INscope 活用法

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

REV.2 2020/10/27 RELEASE

目次

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。

1　概要

本ドキュメントは、TenAsys 社 INtime 製品に含まれるソフトウェアユーティリティである INscope Task Analyzer(INscope)の有効な利用法について紹介しています。

2　INscope

TenAsys 社製リアルタイム拡張製品 INtime ソフトウェアに含まれるユーティリティソフトウェアで、実行中スレッド(タスク)の処理時間測定、スレッディング状況の確認に利用できます。

測定に利用するタイムスタンプは、Pentium 以降すべての x86 プロセッサに存在するカウンタ TSC(Time Stamp Counter)もしくは、HPET(High Precision Event Timer)を利用しタイムスタンプをログとして保存します。

スレッド切り替えタイムスタンプ、API コールイン・コールアップ時タイムスタンプ、INscope 計測 API 実行時タイムスタンプを保存し、タスクの切り替えタイミングだけでなく、タスク内で実行される特定の処理における大まかな処理デルタ時間の計測が可能です。

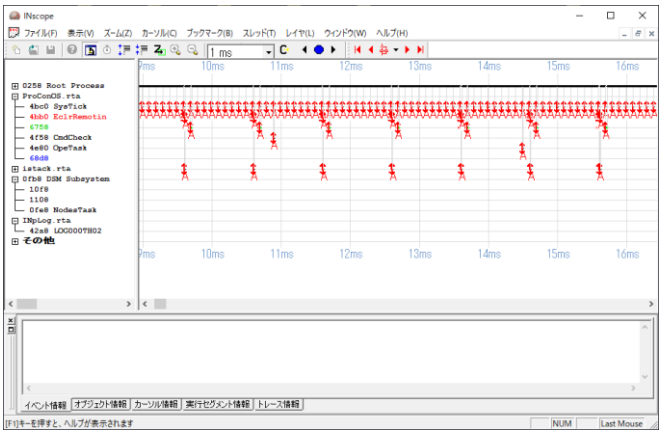


図 1. INscope Task Analyzer

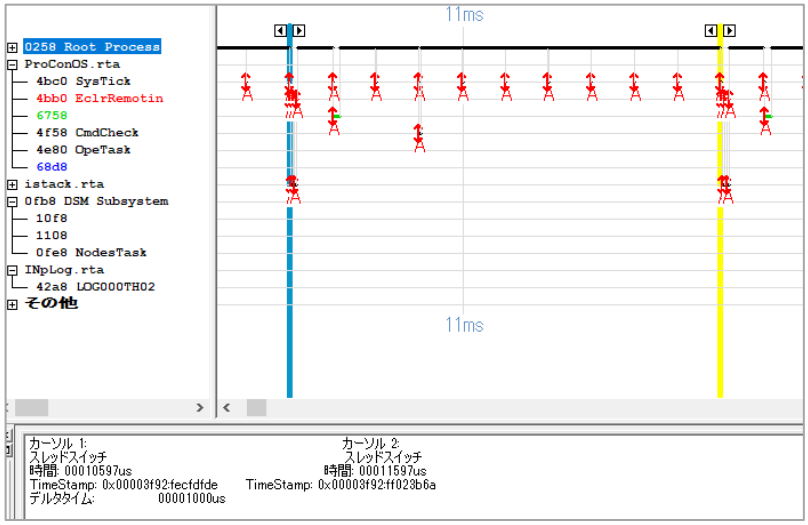


図 2.デルタ時間計測

計測測定用のカーソルを利用することで二点間の経過時間を測定することができます。

INscope API との組み合わせでユーザー処理の処理時間を簡易的に測定することも可能です。

3　利用方法

INscope Task Analyzer をシンプルに利用することで、測定中のログを採取できます。

採取したログは、INscope 上で編集を行い、RTS[INscope Files]として保存し、後ほど INscope から開くこともできます。ここでは INscope の利用法についていくつか紹介します。

