User's Manual

CS+ V3.00.00

統合開発環境

ユーザーズマニュアル Pythonコンソール編

対象デバイス

78K0マイクロコントローラ

RL78ファミリ

78K0Rマイクロコントローラ

V850ファミリ

RXファミリ

RH850ファミリ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2014.08

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：　輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等

   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

(2012. 4)

このマニュアルの使い方

このマニュアルは，RH850 ファミリ，RX ファミリ，V850 ファミリ，RL78 ファミリ，78K0R マイクロコントローラ，および 78K0 マイクロコントローラ用アプリケーション・システムを開発する際の統合開発環境である CS+ について説明します。

CS+ は，RH850 ファミリ，RX ファミリ，V850 ファミリ，RL78 ファミリ，78K0R マイクロコントローラ，および 78K0 マイクロコントローラの統合開発環境（ソフトウエア開発における，設計，実装，デバッグなどの各開発フェーズに必要なツールをプラットフォームである IDE に統合）です。統合することで，さまざまなツールを使い分ける必要がなく，本製品のみを使用して開発のすべてを行うことができます。

対象者	このマニュアルは，CS+ を使用してアプリケーション・システムを開発するユーザを対象としています。
目的	このマニュアルは，CS+ の持つソフトウエア機能をユーザに理解していただき，これらのデバイスを使用するシステムのハードウエア，ソフトウエア開発の参照要資料として役立つことを目的としています。
構成	このマニュアルは，大きく分けて次の内容で構成しています。 1. 概　　説 2. 機　　能 A. ウインドウ・リファレンス B.Python コンソール /Python 関数
読み方	このマニュアルを読むにあたっては，電気，論理回路，マイクロコンピュータに関する一般知識が必要となります。 凡例　データ表記の重み　：左が上位桁，右が下位桁 アクティブ・ロウの表記　：XXX （端子，信号名称に上線） 注　：本文中についた注の説明 注意　：気をつけて読んでいただきたい内容 備考　：本文中の補足説明 数の表記　：10 進数 ... XXXX 16 進数 ... 0xXXXX

目次

1. 概　　説

CS+ は，マイクロコントローラ用の統合開発環境です。Python コンソールは，スクリプト言語である IronPython（.NET Framework 上で動作する Python）を使用して，CS+ を制御することができます。CS+ を制御するための関数，プロパティ，クラス，イベントを追加しています。

このドキュメントでは，Python コンソールの使用方法，ならびに CS+ 向けに機能拡張した関数，プロパティ，クラス，イベントについて説明します。

1.1 概　　要

CS+ が提供する CS+ 制御用の関数，プロパティ，クラス，イベントを使用することにより，プロジェクトの作成，ビルド，デバッグなど，CS+ を制御することができます。

1.2 特　　長

Python コンソールの特長を次に示します。

- IronPython 機能
  IronPython の機能を使用することができます。
  Python コンソールで使用可能な IronPython 言語では，Python 言語が持つ機能に加えて，.NET Framework が持つ様々なクラス・ライブラリを使用することができます。
  IronPython の言語仕様については，以下の URL を参照してください。
  http://ironpython.net/
- プロジェクト機能
  プロジェクトの作成，読み込み，アクティブ・プロジェクトの変更などを行うことができます。
- ビルド機能
  プロジェクト全体でのビルド，ファイル単位でのビルドなどを行うことができます。
- デバッグ機能
  デバッグ・ツールへの接続，切断，プログラムの実行制御，メモリや変数の参照や設定など行うことができます。

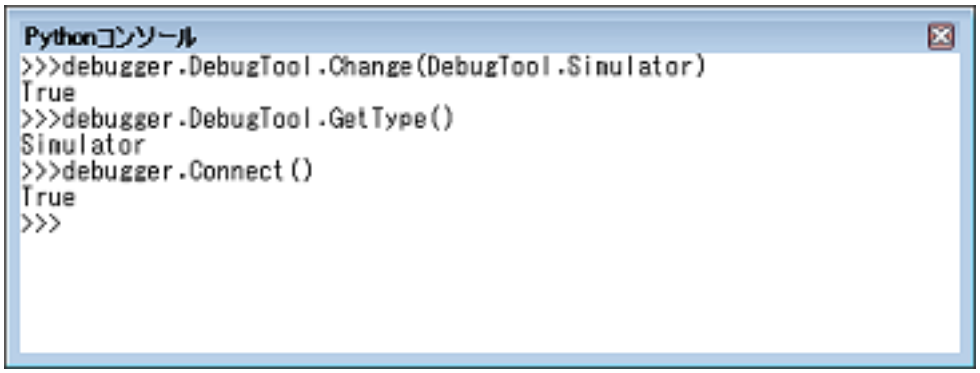
2. 機　　能

この章では，Python コンソールの使用方法について説明します。

2.1 Python 関数を実行する

CS+ では，IronPython 関数や制御文，および CS+ を制御するために追加された CS+ Python 関数（「B.3 CS+ Python 関数／クラス／プロパティ／イベント」参照）をコマンド入力方式で実行することができます。
［表示］メニュー→［Python コンソール］を選択すると，Python コンソール パネルがオープンします。
パネル上で Python 関数や制御文を実行することにより，CS+，およびデバッグ・ツールを操作することができます。

図 2.1　　Python コンソール パネル



備考　　Python コンソール，および Python 関数の詳細については，「B. Python コンソール /Python 関数」を参照してください。

A. ウインドウ・リファレンス

ここでは，Python コンソールに関連したパネルについて説明します。

A.1 説　　明

以下に，Python コンソールに関するパネルの一覧を示します。

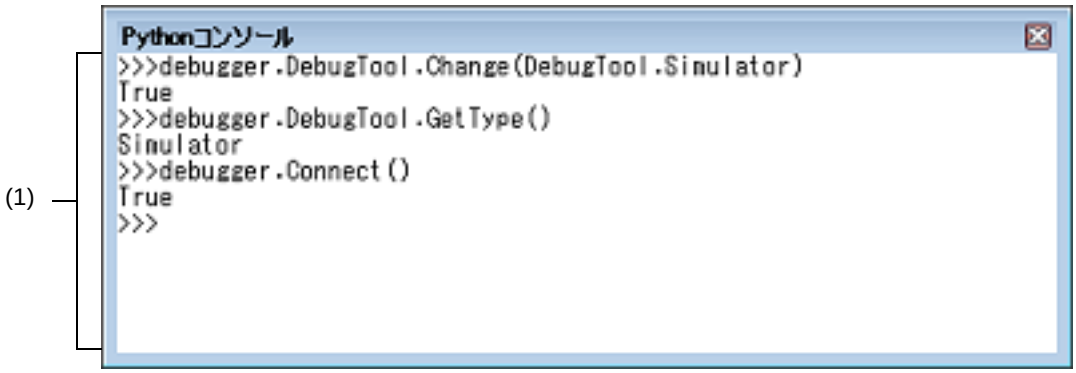
表 A.1　　パネル一覧

パネル名	機能概要
Python コンソール パネル	IronPython を利用して，コマンド入力方式で CS+，およびデバッグ・ツールを操作します。

Python コンソール パネル

IronPython を利用して，コマンド入力方式で CS+，およびデバッグ・ツールを操作します。

図 A.1 Python コンソール パネル



ここでは，以下の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［［ファイル］メニュー（Python コンソール パネル専用部分）］
- ［コンテキスト・メニュー］

［オープン方法］

- ［表示］メニュー→［Python コンソール］の選択

［各エリアの説明］

(1) 入出力エリア
IronPython 関数や制御文，および CS+ Python 関数を入力，実行します。
また，関数の実行結果やエラーの表示も行います。
IronPython 関数の結果を表示する場合は print 文を使用してください。

［［ファイル］メニュー（Python コンソール パネル専用部分）］

Python コンソール パネル専用の［ファイル］メニューは以下のとおりです（その他の項目は共通です）。

Python コンソール を保存	現在パネル上に表示されている内容を，前回保存したテキスト・ファイル（*.txt）に保存します。 なお，起動後にはじめてこの項目を選択した場合は，［名前を付けて Python コンソール を保存 ...］の選択と同等の動作となります。
名前を付けて Python コンソール を保存 ...	現在パネル上に表示されている内容を，指定したテキスト・ファイル（*.txt）に保存するために，名前を付けて保存 ダイアログをオープンします。

［コンテキスト・メニュー］

切り取り	選択している文字列を切り取ってクリップ・ボードに移動します。
コピー	選択している文字列をクリップ・ボードにコピーします。
貼り付け	クリップ・ボードの内容をキャレット位置に挿入します。
すべて選択	このパネルに表示しているすべての文字列を選択状態にします。

強制停止	実行中のコマンドを強制停止します。
クリア	出力結果をすべてクリアします。
Python を初期化	Python を初期化します。
スクリプト・ファイルの選択 ...	スクリプト・ファイルの選択 ダイアログをオープンし，選択した Python のスクリプト・ファイルを実行します。

B. Python コンソール /Python 関数

ここでは，CS+ が提供する Python コンソール，および Python 関数について説明します。

B.1 概　　要

Python コンソール・プラグインは，IronPython 言語を用いたコンソール・ツールです。
IronPython 言語でサポートされている関数や制御文に加えて，CS+ を制御するために追加された CS+ Python 関数も使用することができます。
以下に，CS+ が提供する機能を示します。

- Python コンソール パネル上で，IronPython 関数や制御文，および CS+ Python 関数（「B.3 CS+ Python 関数／クラス／プロパティ／イベント」参照）を実行することができます（「2.1 Python 関数を実行する」参照）。
- CS+ をコマンドラインから起動する際に，スクリプト・ファイルを指定して実行することができます（「CS+ 統合開発環境 ユーザーズマニュアル プロジェクト操作編」参照）。
- プロジェクト・ファイルの読み込み時に，あらかじめ用意しておいたスクリプトを実行することができます（「B.2 関連ファイル」参照）。

B.2 関連ファイル

以下に，CS+ Python 関数の関連ファイルを示します。

- *プロジェクト・ファイル名* .py
  プロジェクト・ファイルと同じフォルダにプロジェクト・ファイルと同じ名前で拡張子が “py” のファイルが存在する場合，プロジェクト・ファイルの読み込み時に自動的に読み込んで実行します。
  アクティブ・プロジェクトが対象となります。

- *ダウンロード・ファイル名* .py
  ダウンロード・ファイルと同じフォルダにダウンロード・ファイルと同じ名前で拡張子が “py” のファイルが存在する場合，ダウンロードした後に自動的に読み込んで実行します。

B.3 CS+ Python 関数／クラス／プロパティ／イベント

ここでは，CS+ Python 関数，クラス，プロパティ，イベントについて説明します。

CS+ Python 関数には，以下の規約があります。

- 引数にデフォルト値が存在する場合，【指定形式】の引数は“ *引数名 = デフォルト値* ”と表記されています。
  引数は，値のみを指定することも可能です。

  例　【指定形式】が function(*arg1*, *arg2* = 1, *arg3* = True) の場合，*arg1* はデフォルト値なし，*arg2* のデフォルト値は 1，*arg3* のデフォルト値は True となります。
  引数は，function("main", 1, True) のように指定することが可能です。

- デフォルト値が存在する引数は，省略することができます。
  ただし，引数が判別可能な場合に限ります。

  例　【指定形式】が function(*arg1*, *arg2* = 1, *arg3* = True) の場合

```
>>>function("main")                  …function("main", 1, True) とみなします
>>>function("main", 2)               …function("main", 2, True) とみなします
>>>function("main", arg3 = False)    …function("main", 1, False) とみなします
>>>function("main", False)           …arg2 と arg3 が判別できないため NG
```

