char    OwnerServiceName[EG_MAX_SERVICENAME+1]; // コレクタ実行サービス名
    char    DatasetName[EG_MAX_DSETNAME+1];    // データセット名
    uint32_t dwInterVal;                        // 収集周期(ms)
    uint8_t  byPriority;                        // スレッドプライオリティ
    uint32_t dwStackSize;                      // スタックサイズ(現状固定)
    uint32_t dwColSize;                        // 格納する1レコードのサイズ(デフォルト 2048)
    uint32_t dwRowSize;                        // 格納数
    void*    hHandle;                          // スレッドハンドル
    uint8_t  bWorkingF;                        // FALSE:停止中 , TRUE:動作中
    uint8_t  bCancelF;                         // TRUE:停止フラグ
} EGLOCALCOLLECTOR;

// CollectorRef オブジェクト
typedef struct
{
    char    CollectorName[EG_MAX_CLCTNAME+1];    // コレクタ名
    char    DatasetName[EG_MAX_DSETNAME+1];    // データセット名(Collector オブジェクト値)
    uint32_t dwColSize;                        // 格納する1レコードのサイズ(Collector オブジェクト値)
    uint32_t dwRowSize;                        // 格納数(Collector オブジェクト値)
} EGCOLLECTORREF;
```

2) .NET(C#)

List< EgHgCollectorRef> clctRefList : Output 用 EgTagRef のリスト

EgHgObject/EgHgCollectorRef 構造体

```
public class EgHgObject
{
    public string Name;                //名前
    ...
}
public class EgHgCollectorRef : EgHgObject
{
    public uint ColSize;                //1 データサイズ
    public uint RowSize;                //データ数
    public string DatasetName;          //データセット名
    public EgHgCollectorRef(){...}      //名前
}
```

4.4.4. タグトリガー(TagTrigger)

ECI ファイルで定義したタグトリガー(TagTrigger)オブジェクトは、以下の RT-edge フレームワーク変数としてアクセスすることができます:

1) C/C++

- ① int32_t egTagTriggers_Count : TagTrigger の配列数
- ② EGTAGTRIGGER* egTagTriggers : TagTrigger の配列

EGTAGTRIGGER 構造体

```
typedef struct
{
    char    Name[EG_MAX_TAGNAME+1];        // EgTag 名
    char    Address[EG_MAX_TAGADDRESS+1];  // アドレス(ソース)
    void*    Entry;                        // EgTag の Entry ポインタ
    uint16_t Type;                         // データ型
    uint16_t Size;                         // データサイズ
    BYTE     Count;                        // 内部プログラムで使用
} EGTAGTRIGGER;                          // 構造体定義
```

2) .NET(C#)

- ① List< EgHgTagTrigger > tagTriggerList : TagTrigger のリスト

EgHgTagRef 構造体

```
public class EgHgObject
{
    public string Name;        //名前
    ...
}
/// <summary>
/// Service で保持する TagTrigger
/// </summary>
public class EgHgTagTrigger: EgHgObject
{
    public ushort Size;        //データサイズ
    public ushort Type;        //データタイプ
    public string Address;     //アドレス情報
    public IntPtr Entry;       //エントリーポインタ
    public byte Count;         //内部プログラムで使用
}
```

4.4.5. メッセージ(Message)

ECI ファイルで定義したメッセージ(Message)オブジェクトは、以下の RT-edge フレームワーク変数としてアクセスすることができます:

1) C/C++

- ① `int32_t egMessages_Count` : Message の配列数
- ② `EGMESSAGEOBJ* egMessageObjs` : Message の配列

EGTAGTRIGGER 構造体

```
typedef struct
{
    char    Name[EG_MAX_MESSAGE_NAME+1];    // メッセージ名(システム内でゆーにな名前)
    uint32_t No;                            // メッセージ No
    char    DestName[EG_MAX_SERVICENAME+1]; // 送信先サービス名
    BYTE    ExecTiming;                     // (XML で実行時に使用) 実行タイミング
    BYTE    Timer;                          // (XML で実行時に使用) 実行タイミング
    BYTE    ArgType;                         // (XML で実行時に使用) Args の Type(Tag のタイプと一緒に)
    BYTE    Size;                            // 引数のサイズ
    BYTE    Args[EG_MAX_MESSAGE_ARGS+1];    // 引数
} EGMESSAGEOBJ;                            // 構造体定義
```

2) .NET(C#)

- ① `List< EgHgMessageObj> messageObjList` : Message のリスト

EgHgMessageObj 構造体

```
public class EgHgObject
{
    public string Name;    //名前
    ...
}
/// <summary>
/// Service で保持する Message
/// </summary>
public class EgHgMessageObj: EgHgObject
{
    public UInt32 No;      // メッセージ No
    public string DestName; // 送信先サービス名
    public byte    ExecTiming; // (XML で実行時に使用) 実行タイミング
    public byte    Timer;      // (XML で実行時に使用) 実行タイミング
    public byte    ArgType;    // (XML で実行時に使用) Args の Type(Tag のタイプと一緒に)
    public UInt32 Size;        // 引数のサイズ
    public byte[]  Args;       // 引数
}
```

4.5. コレクタを用いたタグデータ収集と取得

定周期で複数の EgTag データを収集する場合は、コレクタ(EgCollector)を用いると容易に実現できます。コレクタとは ECI ファイルに記載されたデータ構造に従って、同時刻のバイナリデータ列を生成し、データレコードとしてメールボックスに送信する機能です。

4.5.1. コレクタによるデータ収集

データ収集は ECI ファイルにコレクタの定義を行うことで、RT-edge フレームワーク内で自動収集処理を行います。以下にコレクタを定義した ECI ファイルのサンプルを記載します。

例)

- 1) データセット「Dset001」は、タグ"Temp001","Temp002","Temp003" (計 3 個)を参照
- 2) コレクタ「Clct001」は、データセット「Dset001」のデータを収集する
- 3) その他「MyServ」は、コレクタ「Clct001」を 1000ms 周期で実行する