シンプルトレース　アプリケーション実行中のアクティビティをトレースします。
本トレースにより、定常的なアプリケーションの処理シーケンスやスレッディング(スレッド切り替え)や処理時間の測定ができます。

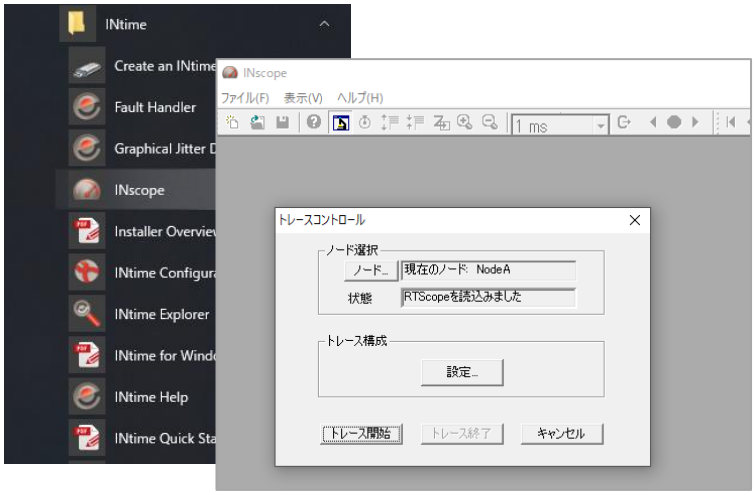
シナリオトレース　INscope API を使用し、特定の条件下でトレースの開始、もしくは開始したトレースの停止を行います。
アプリケーションの動作でスレッディング異常や処理異常が定常的に発生せず、所定の条件下で発生するようなケースでのトレース・測定に有効です。
発生する症状等によりシナリオが分かれますが、本ドキュメントでは二つのシナリオに絞って説明します。

3.1　シンプルトレース

INscope はユーティリティを起動後、起動したノード上のスレッディングアクティビティをすべて記録します。最もシンプルな利用法は、アプリケーション起動後、INscope を起動しトレースを行う方法です:

3.1.1　INscope の起動

スタート-[INtime]-[INscope]から INscope Task Analyzer を起動します:



[ノード]が選択されていれば、[トレース開始]ボタンが有効になります。

デフォルトの設定でよければ、[トレース開始ボタン]からトレースを開始できます。

図 3. INscope 起動

[トレースの開始]で開始した場合、[トレースの終了]もしくは、トレースバッファがフル状態でトレースが終了します。トレース終了後、取得した情報を画面上に表示します。

3.1.2 トレース設定

トレース設定から、トレースのポリシーや設定を変更できます。デフォルトの設定ポリシーを変更する場合、トレースコントロールダイアログから構成の設定を行います:

デフォルト設定

設定項目	デフォルト設定値	説明
ウォッチドッグタイマー	無効	ウォッチドッグタイマーの利用により、設定したウォッチドッグタイマー満了によりトレースを終了します。※ウォッチドッグタイマーのリセットによりトレースを維持します。
バッファフル	バッファ満杯時停止有効	所定のバッファサイズ分のトレース取得時にトレースが終了します。本チェックを外すと、リングバッファとしてバッファを使用します。
システムレイヤートレース	☐ INtime API(なし) ☐ C Library(なし) ☐ Network(なし) ☐ iWin32(なし)	トレース情報に組み込む API 層です。
割り込みトレース	選択なし	トレース情報に組み込む割り込みベクタ情報です

システムレイヤートレースから、トレース情報に組み込む API 層を追加します。

3.1.3 トレース開始

[トレースの開始]からトレースを開始します。トレースを開始すると、[トレースの開始]ボタンが[一時停止]に切り替わり、[トレース終了]ボタンが有効になります。

トレース設定-バッファフル-[バッファ満杯時停止]が有効に設定されている場合、トレース情報にてバッファがフルになった状態で、ボタンは、[リセット]、[トレース閲覧]に替わります:

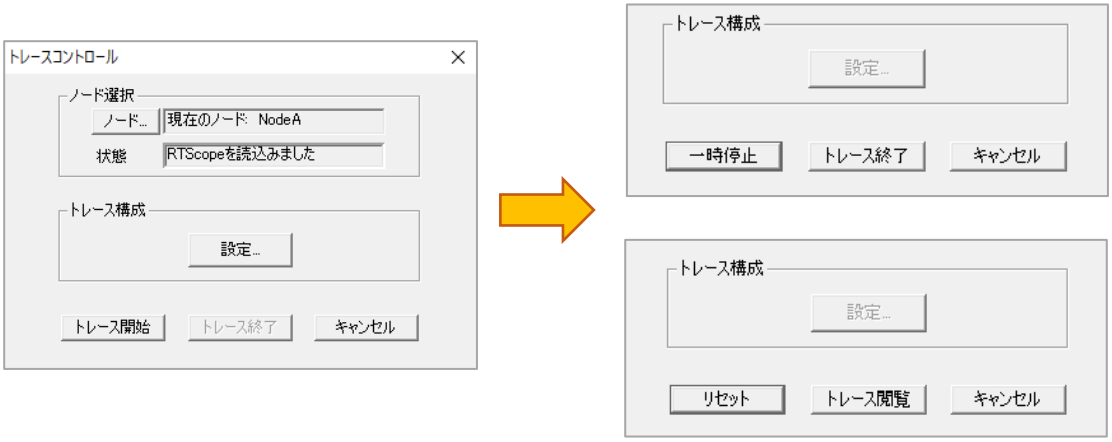


図 4.トレース開始

3.1.4 トレース終了

[トレース終了]、もしくは[トレース閲覧]によりトレースしたログ情報を画面上に表示します:

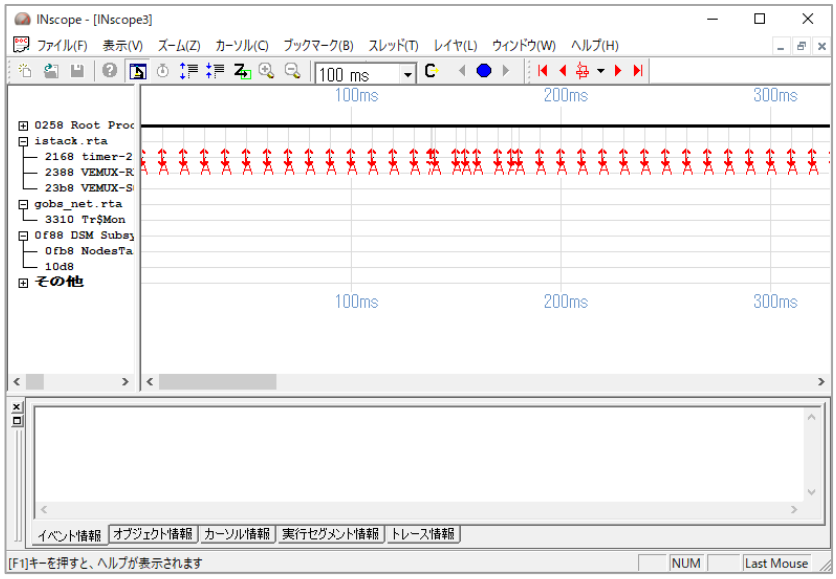
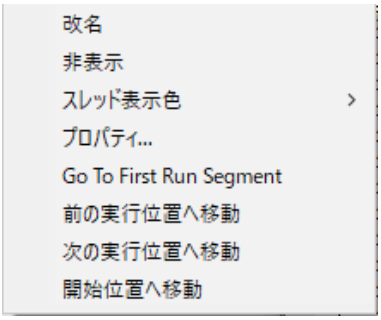