- 引数は，“ *引数名 = 値* ”のように指定することにより，指定順を変更することができます。

  例　【指定形式】が function(*arg1*, *arg2* = 1, *arg3* = True) の場合

```
>>>function(arg3 = False, arg1 = "main", arg2 = 3)      …OK
>>>function(False, "main", 3)   …arg1 = False, arg2 = "main", arg3 = 3
とみなすため NG
```

- 引数にフォルダやファイルへのパスを記載する場合は注意が必要です。
  IronPython では，¥（バックスラッシュ）を制御文字として認識します。例えば，先頭が t で始まるフォルダ名やファイル名の場合，¥t で TAB 文字と認識してしまいます。これを回避するには次のように記載します。

  例 1.　パス指定の前に r を記載すると，IronPython は "" の中がパスと認識します。

```
r"C:¥test¥test.py"
```

  例 2.　¥（バックスラッシュ）ではなく /（スラッシュ）を使用します。

```
"C:/test/test.py"
```

本ドキュメントでは，/（スラッシュ）を使用して記載しています。

B.3.1 CS+ Python 関数（基本操作用）

以下に，CS+ Python 関数（基本操作用）の一覧を示します。

表 B.1 CS+ Python 関数（基本操作用）

関数名	機能概要
ClearConsole	Python コンソールに表示している文字列をクリアします。
CubeSuiteExit	CS+ を終了します。
Help	CS+ Python 関数のヘルプを表示します。
Hook	フック関数を登録します。
Save	編集中のすべてのファイル，およびプロジェクトを保存します。
Source	スクリプト・ファイルを実行します。

ClearConsole

Python コンソールに表示している文字列をクリアします。

[指定形式]

```
ClearConsole()
```

[引数]

なし

[戻り値]

文字列のクリアに成功した場合　: True
文字列のクリアに失敗した場合　: False

[詳細説明]

- Python コンソールに表示している文字列をクリアします。

[使用例]

```
>>>ClearConsole()
True
>>>
```

CubeSuiteExit

CS+ を終了します。

[指定形式]

```
CubeSuiteExit()
```

[引数]

なし

[戻り値]

なし

[詳細説明]

- CS+ を終了します。

注意 プロジェクト・ファイルに変更があった場合でも，編集中のファイルは保存しません。
保存する必要がある場合は，Save 関数を使用してください。

[使用例]

```
>>>CubeSuiteExit()
```

Help

CS+ Python 関数のヘルプを表示します。

[指定形式]

```
Help()
```

[引数]

なし

[戻り値]

なし

[詳細説明]

- CS+ 統合ヘルプを起動して，CS+ Python 関数のヘルプを表示します。

[使用例]

```
>>>Help()
```

Hook

フック関数，またはコールバック関数を登録します。

[指定形式]

```
Hook(scriptFile)
```

[引数]

引数	説明
scriptFile	フック関数，またはコールバック関数が定義されたスクリプト・ファイルを指定します。

[戻り値]

なし

[詳細説明]

- *scriptFile* を読み込んで，スクリプト・ファイルに宣言されているフック関数，またはコールバック関数を登録します。
  フック関数，コールバック関数以外の関数が宣言されていても，問題はありません。
  なお，フック関数，コールバック関数は，スクリプト・ファイルの終了時に登録します。
- フック関数が宣言されている場合は，CS+ のイベント発生後に呼び出されます。
- フック関数の種類を以下に示します。
  なお，フック関数には引数はありません。

フック関数	イベント
BeforeBuild	ビルド開始前
BeforeDownload	ダウンロード前
AfterDownload	ダウンロード後
AfterCpuReset	CPU リセット後
BeforeCpuRun	実行開始前
AfterCpuStop	ブレーク後
AfterActionEvent	アクション・イベント後（Printf イベントのみ）

例　スクリプト・ファイルの記述例

```
def BeforeDownload():
  # ダウンロード前に行いたい処理を追記
```

- フック関数は，以下の操作で初期化されます。
  - プロジェクト・ファイルの読み込み
  - プロジェクト・ファイルの新規作成
  - アクティブ・プロジェクトの変更
  - デバッグ・ツールの切り替え

- Python 初期化

- コールバック関数が宣言されている場合は，CS+ のイベント発生後に呼び出されます。
- コールバック関数名は，“pythonConsoleCallback” です。
  なお，コールバック関数の引数は，コールバック要因を示します。

引数の値	コールバック要因
10	イベント登録後
11	イベント削除後
12	実行の開始前
13	ブレーク後
14	CPU リセット後
18	デバッグ・ツールのプロパティ変更後
19	ダウンロード後
20	メモリ，またはレジスタ変更後
21	アクション・イベント後（Printf イベントのみ）
30	ビルド前
63	XRunBreak で指定した時間経過後

例 スクリプト・ファイルの記述例

```
def pythonConsoleCallback(Id):
  if Id == 63:
    # XRunBreak で指定した時間経過後に行いたい処理を追記
```

注意 1. コールバック関数内では以下の関数を実行しないでください。
debugger.Reset 関数
debugger.Run 関数
debugger.Breakpoint 関数

注意 2. コールバック関数内で，別の条件の debugger.XRunBreak.Set することはできません。
以下のような指定は行わないでください。

```
def pythonConsoleCallback(Id):
  if Id = 63:
    debugger.XRunBreak.Delete()
    debugger.XRunBreak.Set(1, TimeType.Ms, True)
```

［使用例］

```
>>>Hook("E:/TestFile/TestScript/testScriptFile2.py")
```

Save

編集中のすべてのファイル，およびプロジェクトを保存します。

［指定形式］

```
Save()
```

［引数］

なし

［戻り値］

編集中のすべてのファイル，およびプロジェクトの保存に成功した場合　：True
編集中のすべてのファイル，およびプロジェクトの保存に失敗した場合　：False

［詳細説明］

- 編集中のすべてのファイル，およびプロジェクトを保存します。

［使用例］

```
>>>Save()
True
>>>
```

Source

スクリプト・ファイルを実行します。

[指定形式]

```
Source(scriptFile)
```

[引数]

引数	説明
scriptFile	実行するスクリプト・ファイルを指定します。

[戻り値]

なし

[詳細説明]

- *scriptFile* で指定したスクリプト・ファイルを実行します。
- 本関数は，IronPython の execfile と同様の動作を行います。

[使用例]

```
>>>Source("../../testScriptFile2.py")
>>>Source("E:/TestFile/TestScript/testScriptFile.py")
>>>
```

B.3.2 CS+ Python 関数（共通）

以下に，CS+ Python 関数（共通）の一覧を示します。

表 B.2 CS+ Python 関数（共通）

関数名	機能概要
common.GetOutputPanel	出力 パネルの内容を表示します。
common.OutputPanel	出力 パネルに文字列を表示します。
common.PythonInitialize	Python を初期化します。

common.GetOutputPanel

出力 パネルの内容を表示します。

[指定形式]

```
common.GetOutputPanel()
```

[引数]

なし

[戻り値]

出力 パネルに表示している文字列

[詳細説明]

- 出力 パネルに表示している文字列を表示します。

[使用例]

```
>>> common.OutputPanel("------ Start ----- ")
True
>>> com = common.GetOutputPanel()
------ Start -----
>>> print com
------ Start -----
```

common.OutputPanel

出力 パネルに文字列を表示します。

[指定形式]

```
common.OutputPanel(output, messageType = MessageType.Information)
```

[引数]

引数	説明	
output	出力 パネルに表示する文字列を指定します。	
messageType	出力 パネルに表示する文字列に対して，色分け表示を行うメッセージの種類を指定します。 色はオプション ダイアログの［全般 - フォントと色］カテゴリの設定に従います。 指定可能なメッセージの種類を以下に示します。	
	種類	説明
	MessageType.Error	エラー
	MessageType.Information	標準（デフォルト）
	MessageType.Warning	警告

[戻り値]

出力 パネルへの表示に成功した場合 ： True
出力 パネルへの表示に失敗した場合 ： False

[詳細説明]

- *output* に指定した文字列を出力 パネルに表示します。

[使用例]

```
>>>common.OutputPanel(" エラーが発生しました。", MessageType.Error)
True
>>>
```

common.PythonInitialize

Python を初期化します。

[指定形式]

```
common.PythonInitialize(scriptFile = "")
```

[引数]

引数	説明
scriptFile	Python を初期化したあとに実行するスクリプト・ファイルを指定します（デフォルト：指定なし）。 絶対パスで指定してください。

[戻り値]

なし

[詳細説明]

- Python を初期化します。
  定義した関数やインポートしたモジュールをすべて破棄して初期化します。スクリプト実行中にこの関数を実行した場合は，実行状態に関係なく強制的に Python を初期化します。
- *scriptFile* でスクリプト・ファイルを指定した場合，初期化後に指定したスクリプト・ファイルを実行します。
- *scriptFile* を指定しない場合は，Python の初期化のみを行います。

注意 強制的に Python を初期化するため，実行状態によってはエラーが表示されることがあります。

[使用例]

```
>>>common.PythonInitialize()
>>>
>>>common.PythonInitialize("C:/Test/script.py")
```

B.3.3 CS+ Python 関数（プロジェクト用）

以下に，CS+ Python 関数（プロジェクト用）の一覧を示します。

表 B.3 CS+ Python 関数（プロジェクト用）

関数名	機能概要
project.Change	アクティブ・プロジェクトを変更します。
project.Close	プロジェクトを閉じます。
project.Create	プロジェクトを新規作成します。
project.File.Add	アクティブ・プロジェクトにファイルを追加します。
project.File.Exists	アクティブ・プロジェクトにファイルが存在するかどうかを確認します。
project.File.Information	アクティブ・プロジェクトに登録されているファイルの一覧を表示します。
project.File.Remove	アクティブ・プロジェクトからファイルを外します。
project.GetDeviceNameList	マイクロコントローラのデバイス名の一覧を表示します。
project.GetFunctionList	アクティブ・プロジェクトの関数の一覧を表示します。
project.GetVariableList	アクティブ・プロジェクトの変数の一覧を表示します。
project.Information	プロジェクト・ファイルの一覧を表示します。
project.Open	プロジェクトを開きます。

project.Change

アクティブ・プロジェクトを変更します。

[指定形式]

```
project.Change(projectName)
```

[引数]

引数	説明
projectName	変更するプロジェクト・ファイル名，またはサブプロジェクト名をフルパスで指定します。

[戻り値]

アクティブ・プロジェクトを変更するのに成功した場合　：True
アクティブ・プロジェクトを変更するのに失敗した場合　：False

[詳細説明]

- *projectName* に指定したプロジェクトをアクティブ・プロジェクトに変更します。
- *projectName* に指定するプロジェクト・ファイルは，現在開いているプロジェクトに含まれている必要があります。

[使用例]

```
>>>project.Change("C:/project/sample/sub1/subproject.mtpj")
True
>>>
```

project.Close

プロジェクトを閉じます。

[指定形式]

```
project.Close(save = False)
```

[引数]

引数	説明
save	編集中のすべてのファイル，およびプロジェクトを保存するかどうかを指定します。 True ：編集中のすべてのファイル，およびプロジェクトを保存します。 False ：編集中のすべてのファイル，およびプロジェクトを保存しません（デフォルト）。

[戻り値]

プロジェクトを閉じるのに成功した場合 ： True
プロジェクトを閉じるのに失敗した場合 ： False

[詳細説明]

- 現在開いているプロジェクトを閉じます。
- *save* に “True” を指定した場合は，編集中のすべてのファイル，およびプロジェクトを保存します。

[使用例]

```
>>>project.Close()
True
>>>
```

project.Create

プロジェクトを新規作成します。

[指定形式]

```
project.Create(fileName, micomType, deviceName, projectKind = ProjectKind.Auto,
compiler = Compiler.Auto, subProject = False, registerNaming =
RegisterNaming.Structured)
```

[引数]