```
...
<RTedge>
  <Tags>
    <Tag Name="Tag001" Type="6" />
    <Tag Name="Tag002" Type="10" />
    <Tag Name="Tag003" Type="11" />
  </Tags>
  <Datasets>
    <Dataset Name="Dset001">
      <TagRefs>
        <TagRef Name="Tag001"/>
        <TagRef Name="Tag002"/>
        <TagRef Name="Tag003"/>
      </TagRefs>
    </Dataset>
  </Datasets>
  <Services>
    ...
    <Service Name="MyServ" Path="MyServ.exe">
      <Collectors>
        <Collector Name="Clct001" Interval="1000" DatasetName="Dset001"/>
      </Collectors>
    </Service>
  </Services>
</RTedge>
```

4.5.2. コレクタによって収集されたデータの取得

コレクタによって収集されたデータを取得する際は RT-edge API を用います。以下に例として C# でのコード(サンプルシステム格納コード)を下記に記載します。

コレクタデータ取得例(C#)

```
...
while (true)
{
    byte[] data = new byte[2048];
    //コレクタで格納されたメールボックスから値を取得し、
    //自身の保持コレクション変数に格納 参照元メールボックスは collectorRef を
    //ECI ファイルに記載し下記のようにアクセスする
    int ret = EGAPI.EgMailboxReceive(EGAPI.EGFW.clctRefList[0].Name,
                                     ref sendName, ref messageNo, ref data, 100);
    if (ret == 0)
    {
        //メールボックスから取得したデータは byte 配列の為
        //バイト配列をコレクタで集めた Dataset の情報を元に分解する
        //下記 GraphData クラスは独自処理
        GraphData oneData = new GraphData();
        ret = DecodeMailData(EGAPI.EGFW.clctRefList[0].DatasetName,
                             data, ref oneData);
        //グラフ表示用コレクションにデータを格納
        if (ret != 0 || EgClctCollection.TryAdd(oneData, 1) == false)
            break;
    }
    else
        break;
}
...
```



メールボックスから取得したデータはバイナリデータです。バイナリデータの分解手法に規定はありません。上記で呼び出している「DecodeMailData」関数の処理内容は、インストールを行ったサンプルシステム内のコードを参照してください(DispSmpl プロジェクト)。

4.6. タグトリガーによるタグ値変更通知の受信

特定の EgTag データの更新を受けて処理を行いたい場合は、タグトリガー(TagTrigger)を用いると容易に実現できます。タグトリガーとは ECI ファイルに記載されたデータ構造に従って、EgTag データの変更時に更新情報(構造体データ)をメールボックスに送信する機能です。

4.6.1. タグトリガーによるタグ値変更通知

タグ値変更の通知は ECI ファイルにタグトリガーの定義を行うことで、RT-edge フレームワーク内でタグ値の変更時のメッセージ通知を行います。以下にタグトリガーを定義した ECI ファイルのサンプルを記載します。

例)

MyServ では RT-edge 内のタグ "Tag001"、"Tag002"、"Tag003"、に対してタグトリガーの設定を行っており、これらのタグ値が書き換えられたタイミングでメッセージを受信します。

```
...
<RTedge...>
  <Tags>
    <Tag Name="Tag001" Type="6" />
    <Tag Name="Tag002" Type="10" />
    <Tag Name="Tag003" Type="11" />
  ...
</Tags>
...
</Datasets>
<Services>
  ...
  <Service Name="MyServ" Path="MyServ.exe"/>
    <TagTriggers>
      <TagTrigger Name="Tag001"/>
      <TagTrigger Name="Tag002"/>
      <TagTrigger Name="Tag003"/>
    ...
    </TagTriggers>
  </Service>
</Services>
</RTedge>
```


4.6.2. タグトリガーによって通知されたメッセージの受信

タグトリガーによって通知されたメッセージを受信する際は、ユーザメッセージハンドラを用います。ユーザメッセージハンドラ内でメッセージ ID「EM_TAGTRIGGER_NOTIFICATION」に対する処理を追記してください。以下に例として C でのコード(サンプルシステム格納コード)を下記に記載します。

タグトリガー通知受信例(C/C++)

```
...
int UserMessageHandler(const char* senderName, int messNo, void* param)
{
    int32_t result = 0;
    EG_TAGTRIGGER_INFOS* pData;

    switch (messNo)
    {
    case EM_TAGTRIGGER_NOTIFICATION:
    {
        pData = (EG_TAGTRIGGER_INFOS*)param;
        for (int i = 0; i < pData->TblCount; i++)
        {
            type = egTagTriggers[pData->pTblTagTriggers[i].arrNo].Type;
            switch (type)
            {
            case EgVtBoolean:
            {
                bool val = (bool)pData->pTblTagTriggers[i].val;
                printf("受信 : TagName [%s], value [%d], Type [%d] Size [%d]\n",
                    egTagTriggers[pData->pTblTagTriggers[i].arrNo].Name, val, type,
                    egTagTriggers[pData->pTblTagTriggers[i].arrNo].Size);
            }
            break;
            ...省略...
            case EgVtFloat:
            {
                float val = 0;
                memcpy(&val, &pData->pTblTagTriggers[i].val, 4);
                printf("受信 : TagName [%s], value [%f], Type [%d] Size [%d]\n",
                    egTagTriggers[pData->pTblTagTriggers[i].arrNo].Name, val, type,
                    egTagTriggers[pData->pTblTagTriggers[i].arrNo].Size);
            }
            break;
            default:
                break;
            }
        }
    }
    default: //上記以外は抜ける
        break;
    }
    return result;
}

...
// EG_TAGTRIGGER_INFOS 構造体 TagTrigger スレッド間データ受け渡し用 1 メッセージ
typedef struct
{
    uint16_t TblCount; // テーブルの配列数
    EG_TAGTRIGGER_INFO pTblTagTriggers[1]; // タグトリガー構造体テーブル
} EG_TAGTRIGGER_INFOS, * LPEG_TAGTRIGGER_INFOS;
```