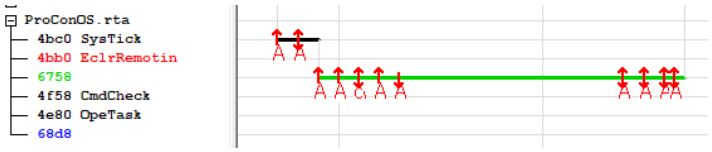
図 5.トレース情報表示

3.1.5 トレース情報の編集

トレース情報ポップアップメニュー

収集したトレースログは分析のため編集することができます。スレッド表示リスト(スレッドペイン)では右クリック-ポップアップメニューから以下の設定が可能です:

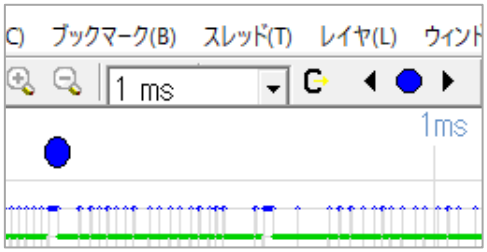


メニュー	設定効果
改名	スレッド表示名を[改名]します: 
非表示	分析に必要ないスレッドを非表示にします。
スレッド表示色	スレッドトレース情報を以下の選択肢から変更することができます: 黒色 青色 赤色 緑色 黄色 橙色 紫色 水色 
プロパティ	選択スレッドのプロパティ情報を最下部オブジェクト情報欄に表示します: 

ブックマーク

所定の箇所にブックマークを作成することで、容易にジャンプ表示することができます:

[ブックマーク]-[ブックマーク挿入モード]にてブックマークを追加します。



3.1.6 トレース情報の保存

[ファイル]-[名前を付けて保存]から、INscope トレース情報を RTS ファイルとして保存することができます。

3.2 シナリオトレース

スレッディング異常や処理異常が定常的に発生せず、また発生頻度が少ない場合、手動でタイミングを合わせてトレース情報を取得するのは困難です。シナリオトレースは、ある特別な条件時のトレース情報を取得する方法です。主にアプリケーションデバッグ時に有効で、INscope API を組み込むことで様々なトレースシナリオを構成することができます(参照:**INscope API 利用)**。

ここでは以下のケースを想定したシナリオトレース方法を説明します:

1. **定常周期処理異常時のトレース**

   一定周期で駆動するアプリケーションスレッドが駆動しないタイミングがある

2. **処理時間測定**

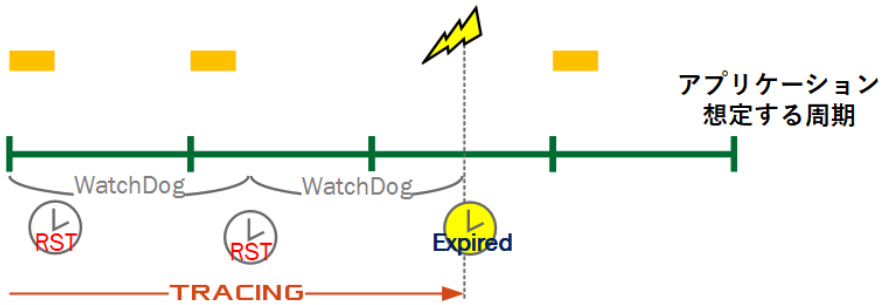
   ユーザスレッド内処理を測定するためのマークを追加する。

3. **異常発生判断時のトレース**

   定常処理において異常が発生した直前のトレースを取得する

3.2.1 定常周期処理異常時のトレース

定常周期処理異常が発生した時点のトレースログを残す場合、ウォッチドッグ設定が有効です。



例えば、通常 100ms 毎に駆動するアプリケーションスレッドのインターバルが 200ms となった瞬間の状態をトレースログに採取する場合、次のように設定します:

設定

設定項目		設定値
ウォッチドッグタイマー	ウォッチドッグタイマー有効	**有効**: タイムアウト値(ms) **180**ms
バッファフル	バッファ満杯時停止	**無効**
システムレイヤートレース	☐ INtime API	**あり**
	☐ C Library	**あり**
	☐ Network	**なし**
	☐ iWin32	**なし**
割り込みトレース	利用可能な割り込み	**なし**