<table>
<tr><th>引数</th><th colspan="2">説明</th></tr>
<tr><td>fileName</td><td colspan="2">作成するプロジェクト・ファイルのフルパスを指定します。
拡張子を指定していない場合は，自動的に補完します。
作成するプロジェクトがメイン・プロジェクトの場合（subProject に “False” を指定した場合）は “.mtpj”，サブプロジェクトの場合（subProject に “True” を指定した場合）は “.mtsp” が補完されます。
また，該当する拡張子以外を指定した場合は拡張子が追加されます。</td></tr>
<tr><td rowspan="7">micomType</td><td colspan="2">作成するプロジェクトのマイクロコントローラの種類を指定します。
指定可能な種類を以下に示します。</td></tr>
<tr><td>種類</td><td>説明</td></tr>
<tr><td>MicomType.RH850</td><td>RH850 用プロジェクト</td></tr>
<tr><td>MicomType.RX</td><td>RX 用プロジェクト</td></tr>
<tr><td>MicomType.V850</td><td>V850 用プロジェクト</td></tr>
<tr><td>MicomType.RL78</td><td>RL78 用プロジェクト</td></tr>
<tr><td>MicomType.K0R</td><td>78K0R 用プロジェクト</td></tr>
<tr><td>MicomType.K0</td><td>78K0 用プロジェクト</td></tr>
<tr><td>deviceName</td><td colspan="2">作成するプロジェクトのマイクロコントローラの品種名を文字列で指定します。</td></tr>
</table>

<table>
<tr><th>引数</th><th colspan="2">説明</th></tr>
<tr><td>projectKind</td><td colspan="2">作成するプロジェクトの種類を指定します。
指定可能な種類を以下に示します。
なお，マイクロコントローラが RH850 で，“ProjectKind.Auto” を指定した場合，または projectKind を指定していない場合は，以下が自動的に指定されます。
シングルコアの場合 ： ProjectKind.Application
マルチコアでメイン・プロジェクトの場合 ： ProjectKind.MulticoreBootLoader
マルチコアでサブプロジェクトの場合 ： ProjectKind.MulticoreApplication</td></tr>
<tr><td></td><td>種類</td><td>説明</td></tr>
<tr><td></td><td>ProjectKind.Application</td><td>アプリケーション用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.Library</td><td>ライブラリ用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.DebugOnly</td><td>デバッグ専用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.Empty</td><td>空のアプリケーション用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.CppApplication</td><td>C++ アプリケーション用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.RI600V4</td><td>RI600V4 用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.RI600PX</td><td>RI600PX 用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.RI850V4</td><td>RI850V4 用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.RI850MP</td><td>RI850MP 用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.RI78V4</td><td>RI78V4 用プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.MulticoreBootLoader</td><td>マルチコア用ブート・ローダ・プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.MulticoreApplication</td><td>マルチコア用アプリケーション・プロジェクト</td></tr>
<tr><td></td><td>ProjectKind.Auto</td><td>指定した micomType，deviceName，subProject から判断して，プロジェクトの種類を選択します（デフォルト）。</td></tr>
</table>

<table>
<tr><th>引数</th><th colspan="2">説明</th></tr>
<tr><td>compiler</td><td colspan="2">使用するコンパイラを指定します。
指定しない場合は，マイクロコントローラの種類によって自動で選択されます。</td></tr>
<tr><td></td><td>種類</td><td>説明</td></tr>
<tr><td></td><td>Compiler.Auto</td><td>指定した micomType から判断して，コンパイラを選択します（デフォルト）。</td></tr>
<tr><td></td><td>Compiler.CC_RH</td><td>CC-RH
micomType に “MicomType.RH850” を指定した場合に本引数を無指定にすると，CC-RH が自動的に選択されます。</td></tr>
<tr><td></td><td>Compiler.CC_RX</td><td>CC-RX
micomType に “MicomType.RX” を指定した場合に本引数を無指定にすると，CC-RX が自動的に選択されます。</td></tr>
<tr><td></td><td>Compiler.CA850</td><td>CA850
micomType に “MicomType.V850” を指定し，deviceName に “V850E”，または “V850ES” を指定した場合に本引数を無指定にすると，CA850 が自動的に選択されます。</td></tr>
<tr><td></td><td>Compiler.CX</td><td>CX
micomType に “MicomType.V850” を指定し，deviceName に “V850E2” を指定した場合に本引数を無指定にすると，CX が自動的に選択されます。</td></tr>
<tr><td></td><td>Compiler.CC_RL</td><td>CC-RL
CS+ for CC で “MicomType.RL78” を指定した場合に本引数を無指定にすると，CC-RL が自動的に選択されます。</td></tr>
<tr><td></td><td>Compiler.CA78K0R</td><td>CA78K0R
micomType に “MicomType.K0R”，または CS+ for CACX で “MicomType.RL78” を指定した場合に本引数を無指定にすると，CA78K0R が自動的に選択されます。</td></tr>
<tr><td></td><td>Compiler.CA78K0</td><td>CA78K0
micomType に “MicomType.K0” を指定した場合に本引数を無指定にすると，CA78K0 が自動的に選択されます。</td></tr>
<tr><td>subProject</td><td colspan="2">メイン・プロジェクト，サブプロジェクトのどちらを作成するかを指定します。
False ： メイン・プロジェクトを作成します（デフォルト）。
True ： サブプロジェクトを作成します。</td></tr>
<tr><td>registerNaming</td><td colspan="2">作成するプロジェクトのマイクロコントローラが RH850 の場合に，IOR 表示方式を指定します。
RH850 以外の場合に本引数を指定した場合は無視されます。
指定可能な種類を以下に示します。</td></tr>
<tr><td></td><td>種類</td><td>説明</td></tr>
<tr><td></td><td>RegisterNaming.Combined</td><td>IOR 表示方式にレジスタ名称連結方式を選択します。</td></tr>
<tr><td></td><td>RegisterNaming.Structured</td><td>IOR 表示方式にレジスタ名称構造化方式を選択します（デフォルト）。</td></tr>
</table>

［戻り値］

プロジェクトを新規作成するのに成功した場合　：True
プロジェクトを新規作成するのに失敗した場合　：False

［詳細説明］

- *fileName* で指定したプロジェクト・ファイルを新規作成します。
  作成するプロジェクトのマイクロコントローラは *micomType* と *deviceName* で指定します。
  作成するプロジェクトの種類は *projectKind* で指定します。
- *subProject* に “True” を指定した場合は，サブプロジェクトを作成します。

［使用例］

```
>>>project.Create("c:/project/test.mtpj", MicomType.RX, "R5F52105AxFN",
ProjectKind.Application)
True
>>>
```

project.File.Add

アクティブ・プロジェクトにファイルを追加します。

[指定形式]

```
project.File.Add(fileName, category = "")
```

[引数]

引数	説明
fileName	アクティブ・プロジェクトに追加するファイルをフルパスで指定します。 複数ファイルを指定する場合は，“[″*file1*″, ″*file2*″]” の形式で指定します。
category	ファイルの追加先カテゴリを指定します（デフォルト：指定なし）。 複数階層を指定する場合は，“[″*one*″, ″*two*″]” の形式で指定します。

[戻り値]

アクティブ・プロジェクトへのファイルの追加に成功した場合 : True
アクティブ・プロジェクトへのファイルの追加に失敗した場合 : False
fileName に複数ファイルを指定した際に 1 つでもファイルの追加に失敗した場合 : False

[詳細説明]

- *fileName* に指定したファイルをアクティブ・プロジェクトに追加します。
- *category* を指定した場合，指定したカテゴリの下にファイルを追加します。
  指定したカテゴリが存在しない場合は，新規に作成します。

[使用例]

```
>>>project.File.Add("C:/project/sample/src/test.c", "test")
True
>>>project.File.Add(["C:/project/sample/src/test1.c", "C:/project/sample/src/
test2.c"], ["test", "src"])
True
```

project.File.Exists

アクティブ・プロジェクトにファイルが存在するかどうかを確認します。

[指定形式]

```
project.File.Exists(fileName)
```

[引数]

引数	説明
fileName	アクティブ・プロジェクトに存在するかどうかを確認するファイルをフルパスで指定します。

[戻り値]

指定したファイルがアクティブ・プロジェクトに存在する場合　：True
指定したファイルがアクティブ・プロジェクトに存在しない場合　：False

[詳細説明]

- *fileName* に指定したファイルがアクティブ・プロジェクトに存在するかどうかを確認します。

[使用例]

```
>>>project.File.Exists("C:/project/sample/src/test.c")
True
>>>
```

project.File.Information

アクティブ・プロジェクトに登録されているファイルの一覧を表示します。

[指定形式]

```
project.File.Information()
```

[引数]

なし

[戻り値]

アクティブ・プロジェクトに登録されているファイル名の一覧（フルパス付き）

[詳細説明]

- アクティブ・プロジェクトに登録されているファイルの一覧をフルパスで表示します。

[使用例]

```
>>>project.File.Information()
C:¥prj¥src¥file1.c
C:¥prj¥src¥file2.c
C:¥prj¥src¥file3.c
>>>
```

project.File.Remove

アクティブ・プロジェクトからファイルを外します。

[指定形式]

```
project.File.Remove(fileName)
```

[引数]

引数	説明
fileName	アクティブ・プロジェクトから外すファイルをフルパスで指定します。 複数ファイルを指定する場合は，“[″*file1*″, ″*file2*″]” の形式で指定します。

[戻り値]

アクティブ・プロジェクトからファイルを外すのに成功した場合　：True
アクティブ・プロジェクトからファイルを外すのに失敗した場合　：False

[詳細説明]

- *fileName* に指定したファイルをアクティブ・プロジェクトから外します。
- ファイルの削除は行いません。

[使用例]

```
>>>project.File.Remove("C:/project/sample/src/test.c")
True
>>>project.File.Remove(["C:/project/sample/src/test1.c", "C:/project/sample/src/
test2.c"])
True
```

project.GetDeviceNameList

マイクロコントローラのデバイス名の一覧を表示します。

[指定形式]

```
project.GetDeviceNameList(micomType, nickName = "")
```

[引数]

引数	説明	
micomType	作成するプロジェクトのマイクロコントローラの種類を指定します。 指定可能な種類を以下に示します。	
	種類	説明
	MicomType.RH850	RH850 用プロジェクト
	MicomType.RX	RX 用プロジェクト
	MicomType.V850	V850 用プロジェクト
	MicomType.RL78	RL78 用プロジェクト
	MicomType.K0R	78K0R 用プロジェクト
	MicomType.K0	78K0 用プロジェクト
nickName	マイクロコントローラの愛称を文字列で指定します（デフォルト：指定なし）。 プロジェクトを新規に作成する際に使用するプロジェクト作成 ダイアログの［使用するマイクロコントローラ］の第一階層に表示されている文字列を指定してください。	

[戻り値]

デバイス名の一覧（文字列）

[詳細説明]

- *micomType* に指定したマイクロコントローラのデバイス名の一覧を表示します。
- *nickName* を指定した場合，指定した愛称に含まれるデバイス名のみを表示します。

[使用例]

```
>>>project.GetDeviceNameList(MicomType.RL78)
R5F10BAF
R5F10AGF
R5F10BAG
R5F10BGG
........
>>>devlist = project.GetDeviceNameList(MicomType.RL78, "RL78/F13 (ROM:128KB)")
R5F10BAG
R5F10BGG
........
>>>
```

project.GetFunctionList

アクティブ・プロジェクトの関数の一覧を表示します。

[指定形式]

```
project.GetFunctionList(fileName = "")
```

[引数]

引数	説明
fileName	関数の一覧を表示するファイルをフルパスで指定します（デフォルト：指定なし）。

[戻り値]

関数情報の一覧（詳細は FunctionInfo クラスを参照してください）

[詳細説明]