```
typedef struct
{
    uint16_t      arrNo;          // ローカルメンバ変数(egTagTriggers)に対応する 配列 No
    union {                      // 変更値 (ただしSegment型、String型は対象外)
        union EgTrigValues
        {
            bool      boolVal;
            float      fltVal;
            double     dblVal;
            char       cVal;
            int16_t     iVal;
            int32_t     lVal;
            int32_t     intVal;
            int64_t     llVal;
            uint8_t     bVal;
            uint16_t    uiVal;
            uint32_t    ulVal;
            uint32_t    uintVal;
            uint64_t    ullVal;
        };
        intptr_t val;
    };
} EGTagTrigInfo,* LPEGTagTrigInfo;
```

4.7. メッセージオブジェクトを利用した送信

ECI ファイルにメッセージ(メッセージオブジェクト)を定義することで、定義したサービスコンテナの初期化処理完了後にメッセージを送信することができる機能です。メッセージは初期化完了後直ぐに送信することも可能ですが、タイマー機能を使用して初期化完了後数秒後にメッセージ送信するといったことも可能です。

また、ECI ファイルで定義したメッセージリストはプログラム内でアクセスすることができ、任意のタイミングで送信することも可能です。

4.7.1. 初期化完了後にメッセージ送信

ECI ファイルにメッセージの定義を行うことで、RT-edge フレームワーク内でコンテナの初期化終了後、メッセージ送信を行います。以下にメッセージを定義した ECI ファイルのサンプルを記載します。

例)

MyServ では RT-edge 内の Service 名 "TEST_EXE" に対して処理開始メッセージの定義、処理停止メッセージの定義を行っており、初期化完了後直ぐに TEST_EXE は処理開始メッセージを受け取り処理を開始します。初期化完了後 60 秒後、MyServ から処理停止メッセージを送信し、TEST_EXE は処理を停止しします。

ExecTiming : 1 … 初期化完了後 Timer で指定した値(秒)後にメッセージを送信

ExecTiming : 0 … メッセージを内部で保持する。任意のタイミングで送信(次章参照)

```
...
<RTedge...>
...
<Services>
...
  <Service Name="TEST_EXE" Path="TEST_EXE.exe"/>
  <Service Name="MyServ" Path="MyServ.exe">
    <Messages>
      <Message Name="EM_SERVICE_RUN" No="102" DestName="TEST_EXE" ExecTiming="1" Timer="0"/>
      <Message Name="EM_SERVICE_PAUSE" No="103" DestName="TEST_EXE" ExecTiming="1" Timer="60"/>
      <Message Name="MY_MESSAGE_0001" No="30001" DestName="TEST_EXE" ExecTiming="0"/>
    ...
  </Messages>
</Service>
</Services>
</RTedge>
```



メッセージ送信は自動で行われますが、送信先のコンテナ(メッセージ受信側)において、受信処理を実装しておく必要があります。受信処理は、4.8. ユーザーメッセージハンドラを参照ください。

4.7.2. 任意のタイミングでメッセージを送信

ECI で定義したメッセージを任意のタイミングで送信するコードを下記に記載します。API は非同期送信 API (EgFW_SendMessageAsync) を使用しております。通常の送信 API (EgMailboxSend) でも可能です。

メッセージオブジェクトを利用したメッセージ送信例(C/C++)

```
...
int iMessIndexNo = 2;
int rtn = 0;
//送信先サービスコンテナ名、メッセージ番号のみ指定
rtn = EgFW_SendMessageAsync(egMessageObjs[iMessIndexNo].DestName,
                           egMessageObjs[iMessIndexNo].No,
                           NULL,
                           0);

if(rtn!=0)
    printf("送信エラー : [%d]\n",rtn);
else
    printf("送信成功\n");
...
```

4.8. ユーザメッセージハンドラ

独自のイベントを作成する場合はユーザメッセージ(20000～29999)を使用します。

ユーザメッセージハンドラ関数は以下の仕様で作成してください。

- 1) 戻り値：int 型 正常終了(0) 異常(0 以外)
- 2) 関数引数 1：送信元メールボックス名 (const char *senderName)
- 3) 関数引数 2：メッセージ番号 (int inMessageNo)
- 4) 関数引数 3：送信データ (void* pData)

作成したユーザメッセージハンドラ関数は EglInit() の Config.fpUserMessHdlFunc で登録できます。登録したユーザメッセージハンドラにはシステムメッセージも届きます。通常は無処理としますが、インターセプトもできます。

ユーザメッセージハンドラ例(C/C++)

```
int32_t UserMessageHandler(const char *senderName, int32_t MessageNo, void*
pData)
{
    int rtn = 0;
    switch (MessageNo)
    {
        case EM_SERVICE_RUN:                // サービスコンテナ開始指令
            result = myStartService( );
            break;
        case EM_SERVICE_STOP:               // サービスコンテナ終了指令
            result = myKillService( );
            break;
        case EM_SERVICE_PAUSE:              // サービスコンテナ一時停止指令
            result = myPauseService( );
            break;
        case EM_SERVICE_UPDATE:             // サービスコンテナアップデート指令
            result = myUpdateService( );
        case 20001:                          // 20001 のメッセージ処理を呼び出し実装
            break;
        case 20002:                          // 20002 のメッセージ処理を呼び出し実装
            break;
        default:
            return -1;
    }
    return rtn;                            // 正常終了時は 0   0 以外は異常終了
}
```



以下の基本サービスコンテナ制御メッセージを受信した場合のリアクション処理インプリメントを推奨します。

- 1) EM_SERVICE_STOP
- 2) EM_SERVICE_START
- 3) EM_SERVICE_PAUSE
- 4) EM_SERVICE_UPDATE

ユーザメッセージハンドラ登録例(C/C++)

```
int32_t result;
EDGECONFIG config = EDGE_CONFIG_DEFAULT;

// Edge メッセージユーザハンドラの指定
config.fpUserMessHdlFunc = UserMessageHandler;
result = EgInit(&config); // RT-edge フレームワークの初期化
if (result == EDGE_SUCCESS)
    printf("正常終了");
```

4.9. サービスコンテナ間メッセージ通信(送信・受信)

RT-edge フレームワークは各サービスにおいて実行ファイル名でメールボックスを自動生成します。目的のサービスコンテナ名を指定してメールボックス送信すればメッセージを送ることができます。

メッセージ送信例(C/C++)

```
#include "egAPI.h"

int32_t result;
RTCDHANDLE hMbox;
result = EgMailboxOpenEC( "TargetApp" ,RTCD_WRITE ,&hMbox);
:
//サービスコンテナ停止指令を発行
result = EgMailboxSend( &hMbox ," MyApp" , EM_SERVICE_STOP, NULL ,0);
...
```

メッセージ送信例(C#)