アプリケーションコード

デバッグ用コード

```
void Thread(void)
{
    // 対象スレッドコード
    while(1)
    {
        RtSleep( 100 );
        //ToDo 処理が 100ms を大きく外れるときがある
        ToDo();
    }
}
```

INscope API 組み込みコード

```
#include <traceapi.h>
void Thread(void)
{
    unsigned short iCounter=0;
    start_RT_trace(1);  //INscope トレースを開始
    while(1)
    {
       RtSleep( 100 );
       RT_I_am_alive(); //ウォッチドッグタイマをリセットし、トレースログ出力を抑制
       //ToDo 処理が 100ms を大きく外れるときがある
       iCounter++;
       log_RT_event('O', iCounter); //正常実行回数等トレースログに出力
       ToDo();
    }
}
```

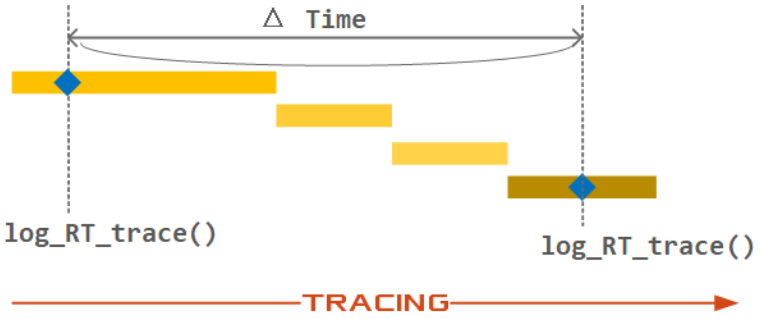
トレース開始

1. INscope アナライザを起動します。
2. [設定]から、上記設定を行います。
3. アプリケーションプロセスを実行します。
4. アプリケーションプロセスを開始し、当該スレッドコード、**start_RT_trace(1)**の実行時に、[トレース開始]から、[一時停止]に変わります。

5. ウォッチドッグタイマーの設定により、180ms 間 **RT_I_am_alive** がコールできない場合、ウォッチドッグタイマー満了となりトレースは停止し、[トレース終了]から[トレース閲覧]に変わります。

3.2.2 処理時間測定

INscope API を利用することで処理時間測定用のマーキングをトレースログ内に出力することができます:



設定

設定項目		設定値
ウォッチドッグタイマー	ウォッチドッグタイマー効	**無効**: タイムアウト値(ms)
バッファフル	バッファ満杯時停止	**無効**
システムレイヤートレース	☐ INtime API	**あり**
	☐ C Library	**あり**
	☐ Network	**なし**
	☐ iWin32	**なし**
割り込みトレース	利用可能な割り込み	**なし**

アプリケーションコード

デバッグ用コード

```
void Thread(void)
{
    // 対象スレッドコード
    while(1)
    {
        RtSleep( 100 );
        //ToDo 処理が 100ms を大きく外れるときがある
        ToDo();
    }
}
```

INscope API 組み込みコード

```
#include <traceapi.h>
void Thread(void)
{
    unsigned short iCounter=0;
    start_RT_trace(1);  //INscope トレースを開始
    while(1)
    {
       RtSleep( 100 );
       iCounter++;
       log_RT_event('S', iCounter); //処理計測開始点
       ToDo();
       log_RT_event('E', iCounter); //処理計測終了点
       if ( iCounter >= 100 ) //100 件採取し、ログを停止・終了
       {
           stop_RT_trace();  //INscope トレース終了
       }
    }
}
```