- アクティブ・プロジェクトの関数の一覧を，以下の形式で表示します。

```
関数名 戻り値の型 開始アドレス 終了アドレス ファイル名
```

- *fileName* を指定した場合，指定したファイルに含まれる関数のみを表示します。
- *fileName* を指定しない場合は，すべての関数を表示します。

注意 この関数は，プログラム解析の関数一覧に表示している情報を使用しています。

[使用例]

```
>>>project.GetFunctionList()
func1 int 0x00200 0x00224 C:¥project¥src¥test1.c
func2 int 0x00225 0x002ff C:¥project¥src¥test2.c
>>>project.GetFunctionList("C:/project/src/test1.c")
func1 int 0x00200 0x00224 C:¥project¥src¥test1.c
>>>
```

project.GetVariableList

アクティブ・プロジェクトの変数の一覧を表示します。

[指定形式]

```
project.GetVariableList(fileName = "")
```

[引数]

引数	説明
fileName	変数の一覧を表示するファイルをフルパスで指定します（デフォルト：指定なし）。

[戻り値]

変数情報の一覧（詳細は VariableInfo クラスを参照してください）

[詳細説明]

- アクティブ・プロジェクトの変数の一覧を，以下の形式で表示します。

```
変数名 属性 型 アドレス サイズ ファイル名
```

- *fileName* を指定した場合，指定したファイルに含まれる変数のみを表示します。
- *fileName* を指定しない場合は，すべての変数を表示します。

注意 この関数は，プログラム解析の変数一覧に表示している情報を使用しています。

[使用例]

```
>>>project.GetVariableList()
var1 volatile int 0x000014e4 4 C:¥project¥src¥test1.c
var2 static int 0x000014e8 4 C:¥project¥src¥test2.c
>>>project.GetVariableList("C:/project/src/test1.c")
var1 volatile int 0x000014e4 4 C:¥project¥src¥test1.c
>>>
```

project.Information

プロジェクト・ファイルの一覧を表示します。

[指定形式]

```
project.Information()
```

[引数]

なし

[戻り値]

プロジェクト・ファイル名の一覧

[詳細説明]

- 開いているプロジェクトに含まれているメイン・プロジェクト，およびサブプロジェクトのプロジェクト・ファイルの一覧を表示します。

[使用例]

```
>>>project.Information()
C:¥project¥sample¥test.mtpj
C:¥project¥sample¥sub1¥sub1project.mtsp
C:¥project¥sample¥sub2¥sub2project.mtsp
>>>
```

project.Open

プロジェクトを開きます。

[指定形式]

```
project.Open(fileName, save = False)
```

[引数]

引数	説明
fileName	プロジェクト・ファイルを指定します。
save	他のプロジェクトを開いていた場合，開いていたプロジェクトを閉じる際に，編集中のすべてのファイル，およびプロジェクトを保存するかどうかを指定します。 True ：編集中のすべてのファイル，およびプロジェクトを保存します。 False ：編集中のすべてのファイル，およびプロジェクトを保存しません（デフォルト）。

[戻り値]

プロジェクトを開くのに成功した場合 ： True
プロジェクトを開くのに失敗した場合 ： False

[詳細説明]

- *fileName* で指定したプロジェクトを開きます。
- 他のプロジェクトを開いていた場合は，開いていたプロジェクトを閉じます。
  save に “True” を指定した場合は，開いていたプロジェクトの編集中のすべてのファイル，およびプロジェクトを保存します。
- 他のプロジェクトを開いていない場合は，*save* の指定は無視します。

[使用例]

```
>>>project.Open("C:/test/test.mtpj")
True
>>>
```

B.3.4 CS+ Python 関数（ビルド・ツール用）

以下に，CS+ Python 関数（ビルド・ツール用）の一覧を示します。

表 B.4 CS+ Python 関数（ビルド・ツール用）

関数名	機能概要
build.All	プロジェクトのビルドを行います。
build.ChangeBuildMode	ビルド・モードの変更を行います。
build.Clean	プロジェクトのクリーンを行います。
build.File	指定したファイルのビルドを行います。
build.Update	ビルド・ツールの依存関係の更新を行います。

build.All

プロジェクトのビルドを行います。

[指定形式]

```
build.All(rebuild = False, waitBuild = True)
```

[引数]

引数	説明
rebuild	プロジェクトをリビルドするかどうかを指定します。 True ：プロジェクトをリビルドします。 False ：プロジェクトをビルドします（デフォルト）。
waitBuild	ビルドが完了するまで待つかどうかを指定します。 True ：ビルドが完了するまで待ちます（デフォルト）。 False ：ビルドが完了するのを待たずに，プロンプトを返します。

[戻り値]

- *waitBuild* に “True” を指定した場合
  ビルドが正常に完了した場合 ： True
  ビルドが失敗，またはキャンセルされた場合 ： False

- *waitBuild* に “False” を指定した場合
  ビルドの実行開始に成功した場合 ： True
  ビルドの実行開始に失敗した場合 ： False

[詳細説明]

- プロジェクトのビルドを行います。
  プロジェクトにサブプロジェクトを追加している場合は，サブプロジェクトのビルドも行います。
- *rebuild* に “True” を指定した場合は，プロジェクトをリビルドします。
- *waitBuild* に “False” を指定した場合は，ビルドが完了するのを待たずに，プロンプトを返します。
- ビルドの成否にかかわらず，ビルドの完了後に build.BuildCompleted イベントが発行されます。

[使用例]

```
>>>build.All()
True
>>>
```

build.ChangeBuildMode

ビルド・モードの変更を行います。

[指定形式]

```
build.ChangeBuildMode(buildmode)
```

[引数]

引数	説明 dd
buildmode	変更するビルド・モードを文字列で指定します。

[戻り値]

ビルド・モードの変更に成功した場合　: True
ビルド・モードの変更に失敗した場合　: False

[詳細説明]

- メイン・プロジェクト，およびサブプロジェクトのビルド・モードを *buildmode* で指定したビルド・モードに変更します。
- *buildmode* が存在しないプロジェクトの場合は，“DefaultBuild” を元にビルド・モードを作成して変更します。

[使用例]

```
>>>build.ChangeBuildMode("test_release")
True
>>>
```

build.Clean

プロジェクトのクリーンを行います。

[指定形式]

```
build.Clean(all = False)
```

[引数]

引数	説明 dd
all	サブプロジェクトも含めてプロジェクトのクリーンを行うかどうかを指定します。 True ：サブプロジェクトを含めたすべてのプロジェクトのクリーンを行います。 False ：アクティブ・プロジェクトのみクリーンを行います（デフォルト）。

[戻り値]

クリーンが正常に完了した場合 ： True
クリーンが失敗した場合 ： False

[詳細説明]

- プロジェクトのクリーンを行います（ビルド時の生成物を削除します）。
- *all* に “True” を指定した場合は，サブプロジェクトのクリーンも行います。

[使用例]

```
>>>build.Clean()
True
>>>
```

build.File

指定したファイルのビルドを行います。

[指定形式]

```
build.File(fileName, rebuild = False, waitBuild = True)
```

[引数]

引数	説明
fileName	ビルドするファイルを指定します。
rebuild	指定したファイルをリビルドするかどうかを指定します。 True ：指定したファイルをリビルドします。 False ：指定したファイルをビルドします（デフォルト）。
waitBuild	ビルドが完了するまで待つかどうかを指定します。 True ：ビルドが完了するまで待ちます（デフォルト）。 False ：ビルドが完了するのを待たずに，プロンプトを返します。

[戻り値]

- *waitBuild* に “True” を指定した場合
  ビルドが成功した場合 ： True
  ビルドが失敗した場合 ： False

- *waitBuild* に “False” を指定した場合
  ビルドの実行開始に成功した場合 ： True
  ビルドの実行開始に失敗した場合 ： False

[詳細説明]

- *fileName* で指定したファイルのビルドを行います。
- *rebuild* に “True” を指定した場合は，指定したファイルをリビルドします。
- *waitBuild* に “False” を指定した場合は，ビルドが完了するのを待たずに，プロンプトを返します。
- ビルドの完了後に build.BuildCompleted イベントが発行されます。

[使用例]

```
>>>build.File("C:/test/test.c")
True
>>>
```

build.Update

ビルド・ツールの依存関係の更新を行います。

[指定形式]

```
build.Update()
```

[引数]

なし

[戻り値]

なし

[詳細説明]

- ビルド時のファイルの依存関係を更新します。

[使用例]

```
>>>build.Update()
>>>
```

B.3.5 CS+ Python 関数（デバッグ・ツール用）

以下に，CS+ Python 関数（デバッグ・ツール用）の一覧を示します。

表 B.5 CS+ Python 関数（デバッグ・ツール用）

関数名	機能概要
debugger.ActionEvent.Delete	アクション・イベントを削除します。
debugger.ActionEvent.Disable	アクション・イベントの設定を無効にします。
debugger.ActionEvent.Enable	アクション・イベントの設定を有効にします。
debugger.ActionEvent.Get	アクション・イベント（Printf イベント）の結果を参照します。
debugger.ActionEvent.Information	アクション・イベント情報を表示します。
debugger.ActionEvent.Set	アクション・イベントを設定します。
debugger.Address	アドレス式を評価します。
debugger.Assemble.Disassemble	逆アセンブルを行います。
debugger.Assemble.LineAssemble	ライン・アセンブルを行います。
debugger.Breakpoint.Delete	ブレークポイントを削除します。
debugger.Breakpoint.Disable	ブレークポイントの設定を無効にします。
debugger.Breakpoint.Enable	ブレークポイントの設定を有効にします。
debugger.Breakpoint.Information	ブレークポイント情報を表示します。
debugger.Breakpoint.Set	ブレークポイントを設定します。
debugger.Connect	デバッグ・ツールに接続します。
debugger.DebugTool.Change	デバッグ・ツールを変更します。
debugger.DebugTool.GetType	デバッグ・ツールの情報を表示します。
debugger.Disconnect	デバッグ・ツールから切断します。
debugger.Download.Binary	バイナリ・ファイルをダウンロードします。
debugger.Download.Binary64Kb	64KB 以内用形式でバイナリ・ファイルをダウンロードします。
debugger.Download.BinaryBank	メモリ・バンク用形式でバイナリ・ファイルをダウンロードします。
debugger.Download.Coverage	カバレッジ・データをダウンロードします。
debugger.Download.Hex	ヘキサ・ファイルをダウンロードします。
debugger.Download.Hex64Kb	64KB 以内用形式でヘキサ・ファイルをダウンロードします。
debugger.Download.HexBank	メモリ・バンク用形式でヘキサ・ファイルをダウンロードします。
debugger.Download.HexIdTag	ID タグ付きヘキサ・ファイルをダウンロードします。
debugger.Download.Information	ダウンロード情報を表示します。
debugger.Download.LoadModule	ロード・モジュールをダウンロードします。
debugger.Erase	フラッシュ・メモリを消去します。
debugger.GetBreakStatus	ブレーク要因を表示します。
debugger.GetCpuStatus	現在の CPU の状態を表示します。
debugger.GetIeStatus	現在の IE の状態を表示します。