```
using RTedge;

edge_API EGAPI = new edge_API();
int message = (int)EGDEFINE.EGMSG.EM_SERVICE_STOP;
:
//サービスコンテナ停止指令を発行
var ret = EGAPI.EgMailboxSendSC("TargetApp", "MyApp", message , new byte[0]);
...
```

メッセージの受信は、システムメッセージであれば RT-edge フレームワーク内処理で、ユーザ独自のメッセージであれば、ユーザメッセージハンドラを使用して受信します。ユーザメッセージハンドラに関しては「4.8. ユーザメッセージハンドラ」を参照ください。

4.10. サービスコンテナ作成時の注意事項

4.10.1. エラーの現示

サービスコンテナは機器との通信障害などデータ更新できないような異常を検知した場合、SERVICE.myService.Error インジケータに true を示すようにしてください。

4.10.2. エラーの回復

サービスコンテナは可能な限りエラー状態の回復処理に努めてください。エラー状態が回復した場合は SERVICE.myService.Error インジケータに false を示すようにしてください。

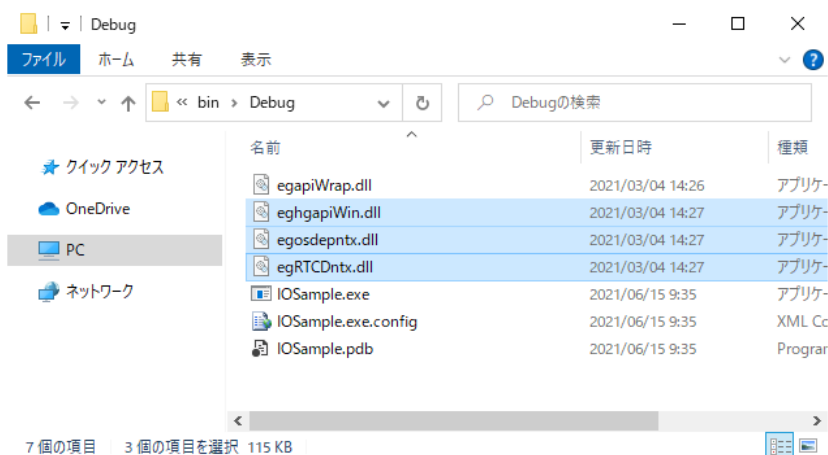
4.10.3. データ更新の扱い

サービスコンテナがエラー状態時にはデータタグを上書き(EgTagWrite)しないようにしてください。Egde システムとしてデータタグは最終正常値を保持するようにします。

4.10.4. デバッグ実行について

VisualStudio から C#アプリケーションのデバッグ実行する場合は、Debug フォルダに以下のファイルを配置してください。配置していない場合、EgInit() コールで失敗します。これらのファイルは RT-edge インストールフォルダ内に格納されています。

- 1) eghgapiWin.dll
- 2) egosdepntx.dll (Windows 環境の場合：egosdepwin.dll)
- 3) egRTCDntx.dll (Windows 環境の場合：egRTCDwin.dll)



< デバッグ実行時の DLL 配置 >

4.10.5. エラーの特定

SERVICE.myService.Error が true を示した場合に、その原因を特定することが出来る情報（エラーコード等）を格納するタグの追加を推奨します。

5. テンプレートを用いたサービスコンテナ作成

サービスコンテナを容易に作成できるように、インストーラにサービスコンテナ作成用テンプレートをご用意しています。サービスコンテナ作成用テンプレートには、以下の機能を実装しています。

- 1) サービスコンテナの初期化・開始・停止・削除関数
- 2) Edge メッセージハンドラ処理関数
- 3) サービスコンテナインジケータタグ更新関数
- 4) 定周期スレッド処理ひな型

サービスコンテナ作成用テンプレートを用いることでサービスコンテナを作成する上でのコード生成短縮が見込め、上記処理を容易に使用することができます。以下に INtime アプリケーション(C言語)用テンプレートを使用したサービスコンテナ、C#(.NET)用テンプレートを使用したサービスコンテナ作成方法について説明します。

5.1. サービスコンテナ作成用テンプレートのインストール

サービスコンテナ作成用テンプレートのインストールは、インストール手順書を参照してください。

5.2. テンプレートを用いた実装方法

5.2.1. INtime(C 言語)アプリケーションにおける方法

手順 1

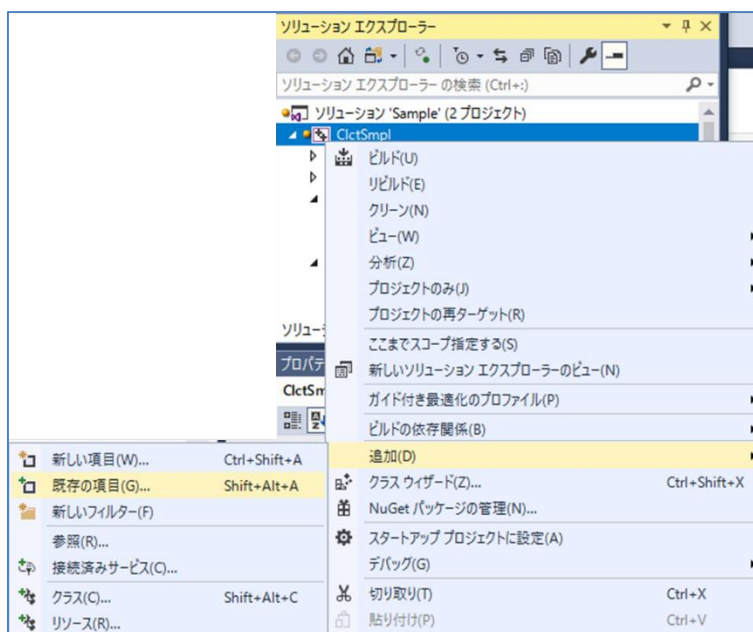
インストール後、以下ファイルをプロジェクトディレクトリへコピーします。

コピー元	コピー先
(インストールフォルダ)¥Template¥RT¥edgeCode.h	(プロジェクト)¥edgeCode.h
(インストールフォルダ)¥Template¥RT¥edgeCode.c	(プロジェクト)¥edgeCode.c

手順 2

Visual Studio において、上記ファイルをプロジェクトへ追加します。

ソリューションエクスプローラからプロジェクトを選択し、コンテキストメニューより「追加」-「既存の項目」を選択し、上記ファイルを追加します。



手順 3

テンプレートコード内に初期化関数を呼び出すファイルへ、以下ヘッダ読み込み処理を記載し、ヘッダファイル"edgeCode.h"内の定義「SERVICENAME」の値を作成するサービスコンテナ名(拡張子を除く)に変更してください。

(Eglnit 関数呼び出し後であれば、ECI ファイルに定義されたサービスコンテナ名を利用することができます。呼び出しは ServiceRealName 変数を使用します。)

呼び出し元ファイルへヘッダ追加(C/C++)

```

. . .