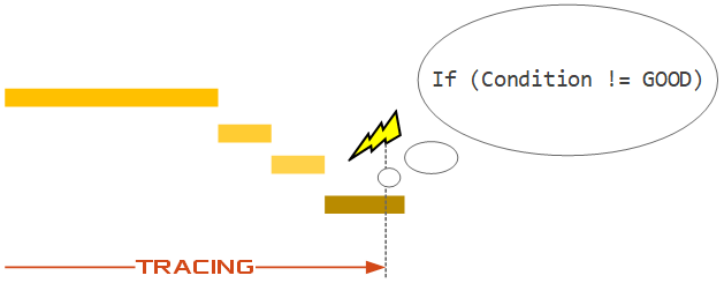
トレース開始

1. INscope アナライザを起動します。
2. [設定]から、上記設定を行います。
3. アプリケーションプロセスを実行します。

4. アプリケーションプロセスを開始し、当該スレッドコード、**start_RT_trace(1)**の実行時に、[トレース開始]から、[一時停止]に変わります。
5. カウンター(iCounter)が 100 以上となった時点で、**stop_RT_trace()**がコールされ、トレースは停止し、[トレース終了]から[トレース閲覧]に変わります。

3.2.3 異常発生判断時のトレース

アプリケーションにより処理時間が通常より長い、処理のステータスでエラーとなる等、「異常が発生した」と判断された前後のトレースログを取得するために INscope API の利用が有効です。



アプリケーションはエラーとなる条件成立時にトレースを停止するように API を組み込みます:

設定

設定項目		設定値
ウォッチドッグタイマー	ウォッチドッグタイマー効	**無効**: タイムアウト値(ms)
バッファフル	バッファ満杯時停止	**無効**
システムレイヤートレース	☐ INtime API	**あり**
	☐ C Library	**あり**
	☐ Network	**なし**
	☐ iWin32	**なし**
割り込みトレース	利用可能な割り込み	**なし**

アプリケーションコード

デバッグ用コード

```
void Thread(void)
{
    // 対象スレッドコード
    while(1)
    {
        if ((WAIT_FAILED == WaitForRtSemaphore(h , WAIT_FOREVER )){
            //プロセス終了時しか発生しないケース
        }
        ToDo();
    }
}
```

INscope API 組み込みコード

```
#include <traceapi.h>
void Thread(void)
{
    unsigned short iCounter=0;
    start_RT_trace(1);  //INscope トレースを開始
    while(1)
    {
        if ((WAIT_FAILED == WaitForRtSemaphore(h , WAIT_FOREVER )){
           //プロセス終了時しか発生しないケース
           stop_RT_trace();  //INscope トレース終了
        }
        iCounter++;
        log_RT_event('-', iCounter);
        ToDo();
    }
}
```

トレース開始

1. INscope アナライザを起動します。
2. [設定]から、上記設定を行います。
3. アプリケーションプロセスを実行します。
4. アプリケーションプロセスを開始し、当該スレッドコード、**start_RT_trace(1)**の実行時に、[トレース開始]から、[一時停止]に変わります。
5. 異常発生時で、**stop_RT_trace()**がコールされ、トレースは停止し、[トレース終了]から[トレース閲覧]に変わります。

4 設定

4.1 メモリ設定

INscope によりトレース可能なトレース件数はメモリサイズにより変わります(4096:DEFAULT - 256000)。本設定は、[表示]-[オプション]から設定します:

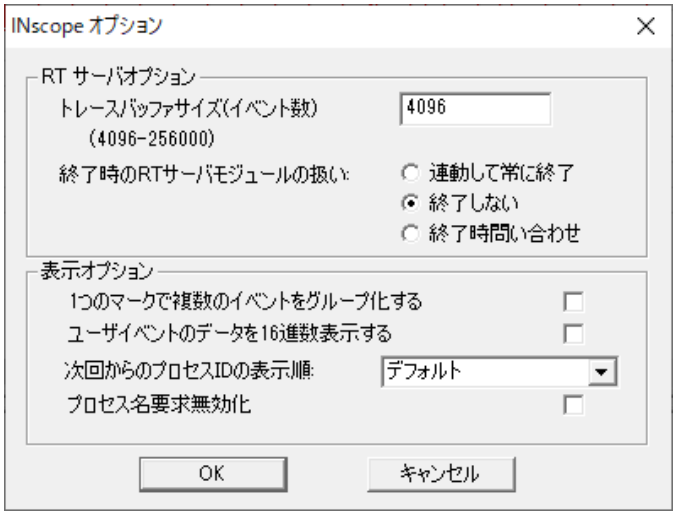


図 6.INscope オプション

トレースレコード数と INscope で消費されるメモリ量はおおむね以下のような関係となります:

レコード数	消費メモリサイズ
4096(DEFAULT)	164K
128000	2104K
256000	4104K

4.2 INscope API 利用

ユーザプロジェクトワークスペース上の設定と、コード上 API 記述により INscope API を組み込むことができます:

INtime Help には、INscope API に関するシンタックスが掲載されています。こちらを合わせて参照ください:
INtime Help -> INtime SDK -> About INtime -> Other system libraries -> INscope API

主な INscope API

INscope API	説明
start_RT_trace	現在設定されているトレース設定にてトレースを開始します。
stop_RT_trace	トレースを停止後、トレースメモリをロックし、INscope に対しトレース完了を通知します。
log_RT_event	INscope トレース情報に任意のタイミングでログイベントを出力できます。
RT_I_am_alive	ウォッチドッグタイマーをリセットします(ウォッチドッグタイマー利用時有効)。

4.2.1 プロジェクトワークスペース設定

ワークスペースから、コンパイラ・リンカー設定プロジェクト(vcxproj)ファイルの[プロパティ]を開きます。[リンカー設定]-[入力]-[追加の依存ファイル]に、INscope API ライブラリファイルを追加します:

追加ライブラリ	**traceapi.lib**

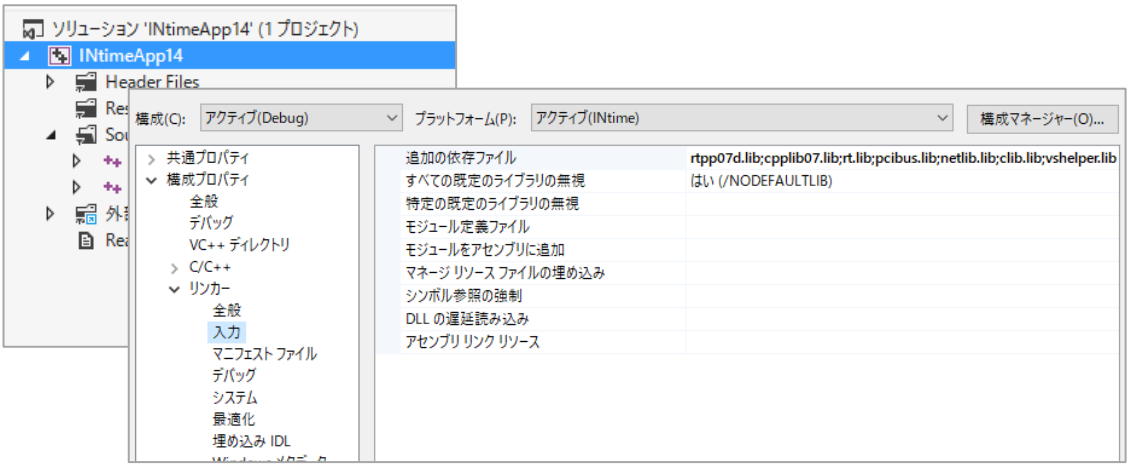


図 7.追加の依存ファイル

4.2.2 コードモジュールの編集

INscope API をコールするモジュールでは、`traceapi.h` をインクルードするように修正します:

```
#include <traceapi.h>

void Thread(void)
{
```

5　改訂履歴

版数	発行日	改定内容
第 1 版	2020 年 10 月	初版発行
第 2 版	2020 年 10 月	画像解像度の更新。 シナリオケースに図解追加