関数名	機能概要
debugger.GetIORList	IOR，SFR の一覧を表示します。
debugger.GetPC	PC 値を表示します。
debugger.Go	プログラムを継続して実行します。
debugger.Ie.GetValue debugger.Ie.SetValue	IE レジスタ，または DCU レジスタを設定／参照します。
debugger.IsConnected	デバッグ・ツールの接続状態を確認します。
debugger.IsRunning	デバッグ・ツールの実行状態を確認します。
debugger.Jump.File debugger.Jump.Address	各種パネルを表示します。
debugger.Map.Clear	マッピング設定をクリアします。
debugger.Map.Information	マップ情報を表示します。
debugger.Map.Set	メモリ・マッピングの設定を行います。
debugger.Memory.Copy	メモリをコピーします。
debugger.Memory.Fill	メモリを補填します。
debugger.Memory.Read	メモリを参照します。
debugger.Memory.ReadRange	指定した個数のメモリを参照します。
debugger.Memory.Write	メモリに書き込みます。
debugger.Memory.WriteRange	複数のデータをメモリに書き込みます。
debugger.Next	プロシージャ・ステップ実行を行います。
debugger.Register.GetValue	レジスタ，I/O レジスタ，SFR を参照します。
debugger.Register.SetValue	レジスタ，I/O レジスタ，SFR に値を設定します。
debugger.Reset	CPU をリセットします。
debugger.ReturnOut	現在の関数を呼び出したプログラムに戻るまで実行します。
debugger.Run	プログラムをリセット後に実行します。
debugger.Step	ステップ実行を行います。
debugger.Stop	デバッグ・ツールの実行を停止します。
debugger.Timer.Clear	条件タイマの計測結果をクリアします。
debugger.Timer.Delete	条件タイマを削除します。
debugger.Timer.Disable	条件タイマを無効にします。
debugger.Timer.Enable	条件タイマを有効にします。
debugger.Timer.Get	条件タイマの計測結果を参照します。
debugger.Timer.Information	条件タイマ情報を表示します。
debugger.Timer.Set	条件タイマを設定します。
debugger.Trace.Clear	トレース・メモリをクリアします。
debugger.Trace.Delete	条件トレースを削除します。
debugger.Trace.Disable	条件トレースを無効にします。

関数名	機能概要
debugger.Trace.Enable	条件トレースを有効にします。
debugger.Trace.Get	トレース・データをダンプします。
debugger.Trace.Information	条件トレース情報を表示します。
debugger.Trace.Set	条件トレースを設定します。
debugger.Upload.Binary	メモリ・データをバイナリ形式で保存します。
debugger.Upload.Coverage	カバレッジ・データを保存します。
debugger.Upload.Intel	メモリ・データをインテル形式で保存します。
debugger.Upload.IntelIdTag	メモリ・データを ID タグ付きインテル形式で保存します。
debugger.Upload.Motorola	メモリ・データをモトローラ形式で保存します。
debugger.Upload.MotorolaIdTag	メモリ・データを ID タグ付きモトローラ形式で保存します。
debugger.Upload.Tektronix	メモリ・データをテクトロニクス形式で保存します。
debugger.Upload.TektronixIdTag	メモリ・データを ID タグ付きテクトロニクス形式で保存します。
debugger.Watch.GetValue	変数値を参照します。
debugger.Watch.SetValue	変数値を設定します。
debugger.Where	スタックのバック・トレースを表示します。
debugger.Whereami	ロケーションを表示します。
debugger.XCoverage.Clear	カバレッジ・メモリをクリアします。
debugger.XCoverage.GetCoverage	カバレッジを取得します。
debugger.XRunBreak.Delete	XRunBreak の設定情報を削除します。
debugger.XRunBreak.Refer	XRunBreak の設定情報を表示します。
debugger.XRunBreak.Set	XRunBreak 情報を設定します。
debugger.XTime	Go-Break 間の時間情報を表示します。
debugger.XTrace.Clear	トレース・メモリをクリアします。
debugger.XTrace.Dump	トレース・データをダンプします。

debugger.ActionEvent.Delete

アクション・イベントを削除します。

[指定形式]

```
debugger.ActionEvent.Delete(actionEventNumber = "")
```

[引数]

引数	説明
actionEventNumber	削除するアクション・イベント番号を指定します。

[戻り値]

アクション・イベントの削除に成功した場合　: True
アクション・イベントの削除に失敗した場合　: False

[詳細説明]

- *actionEventNumber* で指定したアクション・イベントを削除します。
- *actionEventNumber* を指定しない場合は，すべてのアクション・イベント番号のアクション・イベントを削除します。

[使用例]

```
>>>debugger.ActionEvent.Delete(1)
True
>>>debugger.ActionEvent.Delete()
True
>>>
```

debugger.ActionEvent.Disable

アクション・イベントの設定を無効にします。

[指定形式]

```
debugger.ActionEvent.Disable(actionEventNumber = "")
```

[引数]

引数	説明
actionEventNumber	無効にするアクション・イベント番号を指定します。

[戻り値]

アクション・イベントの設定の無効に成功した場合　：True
アクション・イベントの設定の無効に失敗した場合　：False

[詳細説明]

- *actionEventNumber* で指定したアクション・イベントを無効にします。
- *actionEventNumber* を指定しない場合は，すべてのアクション・イベント番号のアクション・イベントを無効にします。

[使用例]

```
>>>debugger.ActionEvent.Disable(1)
True
>>>debugger.ActionEvent.Disable()
True
>>>
```

debugger.ActionEvent.Enable

アクション・イベントの設定を有効にします。

[指定形式]

```
debugger.ActionEvent.Enable(actionEventNumber = "")
```

[引数]

引数	説明
actionEventNumber	有効にするアクション・イベント番号を指定します。

[戻り値]

アクション・イベントの設定の有効に成功した場合　：True
アクション・イベントの設定の有効に失敗した場合　：False

[詳細説明]

- *actionEventNumber* で指定したアクション・イベントを有効にします。
- *actionEventNumber* を指定しない場合は，すべてのアクション・イベント番号のアクション・イベントを有効にします。

[使用例]

```
>>>debugger.ActionEvent.Enable(1)
True
>>>debugger.ActionEvent.Enable()
True
>>>
```

debugger.ActionEvent.Get

アクション・イベント（Printf イベント）の結果を参照します。

[指定形式]

```
debugger.ActionEvent.Get(output = "")
```

[引数]

引数	説明
output	アクション・イベントの結果を出力する際に付与する文字列を指定します（デフォルト：指定なし）。 なお，本引数を指定している場合に，本引数と一致する結果のみを取得したい場合に指定します。

[戻り値]

アクション・イベントの結果リスト（詳細は ActionInfo クラスを参照してください）

[詳細説明]

- アクション・イベント（Printf イベント）の条件で設定したアドレスを実行した際の結果を Python コンソール内で保持し，この debugger.ActionEvent.Get が呼び出されたタイミングで，それまでに保持していた結果をすべて参照します。
- *output* を指定している場合に，*output* と一致する結果のみを出力します。比較は完全一致で比較されます。
- *output* を指定しない場合は，蓄積したすべてのアクション・イベントの結果を出力します。
- アクション・イベントが発生したタイミングで結果を取得したい場合は，Hook を使用してください。また，Python コンソール内で保持する結果の最大数については，debugger.ActionEvent.GetLine プロパティを参照してください。

注意 参照後に Python コンソール内で保持したアクション・イベントの結果は初期化されます。そのため，一度参照した結果を再度参照することはできません。

- アクション・イベントの結果を，以下の形式で表示します。

```
出力する際に付与する文字列 変数式
```

［使用例］

```
>>>ae = ActionEventCondition()
>>>ae.Address = "main"
>>>ae.Output = "result "
>>>ae.Expression = "chData"
>>>ae.ActionEventType = ActionEventType.Printf
>>>ae_number = debugger.ActionEvent.Set(ae)
        :
>>>out = debugger.ActionEvent.Get()
result chData=0x64
result chData=0x65
result chData=0x66
>>>print out[0].Address
main
>>>print out[0].Expression
chData=0x64
```

debugger.ActionEvent.Information

アクション・イベント情報を表示します。

[指定形式]

```
debugger.ActionEvent.Information()
```

[引数]

なし

[戻り値]

アクション・イベント情報のリスト（詳細は ActionEventInfo クラスを参照してください）

[詳細説明]

- 設定されているアクション・イベントの情報を，以下の形式で表示します。
  - Printf イベントの場合

    アクション・イベント番号 アクション・イベント名 状態 アドレス 出力する際に付与する文字列 変数式

  - 割り込みイベントの場合

    アクション・イベント番号 アクション・イベント名 状態 アドレス Interrupt vector: *割り込みベクタ番号* Priority level: *割り込み優先順位*

[使用例]

```
>>>ai = debugger.ActionEvent.Information()
1 Python アクション・イベント 0001 Enable main results: chData
2 Python アクション・イベント 0002 Disable sub Interrupt vector: 0x1c Priority level: 7
>>>print ai[0].Number
1
>>>print ai[0].Name
Python アクション・イベント 0001
>>>
```

debugger.ActionEvent.Set

アクション・イベントを設定します。

[指定形式]

```
debugger.ActionEvent.Set(ActionEventCondition)
```

[引数]

引数	説明
ActionEventCondition	アクション・イベントの条件を指定します。 アクション・イベントの作成については，ActionEventCondition クラスを参照してください。

[戻り値]

設定したアクション・イベント番号（数値）

[詳細説明]

- *ActionEventCondition* で指定されている内容に従って，アクション・イベントを設定します。
- 設定したアクション・イベントは，以下の名前で登録されます。

Python アクション・イベント*数値*

[使用例]

```
>>>ae = ActionEventCondition()
>>>ae.Address = "main"
>>>ae.Output = "chData = "
>>>ae.Expression = "chData"
>>>ae.ActionEventType = ActionEventType.Printf
>>>ae_number = debugger.ActionEvent.Set(ae)
1
>>>print ae_number
1
```

debugger.Address

アドレス式を評価します。

[指定形式]

```
debugger.Address(expression)
```

[引数]

引数	説明
expression	アドレス式を指定します。

[戻り値]

変換したアドレス（数値）

[詳細説明]

- *expression* で指定したアドレス式をアドレスに変換します。

注意 1. CubeSuite+.exe の起動オプションでスクリプトを指定して実行する場合，デバッグ・ツールと接続するまでシンボル変換機能は使用できません。
つまり，本関数は使用できませんので，接続後に実行してください。

注意 2. アドレス式にロード・モジュール名やファイル名を指定する場合は，ダブルクォーテーション（" "）で囲む必要がある場合があります。詳細については，「CS+ 統合開発環境 ユーザーズマニュアル デバッグ・ツール編」を参照してください。

例 ファイル名 C:¥path¥test.c，関数 sub を指定する場合

```
"¥"C:/path/test.c¥"#sub"
```

または

```
"¥"C:¥¥path¥¥test.c¥"#sub"
```

[使用例]

```
>>>debugger.Address("main")
0x4088
>>>debugger.Address("main + 1")
0x4089
>>>
```

debugger.Assemble.Disassemble

逆アセンブルを行います。

[指定形式]

```
debugger.Assemble.Disassemble(address, number = 1, code = True)
```

[引数]

引数	説明
address	逆アセンブルを開始するアドレスを指定します。
number	表示行数を指定します（デフォルト：1）。
code	命令コードを表示するかどうかを指定します。 True ：命令コードを表示します（デフォルト）。 False ：命令コードを表示しません。

[戻り値]

逆アセンブル結果のリスト（詳細は DisassembleInfo クラスを参照してください）

[詳細説明]

- *address* で指定したアドレスから逆アセンブルします。
- *number* を指定した場合は，指定した数分の行を表示します。
- *code* に “False” を指定した場合は，命令コードを表示しません。
- *address* に “.” を指定した場合は，直前の逆アセンブルの続きのアドレスを指定したと解釈します。

[使用例]

```
>>>debugger.Assemble.Disassemble("main")
0x00004088  F545    br _TestInit+0x8e
>>>debugger.Assemble.Disassemble("main", 2)
0x00004088  F545    br _TestInit+0x8e
0x0000408A  0A5A    mov 0xa, r11
>>>debugger.Assemble.Disassemble("main", 5, False)
0x00004088  br _TestInit+0x8e
0x0000408A  mov 0xa, r11
0x0000408C  movea 0x19, r0, r13
0x00004090  mov r13, r12
0x00004092  movhi 0xffff, gp, r1
>>>
```

debugger.Assemble.LineAssemble

ライン・アセンブルを行います。

[指定形式]

```
debugger.Assemble.LineAssemble(address, code)
```

[引数]