#include "edgeCode.h" // サービスコンテナ共通定義/関数ヘッダ追加

. . .

. . .

```

サービスコンテナ名の変更(C/C++)

```

. . .

////////////////////////////////////
/// 各種定義追加
///

/// <summary>
/// サービスコンテナ名
/// </summary>
#define SERVICENAME "Smp1RTA" // rta 拡張子を除くプログラムファイル名に変更

. . .

. . .

```

手順 4

以下は main 関数での呼び出し例です。下記コードを実装することで、サンプルは動作します。以上でテンプレートコードを使用した準備は完了です。必要に応じて main 関数内、テンプレートコード内のコードを修正して、目的に応じたサービスを作成してください。

メイン関数(C/C++)

```
int main(int argc, char* argv[])
{
    EVENTINFO ei; // Intime システムイベントの処理実装の為定義

    InitService(&MyState); // サービスコンテナの初期化 関数呼び出し
    SynchronizeRtLoader(); // サービスコンテナ準備完了の通知 関数呼び出し

    while (1) // イベント処理待機
    {
        // Intime システムイベントの処理分岐
        RtNotifyEvent(ALL_NOTIFICATIONS, WAIT_FOREVER, &ei);
        switch (ei.dwNotifyType)
        {
            case TERMINATE: // Intex からの削除指令
                KillService(&MyState); // サービスコンテナ削除 (プロセス停止)
                EgFinalize(); // Edge システム分離(必須 RT-edge フレームワーク後処理)
                exit(0); // サービスコンテナプロセス削除
                break;
            case KERNEL_SHUTDOWN_PENDING: // Intime カーネルの停止指令
                KillService(&MyState); // サービスコンテナ削除 (プロセス停止)
                EgFinalize(); // Edge システム分離(必須 RT-edge フレームワーク後処理)
                exit(0); // サービスコンテナプロセス削除
                break;
            case NT_BLUESCREEN: // Windows クラッシュ発生
                break;
        }
    }
    return 0;
}
```



テンプレートコード内コメントに[TODO]と記載がある箇所は必ず修正の検討を行ってください。

5.2.2. C#アプリケーションにおける方法

手順 1

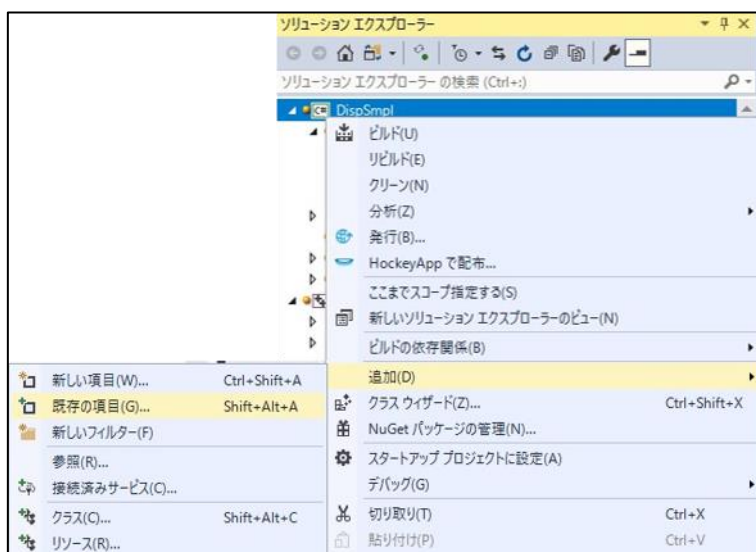
インストール後、以下ファイルをプロジェクトディレクトリへコピーします。

コピー元	コピー先
(インストールフォルダ)¥Template¥CSharp¥edgeCode.cs	(プロジェクト)¥ edgeCode.cs

手順 2

Visual Studio において、上記ファイルをプロジェクトへ追加します。

ソリューションエクスプローラからプロジェクトを選択し、コンテキストメニューより「追加」-「既存の項目」を選択し、上記ファイルを追加します。



手順 3

テンプレートコード内、RT-edge API を呼び出すファイルへ、以下 using ステートメントを追加します。また追加したファイル"edgeCode.cs"内の定義「SERVICENAME」の値を作成するサービス名(拡張子を除く)に変更してください。

(EgInit 関数呼び出し後であれば、ECI ファイルに定義されたサービスコンテナ名を利用することができます。呼び出しは ServiceRealName 変数を使用します。)

呼び出し元ファイルへステートメント追加(C#)

```

. . .
using RTedge;    // RT-edge system API
using EdgeCode; // RT-edge template code
. . .

```

サービスコンテナ名の変更(C#)

```

. . .
////
/// サービスコンテナ名
//
public const string SERVICENAME = "SamplWPF"; // exe 拡張子を除くプログラムファイル名に変更
. . .

```

手順 4

以下は Window クラスのコンストラクタ内で初期化関数を呼び出す例です。またサービス終了コールバック関数を実装し、テンプレートコード内の終了処理関数として登録することで、サンプルは動作します。以上でテンプレートコードを使用した準備は完了です。必要に応じて追加した関数内、テンプレートコード内のコードを修正して、目的に応じたサービスコンテナを作成してください。

初期化処理/終了コールバック関数(C#)

```

. . .
public partial class MainWindow : Window
{
    edgeCode EgProc = null;          //RT-edge フレームワーク+API を使用したコードクラス
    public MainWindow()
    {
        InitializeComponent();        //Component の初期化(自動生成コード)
        EgProc = new edgeCode();      //RT-edge 処理クラスの作成
        EgProc.mInstMainClose = Stop; //RT-edge フレームワーク内
                                     //終了メッセージ処理時の関数指定

        EgProc.InitService();         //RT-edge 処理クラスの初期化
    }
    /// <summary>
    /// サービスコンテナ終了処理
    /// </summary>
    public void Stop()
    {
        //アプリケーション終了処理
    }
}
. . .

```



テンプレートコード内コメントに[TODO]と記載がある箇所は必ず修正の検討を行ってください。

5.3. ECI ファイルの生成とタグの定義

サービスコンテナの開発には ECI ファイルの生成とサービスコンテナ内でアクセスするタグの定義が必要になります。ECI ファイルの生成・設定は RT-edge に付属する RT-edge コンフィグレーションツールを利用します。RT-edge コンフィグレーションツールの詳細と操作方法につきましては「RT-edge コンフィグレーションツール 取扱説明書」をご参照ください。

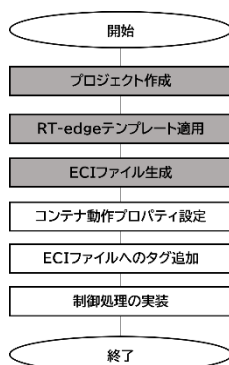
5.4. カスタムコンテナ開発例: デジタル I/O ボードの制御

本節では INtime 上で動作し、コントローラに接続されたデジタル I/O ボードの制御を行うカスタ

ムコンテナを例として、開発手順について説明します。

制御の内容は、デジタル I/O ボードの入力ポートから取得した値をタグ A に書き込み、同時にタグ B から読み込んだ値をデジタル I/O ボードの出力ポートに出力することとしています。

開発の流れを下図に示します。図のうち、グレースアウトしている作業は本書の各項目をご参照ください。



生成したカスタムコンテナの ECI ファイルに、以下のタグを追加します。タグの追加は RT-edge コンフィグレーションツールを使用してください。

追加対象 エレメント	タグ名	型	値	説明
Tags	TagIn	3	—	デジタル I/O ボードからの入力値を格納します。
	TagOut	3	—	デジタル I/O ボードへの出力値を格納します。
	SERVICE.IoSmpl.Cycle	7	10	コンテナのデータ更新周期設定
	SERVICE.IoSmpl.InPriority	3	172	コンテナのデータ入力プライオリティ
	SERVICE.IoSmpl.OutPriority	3	171	コンテナのデータ出力プライオリティ
	SERVICE.IoSmpl.Mode	3	1	コンテナのデータ更新動作モード:サイクリック
	SERVICE.IoSmpl.AutoRun	1	1	コンテナのデータ更新自動実行:開始

続いて、RT-edge テンプレートを追加したソースコードの main 関数内で以下の処理を実装します。

- 1) 制御対象のデジタル I/O ボードの検索
- 2) デジタル I/O ボードの入出力許可設定
- 3) デジタル I/O ボードからの入力値を TagIn にセットする
- 4) TagOut から取得した値をデジタル I/O ボードの出力にセットする

カスタムコンテナの開発例の説明は以上です。

6. サンプルシステム

サービスを複数使用したシステムのサンプルとして以下のシステムをご用意しています。
アプリケーションシステム、カスタムコンテナを作成する際のサンプルとしてご利用ください。

6.1. サンプルシステム実行ファイルのインストール

サンプルシステム実行ファイルのインストールはインストールマニュアルを参照してください。

6.2. サンプルシステム構成

サンプルシステムでは、外部からのデータ(デバイスのレジスタ入力やアナログポートからの入力など)を取り込むことを想定した「IoSmpl」コンテナ、取り込んだデータを画面へ表示させる「DispSmpl」コンテナで構成しています。

RT-edge フレームワークを利用して EgCollector でデータを定周期で収集し、画面でグラフ表示を行います。EgCollector は「DispSmpl」コンテナ、「DispSmpl」コンテナ内に実装することも可能ですが、サンプルではコンテナの役割を分ける目的で EgCollector によるデータ収集処理のみを行う「ClctSmpl」コンテナを実装しています。

以下に構成図と構成表を記載します。

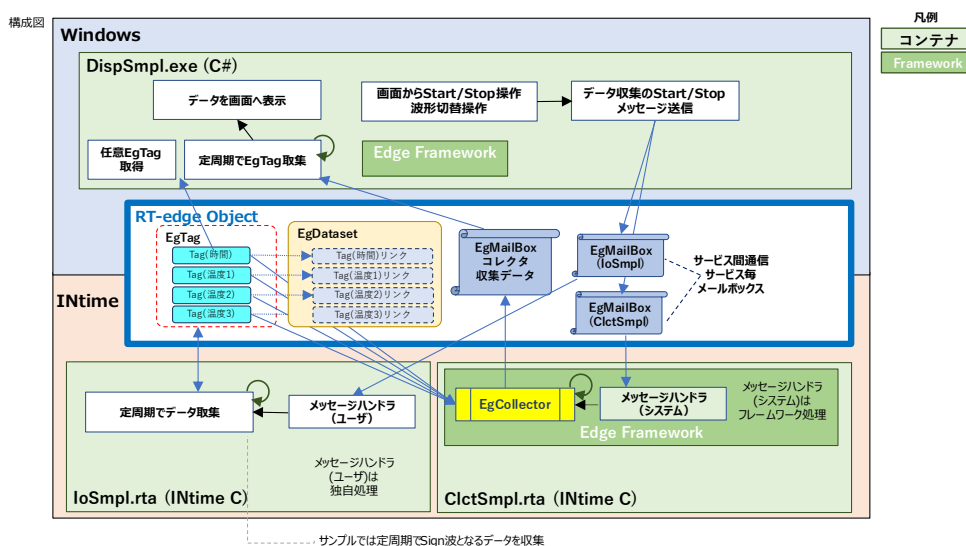


図 18 サンプルシステム構成図

表 15 サンプルシステム構成表

コンテナ名	DispSmpl.exe	IoSmpl.rta	ClctSmpl.rta
用途	画面表示 収集したデータを表示/収集命令操作	データ収集 外部からのデータ収取（アナログポートからの入力など）を想定（サンプルとして正弦波を出力）	Tag データ収集
OS/言語	Windows / C#言語	INtime / C 言語	INtime / C 言語
使用 RT-edge オブジェクト	EgTagRef EgCollectorRef EgDataset	EgTagRef	EgCollector EgTagRef EgMailBox

6.3. ソリューション構成

サンプルシステムのソリューション構成を以下に記載します。

サンプルシステムは Visual Studio2017 でビルドしたものです。

ディレクトリ/ファイル	説明
SampleSystem	サンプルシステムソリューションフォルダ
- Sample.sln	サンプルシステムソリューションファイル
- bin	成果物出力フォルダ
- Release	Release ビルド出力フォルダ
- ClctSmpl.rta	ClctSmpl コンテナモジュール(rta)
- ClctSmpl.xml	ClctSmpl ECI ファイル
- DispSmpl.exe	DispSmpl コンテナモジュール(exe)
- DispSmpl.xml	DispSmpl ECI ファイル
- EgBoot.xml	EgBoot (ブートストラッパー) ECI ファイル
- IoSmpl.rta	IoSmpl コンテナモジュール(rta)
- IoSmpl.xml	IoSmpl ECI ファイル
- ClctSmpl	ClctSmpl コンテナプロジェクトフォルダ
- ClctSmpl.c	ClctSmpl コンテナメインコード
- ClctSmpl.vcxproj	ClctSmpl コンテナプロジェクトファイル
- ClctSmpl.vcxproj.filters	ClctSmpl コンテナプロジェクトファイル
- edgeCode.c	ClctSmpl コンテナ RT-edge API を用いた実装コードファイル
- edgeCode.h	ClctSmpl コンテナ RT-edge API を用いた実装ヘッダファイル
- Readme.txt	コンテナ説明ファイル
- DispSmpl	DispSmpl コンテナ設定ファイル
- App.config	DispSmpl コンテナ xaml ファイル
- App.xaml	DispSmpl コンテナコードファイル
- App.xaml.cs	DispSmpl コンテナプロジェクトファイル
- DispSmpl.csproj	DispSmpl コンテナ RT-edge API を用いた実装ファイル
- edgeCode.cs	DispSmpl コンテナ画面コード
- MainWindow.xaml	DispSmpl コンテナ画面コード
- MainWindow.xaml.cs	DispSmpl コンテナ Struct 定義ファイル
- MyStruct.cs	コンテナ説明ファイル
- ReadMe.txt	プロパティフォルダ
- Properties	リソースコード/設定コード群
- ※	
- IoSmpl	IoSmpl コンテナプロジェクトフォルダ
- edgeCode.c	IoSmpl コンテナ RT-edge API を用いた実装コードファイル
- edgeCode.h	IoSmpl コンテナ RT-edge API を用いた実装ヘッダファイル
- IoSmpl.c	IoSmpl コンテナメインコード
- IoSmpl.vcxproj	IoSmpl コンテナプロジェクトファイル
- IoSmpl.vcxproj.filters	IoSmpl コンテナプロジェクトファイル
- ReadMe.txt	コンテナ説明ファイル

6.4. ECI ファイル構成

サンプルシステムの ECI ファイルについて記載します：

1) EgBoot.xml : RT-edge ブートストラッパー設定

RT-edge におけるサービスコンテナ、および関連サービス・アプリケーションの起動設定は、RT-edge ブートストラッパー設定により行います。設定は、RTedge エlement 内の Service Element 内に、各サービスコンテナ用の Element (Service Element) を追加します。

例：EgBoot.xml - IoSmpl サービスコンテナ設定