引数	説明
address	アセンブルを開始するアドレスを指定します。
code	アセンブルする文字列を指定します。

[戻り値]

ライン・アセンブルに成功した場合 : True
ライン・アセンブルに失敗した場合 : False

[詳細説明]

- *code* で指定した文字列を *address* で指定したアドレスからアセンブルします。
- *address* に “.” を指定した場合は，直前のアセンブルの続きのアドレスを指定したと解釈します。

[使用例]

```
>>>debugger.Assemble.Disassemble("main")
0x00004088  F545    br _TestInit+0x8e
>>>debugger.Assemble.Disassemble(".")
0x0000408A  0A5A    mov 0xa, r11
>>>debugger.Assemble.LineAssemble("main", "mov r13, r12")
True
>>>debugger.Assemble.Disassemble("main", 1, False)
0x00004088  mov r13, r12
>>>
```

debugger.Breakpoint.Delete

ブレークポイントを削除します。

[指定形式]

```
debugger.Breakpoint.Delete(breakNumber = "")
```

[引数]

引数	説明
breakNumber	削除するブレーク・イベント番号を指定します。

[戻り値]

ブレークポイントの削除に成功した場合　: True
ブレークポイントの削除に失敗した場合　: False

[詳細説明]

- *breakNumber* で指定したブレーク・イベントを削除します。
- *breakNumber* を指定しない場合は，すべてのブレーク・イベント番号のブレークを削除します。

[使用例]

```
>>>debugger.Breakpoint.Enable(1)
True
>>>debugger.Breakpoint.Disable(1)
True
>>>debugger.Breakpoint.Delete(1)
True
>>>
```

debugger.Breakpoint.Disable

ブレークポイントの設定を無効にします。

[指定形式]

```
debugger.Breakpoint.Disable(breakNumber = "")
```

[引数]

引数	説明
breakNumber	無効にするブレーク・イベント番号を指定します。

[戻り値]

ブレークポイントの設定の無効に成功した場合　: True
ブレークポイントの設定の無効に失敗した場合　: False

[詳細説明]

- *breakNumber* で指定したブレーク・イベントを無効にします。
- *breakNumber* を指定しない場合は，すべてのブレーク・イベント番号のブレークを無効にします。

[使用例]

```
>>>debugger.Breakpoint.Enable(1)
True
>>>debugger.Breakpoint.Disable(1)
True
>>>debugger.Breakpoint.Delete(1)
True
>>>
```

debugger.Breakpoint.Enable

ブレークポイントの設定を有効にします。

[指定形式]

```
debugger.Breakpoint.Enable(breakNumber = "")
```

[引数]

引数	説明
breakNumber	有効にするブレーク・イベント番号を指定します。

[戻り値]

ブレークポイントの設定の有効に成功した場合　: True
ブレークポイントの設定の有効に失敗した場合　: False

[詳細説明]

- *breakNumber* で指定したブレーク・イベントを有効にします。
- *breakNumber* を指定しない場合は，すべてのブレーク・イベント番号のブレークを有効にします。

[使用例]

```
>>>debugger.Breakpoint.Enable(1)
True
>>>debugger.Breakpoint.Disable(1)
True
>>>debugger.Breakpoint.Delete(1)
True
>>>
```

debugger.Breakpoint.Information

ブレークポイント情報を表示します。

[指定形式]

```
debugger.Breakpoint.Information()
```

[引数]

なし

[戻り値]

ブレークポイント情報のリスト（詳細は BreakpointInfo クラスを参照してください）

[詳細説明]

- 設定されているブレークポイントの情報を，以下の形式で表示します。

```
ブレーク・イベント番号  ブレーク名  状態  アドレス・ロケーション
```

[使用例]

```
>>>debugger.Breakpoint.Information()
   1 Python ブレーク 0001 Enable  0x000002dc
   2 ブレーク 0001 Enable  test1.c#_sub1
   3 Python ブレーク 0002 Enable  0x000002ec
   4 ブレーク 0002 Enable  test1.c#_sub1+10
>>>
```

debugger.Breakpoint.Set

ブレークポイントを設定します。

[指定形式]

```
debugger.Breakpoint.Set(BreakCondition)
```

[引数]

引数	説明
BreakCondition	ブレーク条件を指定します。 ブレーク条件の作成については，BreakCondition クラスを参照してください。

[戻り値]

設定したブレーク・イベント番号（数値）

[詳細説明]

- *BreakCondition* で指定されている内容に従って，ブレークポイントを設定します。
- ブレーク名は，“Python ブレーク *xxxx*”（*xxxx*：4 桁の数字）となります。

[使用例]

```
>>>Condition = BreakCondition()
>>>Condition.Address = "main"
>>>breakNumber = debugger.Breakpoint.Set(Condition)
1
>>>print breakNumber
1
>>>debugger.Breakpoint.Information()
   1 Python ブレーク 0001 Enable  0x000002dc
```

debugger.Connect

デバッグ・ツールに接続します。

[指定形式]

```
debugger.Connect()
```

[引数]

なし

[戻り値]

デバッグ・ツールとの接続に成功した場合 : True
デバッグ・ツールとの接続に失敗した場合 : False

[詳細説明]

- デバッグ・ツールに接続します。

[使用例]

```
>>>debugger.Connect()
True
>>>
```

debugger.DebugTool.Change

デバッグ・ツールを変更します。

[指定形式]

```
debugger.DebugTool.Change(debugTool)
```

[引数]

引数	説明	
debugTool	変更するデバッグ・ツールを指定します。 指定可能なデバッグ・ツールを以下に示します。	
	種類	説明
	DebugTool.Simulator	シミュレータ
	DebugTool.Minicube	MINICUBE
	DebugTool.Minicube2	MINICUBE2（シリアル接続）
	DebugTool.Minicube2Jtag	MINICUBE2（JTAG 接続）
	DebugTool.Iecube	IECUBE
	DebugTool.Iecube2	IECUBE2
	DebugTool.E1Jtag	E1（JTAG 接続）
	DebugTool.E1Serial	E1（シリアル接続）
	DebugTool.E1Lpd	E1（LPD 接続）
	DebugTool.E20Jtag	E20（JTAG 接続）
	DebugTool.E20Serial	E20（シリアル接続）
	DebugTool.E20Lpd	E20（LPD 接続）

[戻り値]

デバッグ・ツールの変更に成功した場合　：True
デバッグ・ツールの変更に失敗した場合　：False

[詳細説明]

- *DebugTool* で指定したデバッグ・ツールに変更します。
  ただし，変更可能なデバッグ・ツールは，使用するデバイスによって異なります。変更可能なデバッグ・ツールは，プロジェクト・ツリーで［デバッグ・ツール］を選択し，コンテキストメニューの［使用するデバッグ・ツール］で確認してください。

注意　選択エミュレータできないエミュレータも指定できてしまいます。
CS+ のデバッグ・ツールで選択できるエミュレータのみ指定してください。

［使用例］

```
>>>debugger.DebugTool.Change(DebugTool.Simulator)
True
>>>
```

debugger.DebugTool.GetType

デバッグ・ツールの情報を表示します。

[指定形式]

```
debugger.DebugTool.GetType()
```

[引数]

なし

[戻り値]

デバッグ・ツールの種類の定数

デバッグ・ツールの種類の定数	説明
Simulator	シミュレータ
Minicube	MINICUBE
Minicube2	MINICUBE2（シリアル接続）
Minicube2Jtag	MINICUBE2（JTAG 接続）
Iecube	IECUBE
Iecube2	IECUBE2
E1Jtag	E1（JTAG 接続）
E1Serial	E1（シリアル接続）
E1Lpd	E1（LPD 接続）
E20Jtag	E20（JTAG 接続）
E20Serial	E20（シリアル接続）
E20Lpd	E20（LPD 接続）

[詳細説明]

- デバッグ・ツールの情報を表示します。

[使用例]

```
>>>debugType = debugger.DebugTool.GetType()
Minicube2
>>>if debugType != DebugTool.Simulator:
... debugger.DebugTool.Change(DebugTool.Simulator)
...
>>>
```

debugger.Disconnect

デバッグ・ツールから切断します。

[指定形式]

```
debugger.Disconnect()
```

[引数]

なし

[戻り値]

デバッグ・ツールからの切断に成功した場合　: True
デバッグ・ツールからの切断に失敗した場合　: False

[詳細説明]

- デバッグ・ツールから切断します。

[使用例]

```
>>>debugger.Disconnect()
True
>>>
```

debugger.Download.Binary

バイナリ・ファイルをダウンロードします。

[指定形式]

```
debugger.Download.Binary(fileName, address, append = False, flashErase = False)
```

[引数]

引数	説明
fileName	ダウンロード・ファイルを指定します。
address	ダウンロード開始アドレスを指定します。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。
fileName，*address* のみを指定することはできません。

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- バイナリ形式のデータをダウンロードします。

[使用例]

```
>>>debugger.Download.Binary("C:/test/testModule.bin",  0x1000, False)
True
>>>debugger.Download.Binary("C:/test/testModule2.bin", 0x2000, True)
False
>>>
```

debugger.Download.Binary64Kb

64KB 以内用形式でバイナリ・ファイルをダウンロードします。

[指定形式]

```
debugger.Download.Binary64Kb(fileName, address, append = False, flashErase = False)
```

[引数]

引数	説明
fileName	ダウンロード・ファイルを指定します。
address	ダウンロード開始アドレスを指定します。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。
fileName，*address* のみを指定することはできません。

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- メモリ・バンク使用時に，64KB 以内用形式でバイナリ・ファイルをダウンロードします。

[使用例]

```
>>>debugger.Download.Binary64Kb("C:/test/testModule.bin",  0x1000, False)
True
>>>debugger.Download.Binary64Kb("C:/test/testModule2.bin", 0x2000, True)
False
>>>
```

debugger.Download.BinaryBank

メモリ・バンク用形式でバイナリ・ファイルをダウンロードします。

[指定形式]

```
debugger.Download.BinaryBank(fileName, address, append = False, flashErase = False)
```

[引数]

引数	説明
fileName	ダウンロード・ファイルを指定します。
address	ダウンロード開始アドレスを指定します。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。
fileName，*address* のみを指定することはできません。

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- メモリ・バンク使用時に，メモリ・バンク用形式でバイナリ・ファイルをダウンロードします。

[使用例]

```
>>>debugger.Download.BinaryBank("C:/test/testModule.bin",  0x1000, False)
True
>>>debugger.Download.BinaryBank("C:/test/testModule2.bin", 0x2000, True)
False
>>>
```

debugger.Download.Coverage

カバレッジ・データをダウンロードします。【IECUBE】【IECUBE2】【シミュレータ】

[指定形式]

```
debugger.Download.Coverage(fileName)
```

[引数]

引数	説明
fileName	カバレッジ・データ・ファイルを指定します。

[戻り値]

ダウンロードに成功した場合　: True
ダウンロードに失敗した場合　: False

[詳細説明]

- カバレッジ・データをダウンロードします。

[使用例]

```
>>>debugger.Download.Coverage("C:/test/testModule.csrcv")
True
>>>
```

debugger.Download.Hex

ヘキサ・ファイルをダウンロードします。

[指定形式]

```
debugger.Download.Hex(fileName, offset = 0, append = False, flashErase = False)
```

[引数]

引数	説明
fileName	ダウンロード・ファイルを指定します。
offset	オフセットを指定します（デフォルト：0）。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。
fileName，*offset* のみを指定することはできません。

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- ヘキサ形式のデータをダウンロードします。

[使用例]

```
>>>debugger.Download.Hex("C:/test/testModule.hex")
True
>>>
```

debugger.Download.Hex64Kb

64KB 以内用形式でヘキサ・ファイルをダウンロードします。

[指定形式]

```
debugger.Download.Hex64Kb(fileName, offset = 0, append = False, flashErase = False)
```