```
<Service Name="IoSmpl" Path="IoSmpl.rta" >
</Service>
```

Service Name="IoSmpl"

IoSmpl サービスコンテナ登録名 (24 文字まで指定可能)

Path="IoSmpl.rta"

IoSmpl サービスコンテナのファイル名(IoSmpl.rta)を指定します。

サンプルシステムでは以下のような設定になります。

```
<?xml version="1.0" encoding="utf-8"?>
<RTedge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  :
  <Services>
    <Service Name ="EgBoot" >
    </Service>
    <Service Name="EgLog" Path="EgLog.exe" Argument="DispNumMax=500" >
    </Service>
    <Service Name="IoSmpl" Path="IoSmpl.rta" >
    </Service>
    <Service Name="ClctSmpl" Path="ClctSmpl.rta" >
    </Service>
    <Service Name="DispSmpl" Path="DispSmpl.exe" >
    </Service>
  </Services>
</RTedge>
```

自身の設定

EgLog(ログ出力・表示機能)の起動設定

IoSmpl の起動設定

ClctSmpl の起動設定

DispSmpl の起動設定

2) IoSmpl.xml : IO サンプルサービスコンテナ設定

各サービスコンテナの ECI ファイルでは、サービスコンテナプロパティタグを定義することで、サービスコンテナの動作をカスタマイズすることができます。

例 : IoSmpl.xml - サービスコンテナ 動作自動スタートタグ(AutoRun)の指定

```
<Tag Name="SERVICE.IoSmpl.AutoRun" Type="1" Value="0"/>
```

Tag Name=" SERVICE.IoSmpl.AutoRun"	タグ登録名(IoSmpl サービスコンテナ用動作自動スタートタグ名)
Type="1"	Service Name= で指定した名前を指定する(左記 IoSmpl 部)
Value="0"	値の型。ここでは 0 or 1 を扱う Boolean 型のため"1"を指定
	0: 開始しない
	1: 開始する

サンプルシステムでは以下のような設定になります。AutoRun 以外のサービスコンテナプロパティタグの詳細は、RT-edge ユーザーズマニュアルを参照してください。