[引数]

引数	説明
fileName	ダウンロード・ファイルを指定します。
offset	オフセットを指定します（デフォルト：0）。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。
fileName，*offset* のみを指定することはできません。

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- メモリ・バンク使用時に，64KB 以内用形式でヘキサ・ファイルをダウンロードします。

[使用例]

```
>>>debugger.Download.Hex64Kb("C:/test/testModule.hex")
True
>>>
```

debugger.Download.HexBank

メモリ・バンク用形式でヘキサ・ファイルをダウンロードします。

[指定形式]

```
debugger.Download.HexBank(fileName, offset = 0, append = False, flashErase = False)
```

[引数]

引数	説明
fileName	ダウンロード・ファイルを指定します。
offset	オフセットを指定します（デフォルト：0）。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。
fileName，*offset* のみを指定することはできません。

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- メモリ・バンク使用時に，メモリ・バンク用形式でヘキサ・ファイルをダウンロードします。

[使用例]

```
>>>debugger.Download.HexBank("C:/test/testModule.hex")
True
>>>debugger.Download.HexBank("C:/test/testModule2.hex", 0x1000, True)
False
>>>
```

debugger.Download.HexIdTag

ID タグ付きヘキサ・ファイルをダウンロードします。

[指定形式]

```
debugger.Download.HexIdTag(fileName, offset = 0, append = False, flashErase = False)
```

[引数]

引数	説明
fileName	ダウンロード・ファイルを指定します。
offset	オフセットを指定します（デフォルト：0）。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。
fileName，*offset* のみを指定することはできません。

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- ID タグ付きヘキサ・ファイルをダウンロードします。

[使用例]

```
>>>debugger.Download.HexIdTag("C:/test/testModule.hex")
True
>>>debugger.Download.HexIdTag("C:/test/testModule2.hex", 0x1000, True)
False
>>>
```

debugger.Download.Information

ダウンロード情報を表示します。

[指定形式]

```
debugger.Download.Information()
```

[引数]

なし

[戻り値]

ダウンロード情報のリスト（詳細は DownloadInfo クラスを参照してください）

[詳細説明]

- ダウンロード情報を，以下の形式で表示します。

ダウンロード番号： *ダウンロード・ファイル名*

[使用例]

```
>>>debugger.Download.Information()
   1: DefaultBuild¥test.lmf
```

debugger.Download.LoadModule

ロード・モジュールをダウンロードします。

[指定形式]

```
debugger.Download.LoadModule(fileName = "", downloadOption = DownloadOption.Both,
append = False, flashErase = False)
```

[引数]

引数	説明	
fileName	ダウンロード・ファイルを指定します。	
downloadOption	オプションを指定します。 指定可能なオプションを以下に示します。	
	種類	説明
	DownloadOption.NoSymbol	シンボル情報を読み込みません。
	DownloadOption.SymbolOnly	シンボル情報のみ読み込みます。
	DownloadOption.Both	シンボル情報とオブジェクト情報の両方を読み込みます（デフォルト）。
append	追加ダウンロードするかどうかを指定します。 True ：追加ダウンロードします。 False ：上書きダウンロードします（デフォルト）。	
flashErase	ダウンロード前にフラッシュ・メモリを初期化するかどうかを指定します。 True ：ダウンロード前にフラッシュ・メモリを初期化します。 False ：ダウンロード前にフラッシュ・メモリを初期化しません（デフォルト）。	

[戻り値]

ダウンロードに成功した場合 ： True
ダウンロードに失敗した場合 ： False

[詳細説明]

- ロード・モジュールをダウンロードします。
- *fileName* を指定しない場合は，デバッグ・ツールのプロパティ パネルの［ダウンロード・ファイル設定］タブに指定されているファイルをダウンロードします。
- *downloadOption* を指定した場合，指定した内容に従って処理を行います。

[使用例]

```
>>>debugger.Download.LoadModule("C:/test/testModule.lmf")
True
>>>debugger.Download.LoadModule("C:/test/testModule2.lmf", DownloadOption.SymbolOnly,
True)
False
>>>
```

debugger.Erase

フラッシュ・メモリを消去します。

[指定形式]

```
debugger.Erase(eraseOption = EraseOption.Code)
```

[引数]

引数	説明	
eraseOption	オプションを指定します。 指定可能なオプションを以下に示します。	
	種類	説明
	EraseOption.Code	コード・フラッシュ・メモリを消去します（デフォルト）。
	EraseOption.Data	データフラッシュ・メモリを消去します。
	EraseOption.External	外部空間にあるフラッシュ・メモリを消去します。

注意 IECUBE，IECUBE2，シミュレータにはコード・フラッシュ・メモリの消去機能がないため，IECUBE，IECUBE2，シミュレータを使用する場合は，*eraseOption* の省略，および EraseOption.Code の指定はできません。

[戻り値]

フラッシュ・メモリの消去に成功した場合 ： True
フラッシュ・メモリの消去に失敗した場合 ： False

[詳細説明]

- *eraseOption* で指定したフラッシュ・メモリを消去します。

[使用例]

```
>>>debugger.Erase()
True
>>>debugger.Erase(EraseOption.External)
False
>>>
```

debugger.GetBreakStatus

ブレーク要因を表示します。

[指定形式]

```
debugger.GetBreakStatus()
```

[引数]

なし

[戻り値]

ブレーク要因の文字列（[詳細説明] 参照）

備考 1. BreakStatus という enum 定義の文字列部分を返します。

備考 2. 条件判断する場合は，“BreakStatus. *文字列*” と記述してください。

[詳細説明]

- ブレーク要因を表示します。
  実行中は，“None” になります。

ブレーク要因の文字列	説明	78K0			RL78,78K0R			V850			
		Iecube	Minicube2 注 1	Simulator	Iecube	Minicube2 注 1	Simulator	Iecube	Minicube 注 2	Minicube2 注 1	Simulator
None	ブレークしていない	○	○	－	○	○	－	○	○	○	－
Manual	強制ブレーク	○	○	○	○	○	○	○	○	○	○
Event	イベントによるブレーク	○	○	○	○	○	○	○	○	○	○
Software	ソフトウエア・ブレーク	○	○	－	○	○	－	○	○	○	－
TraceFull	トレース・フルによるブレーク	○	－	○	○	－	○	○	－	－	○
TraceDelay	トレース・ディレイによるブレーク	○	－	－	○	－	－	－	－	－	－
NonMap	ノンマップ・エリアをアクセス	○	－	○	○	－	○	○	－	－	○
WriteProtect	ライト・プロテクト領域に対してライト	○	－	○	○	－	○	○	－	－	○
ReadProtect	リード・プロテクト領域からリード	○	－	－	－	－	－	－	－	－	－
SfrIllegal	SFR に対してイリーガルなアクセス	○	－	－	－	－	－	－	－	－	－
SfrReadProtect	リード禁止の SFR からリード	○	－	－	○	－	－	－	－	－	－

ブレーク要因の文字列	説明	78K0			RL78,78K0R			V850			
		Iecube	Minicube2 注1	Simulator	Iecube	Minicube2 注1	Simulator	Iecube	Minicube 注2	Minicube2 注1	Simulator
SfrWriteProtect	ライト禁止の SFR に対してライト	○	－	－	○	－	－	－	－	－	－
IorIllegal	周辺 I/O レジスタに対してイリーガルなアクセス（アドレス付き）	－	－	－	－	－	－	○	－	－	－
StackOverflow	スタック・オーバーフローによるブレーク	○	－	－	○	－	－	－	－	－	－
StackUnderflow	スタック・アンダーフローによるブレーク	○	－	－	○	－	－	－	－	－	－
UninitializeStackPointer	スタック・ポインタ初期化忘れによるブレーク	○	－	－	○	－	－	－	－	－	－
UninitializeMemoryRead	初期化していないメモリをリードした	○	－	－	○	－	－	－	－	－	－
TimerOver	実行時間オーバーを検出した	○	－	－	○	－	－	○	－	－	－
UnspecifiedIllegal	周辺チップ機能に関するユーザ・プログラムの不正動作が発生	○	－	－	○	－	－	－	－	－	－
ImsIxsIllegal	IMS，IXS レジスタ不正書き込みによるブレーク	○	－	－	－	－	－	－	－	－	－
BeforeExecution	実行前ブレーク	○	－	－	○	－	－	－	－	－	－
SecurityProtect	セキュリティ保護領域に対してアクセス	－	－	－	－	－	－	－	－	－	－
FlashMacroService	フラッシュ・マクロ・サービス中	－	－	－	－	－	－	－	○	○	－
RetryOver	RETRY 回数オーバ・ブレーク	○	－	－	－	－	－	－	－	－	－
FlashIllegal	フラッシュ・イリーガル・ブレーク	○	－	－	○	－	－	－	－	－	－
Peripheral	周辺からのブレーク	○	－	－	○	－	－	－	－	－	－
WordMissAlignAccess	奇数番地に対するワード・アクセスを行った	－	－	－	○	－	○	－	－	－	－
Temporary	テンポラリ・ブレーク	○	○	○	○	○	○	○	○	○	○
Escape	エスケープ・ブレークによるブレーク	－	－	－	－	－	－	○	○	○	－
Fetch	ガード領域，フェッチ禁止領域をフェッチした	○	－	－	○	－	－	－	－	－	－
IRamWriteProtect	IRAM ガード領域の書き込み（アドレス付き）注3	－	－	－	－	－	－	○	－	－	－

ブレーク要因の文字列	説明	78K0			RL78,78K0R			V850			
		Iecube	Minicube2 注1	Simulator	Iecube	Minicube2 注1	Simulator	Iecube	Minicube 注2	Minicube2 注1	Simulator
IllegalOpcodeTrap	不正命令例外発生によるブレーク	－	－	－	－	－	－	○	△ 注6	－	－
Step	ステップ実行・ブレーク注4	○	○	○	○	○	○	－	－	－	○
FetchGuard	フェッチガード・ブレーク注4	○	－	－	○	－	－	－	－	－	－
TraceStop	トレース・ストップ注4	○	－	－	○	－	－	－	－	－	－
ExecutionFails	実行しようとして失敗注5	○	○	－	○	○	－	○	○	○	－

注 1. Minicube2，E1Serial，E20Serial のすべてに該当します。

注 2. Minicube，E1Jtag，E20Jtag，Minicube2Jtag のすべてに該当します。

注 3. ブレーク時に IRAM ガード領域のベリファイ・チェックを行い，値が書き換わっていた場合です（該当アドレスが複数ある場合は，最初のアドレスのみ表示します）。

注 4. トレース時のみのブレーク要因です。

注 5. ブレーク時のみのブレーク要因です。

注 6. V850-MINICUBE で ET コア系デバイス（ME2 など）で，実行後イベントを使用した場合は表示しません。

ブレーク要因の文字列	説明	RX		V850E2				RH850		
		E1Jtag, E1Serial E20Jtag, E20Serial	Simulator	Iecube2	Minicube 注2	Minicube2 注1	Simulator	Full-spec emulator	E1 / E20	SIM
None	ブレークしていない	○	－	○	○	○	－	－	－	－
Manual	強制ブレーク	○	○	○	○	○	○	○	○	○
Event	イベントによるブレーク	○	○	○	○	○	○	○	○	○
Software	ソフトウエア・ブレーク	○	－	○	○	○	－	○	○	－
TraceFull	トレース・フルによるブレーク	○	○	○	－	－	○	○	○	○
NonMap	ノンマップ・エリアをアクセス	－	－	－	－	－	○	－	－	○
WriteProtect	ライト・プロテクト領域に対してライト	－	－	－	－	－	○	－	－	○
TimerOver	実行時間オーバーを検出した	－	－	○	○	－	－	－	－	－
FlashMacroService	フラッシュ・マクロ・サービス中	－	－	○	○	○	－	－	－	－
Temporary	テンポラリ・ブレーク	○	○	○	○	○	○	○	○	○
IllegalOpcodeTrap	不正命令例外発生によるブレーク	－	－	○	○	－	－	－	－	－
Step	ステップ実行・ブレーク注3	○	－	－	－	－	○	○	○	○
ExecutionFails	実行しようとして失敗注4	○	－	○	○	○	－	－	－	－