```
<?xml version="1.0" encoding="utf-8"?>
<RTedge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
:
<Tags>
  <Tag Name ="SERVICE.IoSmpl.Cycle" Type="7" Value="10"/>
  <Tag Name ="SERVICE.IoSmpl.InPriority" Type="3" Value="172"/>
  <Tag Name ="SERVICE.IoSmpl.OutPriority" Type="3" Value="171"/>
  <Tag Name ="SERVICE.IoSmpl.Mode" Type="3" Value="1"/>
  <Tag Name ="SERVICE.IoSmpl.AutoRun" Type="1" Value="0"/>
</Tags>
</RTedge>
```

3) ClctSmpl.xml : Tag データ収集サービスコンテナ設定

Collector 定義(データ収集定義)はサービスコンテナ用のエレメント(Service エレメント)内に記載します。Collector 定義で用いる Dataset オブジェクトも定義が必要です。

例 : ClctSmpl.xml - Collector 定義の指定

```
<Collector Name="ClctIO" Interval="100" Priority="140"
DatasetName="Dset1"/>
```

Collector Name="ClctIo"	Collector 名
Interval="100"	収集周期(ms)
Priority="140"	収集スレッド Priority
DatasetName="Dset1"	収集データセット名

例 : EgBoot.xml - Dataset 定義の指定

```
<Dataset Name="Dset1">
  <TagRefs>
    <TagRef Name="Temp001"/>
    <TagRef Name="Temp002"/>
    <TagRef Name="Temp003"/>
  </TagRefs>
</Dataset>
```

Dataset Name="Dset1"	Dataset 名
TagRefs	登録するタグへのリンク情報
TagRef Name="Temp001"	登録するタグへのリンク名



サンプルシステムでは、EgBoot.xml に Dataset の定義を行っていますが、Collector を定義している ECI(サンプルシステムでは ClctSmpl.xml)に記載することも可能です。

サンプルシステムでの ClctSmpl.xml は以下のような設定になります。

```
<?xml version="1.0" encoding="utf-8"?>
<RTedge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
:
  <Services>
    <Service Name ="ClctSmpl" >
      <TagRefs_OUT>
        <TagRef Name="Temp001"/>
        <TagRef Name="Temp001"/>
        <TagRef Name="Temp001"/>
      </TagRefs_OUT >
      <Collectors>
        <Collector Name="ClctIO" Interval="100" Priorit="140"
          DatasetName="Dset1" />
      </Collectors>
      <MailBoxes>
        <MailBox Name="CLCTMBOX" ColSize="2048" RowSize="1024"/>
      </MailBoxes>
    </Service>
  </Services>
</RTedge>
```

Collector と比較するために使用するメールボックスへ格納用 TagRef の定義

Collector の定義

Collector と比較するために使用するメールボックスの定義

4) DispSmpl.xml : 表示用ツールサービスコンテナ設定

サンプルシステムで用意している DispSmpl サービスコンテナは、ClctSmpl サービスコンテナで収集したデータを読み込み表示します。データはメールボックス(Collector 名)に格納されているため CollectorRef を利用してメールボックス名を取得します。CollectorRef 定義はサービスコンテナ用のエレメント(Service エレメント)内に記載します。

例 : DispSmpl.xml – CollectorRef 定義の指定

```
<CollectorRef Name="ClctIO"/>
```

CollectorRef Name="ClctIO"

Collector 名

サンプルシステムでの DispSmpl.xml は以下のような設定になります。

```
<?xml version="1.0" encoding="utf-8"?>
<RTedge xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
:
  <Services>
    <Service Name ="DispSmpl" >
      <TagRefs_OUT>
        <TagRef Name="Temp001"/>
        <TagRef Name="Temp001"/>
        <TagRef Name="Temp001"/>
      </TagRefs_OUT >
      <CollectorRefs>
        <CollectorRef Name="ClctIO"/>
      </CollectorRefs>
    </Service>
  </Services>
</RTedge>
```

個別参照用の TagRef 定義

CollectorRef の定義

6.5. 実行方法

RT-edge 実行環境に本ソリューション Release フォルダ以下のファイルを上書きコピーしてください。EgBoot.exe を起動することで、システムが起動します。

以下に上書きコピーの例とサンプルシステムの起動画面を記載します。

例) RT-edge インストールパス : C:\RTedge

1) コピー先フォルダ : 実行環境 PC - C:\RTedge\bin

2) コピー元ファイル : SampleSystem ソリューションフォルダ\bin\Release\※

(※ : rta/exe/xml ファイル群)

2) のファイルを 1)へコピーします。同名のファイルが存在する場合は上書きします。

EgBoot.exe を起動することで以下の画面が起動します。

The screenshot shows the RT-edge Sample System interface. The top part displays two graphs: 'EgBoot コレクタ で取得した値のグラフ' (Graph of values obtained by EgBoot collector) and '自作 収集処理 で取得した値のグラフ' (Graph of values obtained by custom collection processing). The bottom part contains control buttons for 'Start', 'Pause', and 'Stop' for various components, including 'Collector Start' and 'Collector Stop'. A 'TagRef_OUT' list is visible on the left.

各サービスコンテナの「Start」、コレクタの「Start」を押下後、画面描画処理の「開始」を押下することでデータを収集し、グラフとして表示します。

TagRef_OUT のリストを表示します

タグ選択コンボボックスから任意のタグを選択し、「取得」を押下することでタグの情報を取得し、型・アドレス・コメント情報を表示します。

更新履歴

版	日付	更新説明
1	2021.6	初版作成
2	2022.6	<ul style="list-style-type: none">作成例追加サービスコンテナ名についての説明変更
3	2023.5	タグトリガー機能についての説明を追加
4	2025.8	<ul style="list-style-type: none">RTCD の名称を「RT-edge Object」に変更ECI によるメッセージ送信例を記載

INDUSTRIAL REALTIME EDGE COMPUTERS

Container Creation Manual

発行元:株式会社マイクロネット

TEL: +81(0)299-90-1733

FAX:+81(0)299-92-8557

- 本書の内容、及び付属のソフトウェアの全部または一部を無断で転載することは禁止しております。
- 本製品の内容については、将来予告なしに変更することがあります。
- 本製品の内容について万一ご不審な点や記載もれなどお気づきの点がございましたら、お手数ですが、当社までご連絡ください。
- Windows XP、Windows 7、Windows 8、Windows 10 等、Windows は、米国 Microsoft Corporation における登録商標です。
- Visual Studio、Visual C++等は、米国、およびその他の国における Microsoft Corporation の登録商標です。
- INtime は米国 TenAsys における登録商標です。
- その他、記載されている会社名、製品名は、各社の商標又は登録商標です