ブレーク要因の文字列	説明	RX		V850E2				RH850		
		E1Jtag, E1Serial E20Jtag, E20Serial	Simulator	Iecube2	Minicube 注 2	Minicube2 注 1	Simulator	Full-spec emulator	E1 / E20	SIM
WaitInstruction	WAIT 命令実行によるブレーク	―	○	―	―	―	―	―	―	―
UndefinedInstructionException	未定義命令例外発生によるブレーク	―	○	―	―	―	―	―	―	―
PrivilegeInstructionException	特権命令例外発生によるブレーク	―	○	―	―	―	―	―	―	―
AccessException	アクセス例外発生によるブレーク	―	○	―	―	―	―	―	―	―
FloatingPointException	浮動小数点例外発生によるブレーク	―	○	―	―	―	―	―	―	―
InterruptException	割り込み発生によるブレーク	―	○	―	―	―	―	―	―	―
IntInstructionException	INT 命令例外発生によるブレーク	―	○	―	―	―	―	―	―	―
BrkInstructionException	BRK 命令例外発生によるブレーク	―	○	―	―	―	―	―	―	―
IOFunctionSimulationBreak	周辺機能シミュレーションによるブレーク	―	○	―	―	―	―	―	―	―
IllegalMemoryAccessBreak	不正なメモリ・アクセスによるブレーク	―	○	―	―	―	―	―	―	―
StreamIoError	ストリーム入出力エラーによるブレーク	―	○	―	―	―	―	―	―	―
CoverageMemoryAllocationFailure	カバレッジ・メモリの確保に失敗	―	○	―	―	―	―	―	―	―
TraceMemoryAllocationFailure	トレース・メモリの確保に失敗	―	○	―	―	―	―	―	―	―
StepCountOver	ステップ回数オーバー	―	―	―	―	―	―	○	○	○
DebuggingInformationAcquisitionFailure	デバッグ情報取得に失敗	―	―	―	―	―	―	○	○	○

注 1. Minicube2，E1Serial，E20Serial のすべてに該当します。

注 2. Minicube，E1Jtag，E20Jtag，Minicube2Jtag のすべてに該当します。

注 3. トレース時のみのブレーク要因です。

注 4. ブレーク時のみのブレーク要因です。

［使用例］

```
>>>debugger.GetBreakStatus()
Temporary
>>>a = debugger.GetBreakStatus()
Temporary
>>>print a
Temporary
>>>if (debugger.GetBreakStatus() == BreakStatus.Temporary):
... print "テンポラリ・ブレークしました"
...
Temporary
テンポラリ・ブレークしました
>>>
```

debugger.GetCpuStatus

現在の CPU の状態を表示します。

[指定形式]

```
debugger.GetCpuStatus()
```

[引数]

なし

[戻り値]

現在の CPU の状態（文字列）

CPU の状態	説明
Hold	バス・ホールド中
HoldStopIdle	バス・ホールド / ソフトウエア STOP/ ハードウエア STOP/IDLE モード中
PowOff	ターゲットに電源が供給されていない状態
Reset	リセット状態
Standby	スタンバイ・モード中
Stop	STOP モード中
StopIdle	ソフトウエア STOP/ ハードウエア STOP/IDLE モード中
Wait	ウエイト状態
Halt	HALT モード中
Sleep	スリープ状態中
None	該当なし

[詳細説明]

- 現在の CPU の状態を表示します。

[使用例]

```
>>>debugger.GetCpuStatus()
Stop
>>>
```

debugger.GetIeStatus

現在の IE の状態を表示します。

[指定形式]

```
debugger.GetIeStatus()
```

[引数]

なし

[戻り値]

現在の IE の状態（文字列）

IE の状態	説明
Break	ブレーク中
Coverage	カバレッジ動作中
Timer	タイマ動作中
Tracer	トレース動作中
Step	ステップ実行中
Run	ユーザ・プログラム実行中
RunOrStep	ユーザ・プログラム実行中，またはステップ実行中

注意 PM+ のワークスペースを CS+ のプロジェクトに変換した場合，メイン・プロジェクトにはデバッグ・ツールがありません。そのため，メイン・プロジェクトがアクティブ・プロジェクトである場合は，“None” を返します。
また，デバッグ・ツールと接続前の場合も，“None” を返します。

[詳細説明]

- 現在の IE の状態を表示します。

[使用例]

```
>>>debugger.GetIeStatus()
Run
>>>
```

debugger.GetIORList

IOR，SFR の一覧を表示します。

[指定形式]

```
debugger.GetIORList(category = "")
```

[引数]

引数	説明
category	IOR，SFR が定義されたカテゴリを指定します（デフォルト：指定なし）。

[戻り値]

IOR，SFR 情報のリスト（詳細は IORInfo クラスを参照してください）

[詳細説明]

- アクティブ・プロジェクトの IOR，SFR の一覧を表示します。
- *category* で定義した IOR，SFR の一覧を表示します。
- *category* を指定しない場合は，すべての IOR，SFR の一覧を表示します。
- IOR，SFR の一覧を，以下の形式で表示します。

```
IOR またはSFR 名 値 型 サイズ アドレス
```

[使用例]

```
>>> ior = debugger.GetIORList()
AD0.ADDRA 0x0000 IOR 2 0x00088040
AD0.ADDRB 0x0000 IOR 2 0x00088042
AD0.ADDRC 0x0000 IOR 2 0x00088044
        :
>>> print ior[0].IORName
AD0.ADDRA
>>> print funcinfo[0].Type
IOR
>>> print funcinfo[0].Address
557120
>>> project.GetIORList("DMA0")
DMAC0.DMCSA 0x00000000 IOR 4 0x00082000
        :
DMAC0.DMMOD.SMOD 0x0 IOR 3bits 0x8200c.12
DMAC0.DMMOD.SZSEL 0x0 IOR 3bits 0x8200c.16
```

debugger.GetPC

PC 値を表示します。

[指定形式]

```
debugger.GetPC()
```

[引数]

なし

[戻り値]

PC 値（数値）

[詳細説明]

- PC 値を表示します。

[使用例]

```
>>>debugger.GetPC()
0x92B0
```

debugger.Go

プログラムを継続して実行します。

[指定形式]

```
debugger.Go(goOption = GoOption.Normal)
```

[引数]

引数	説明	
goOption	オプションを指定します。 指定可能なオプションを以下に示します。	
	種類	説明
	GoOption.IgnoreBreak	ブレークポイントを無視した実行を行います。
	GoOption.WaitBreak	プログラムが停止するまで待機します。
	GoOption.Normal	ブレークポイントは有効で，プログラムが停止するまで待機しません（デフォルト）。

[戻り値]

なし

[詳細説明]

- プログラムを継続して実行します。
- *goOption* を指定した場合，指定した内容に従って処理を行います。

[使用例]

```
>>>debugger.Go()
>>>debugger.Go(GoOption.WaitBreak)
>>>
```

debugger.Ie.GetValue
debugger.Ie.SetValue

IE レジスタ，またはDCU レジスタを設定／参照します。

[指定形式]

```
debugger.Ie.GetValue(ieType, address)
debugger.Ie.SetValue(ieType, address, value)
```

[引数]

引数	説明	
ieType	レジスタを指定します。 指定可能なレジスタを以下に示します。	
	種類	説明
	IeType.Reg	IE レジスタ【78K0】【RL78】【78K0R】【IECUBE【V850】】【IECUBE2【V850】】
	IeType.Dcu	DCU レジスタ【IECUBE【V850】】
address	参照／設定アドレスを指定します。	
value	設定値を指定します。	

[戻り値]

debugger.Ie.GetValue はレジスタ値（数値）
debugger.Ie.SetValue は正常に設定できれば True，失敗すれば False

[詳細説明]

- debugger.Ie.GetValue は，*address* で指定したレジスタ値を表示します。
  レジスタの種類は *ieType* で指定します。
- debugger.Ie.SetValue は，*address* で指定したレジスタに *value* を書き込みます。
  レジスタの種類は *ieType* で指定します。

備考　　DCU レジスタの参照を行うと，レジスタ値は 0 にリセットされます。

[使用例]

```
>>>debugger.Ie.GetValue(IeType.Reg, 0x100)
0x12
>>>debugger.Ie.SetValue(IeType.Reg, 0x100, 0x10)
True
>>>debugger.Ie.GetValue(IeType.Reg, 0x100)
0x10
>>>
```

debugger.IsConnected

デバッグ・ツールの接続状態を確認します。

[指定形式]

```
debugger.IsConnected()
```

[引数]

なし

[戻り値]

デバッグ・ツールに接続している場合　：True
デバッグ・ツールに接続していない場合　：False

[詳細説明]

- デバッグ・ツールの接続状態を確認します。

[使用例]

```
>>>if debugger.IsConnected() == True :
...  print "OK"
...
True
OK
>>>
```

debugger.IsRunning

ユーザ・プログラムの実行状態を確認します。

[指定形式]

```
debugger.IsRunning()
```

[引数]

なし

[戻り値]

ユーザ・プログラムを実行している場合　：True
ユーザ・プログラムを実行していない場合　：False

[詳細説明]

- ユーザ・プログラムの実行状態を確認します。

[使用例]

```
>>>if debugger.IsRunning() == True :
...  print "OK"
...
True
OK
>>>
```

debugger.Jump.File
debugger.Jump.Address

各種パネルを表示します。

[指定形式]

```
debugger.Jump.File(fileName, lineNumber = 1)
debugger.Jump.Address(jumpType, adddress = 0)
```

[引数]

引数	説明	
fileName	表示するファイル名を指定します。	
lineNumber	表示する行を指定します（デフォルト：1）。	
jumpType	表示するパネルの種類を指定します。 指定可能なパネルの種類を以下に示します。	
	種類	説明
	JumpType.Source	エディタ パネル
	JumpType.Assemble	逆アセンブル パネル
	JumpType.Memory	メモリ パネル
address	表示するアドレスを指定します（デフォルト：0）。	

[戻り値]

なし

[詳細説明]

- debugger.Jump.File は，*fileName* で指定したファイルをエディタ パネルで表示します。
  lineNumber を指定した場合，*fileName* で指定したファイルの *lineNumber* で指定した行が表示されます。
- debugger.Jump.Address は，*jumpType* で指定したパネルを表示します。
  address を指定した場合，指定した *address* に該当する部分を表示します。

[使用例]

```
>>>debugger.Jump.File("C:/test/testJump.c")
>>>debugger.Jump.File("C:/test/testJump.h", 25)
>>>debugger.Jump.Address(JumpType.Memory, 0x2000)
>>>
```

debugger.Map.Clear

マッピング設定をクリアします。

[指定形式]

```
debugger.Map.Clear()
```

[引数]

なし

[戻り値]

メモリ・マップのクリアに成功した場合　：True
メモリ・マップのクリアに失敗した場合　：False

[詳細説明]

- マッピング設定をクリアします。

[使用例]

```
>>>debugger.Map.Clear()
True
>>>
```

debugger.Map.Information

マップ情報を表示します。

[指定形式]

```
debugger.Map.Information()
```

[引数]

なし

[戻り値]

マップ情報のリスト（詳細は MapInfo クラスを参照してください）

[詳細説明]

- マップ情報を，以下の形式で表示します。

番号： 開始アドレス 終アドレス アクセス・サイズ メモリ種別

[使用例]

```
>>>debugger.Map.Information()
1: 0x00000000 0x0005FFFF 32 ( 内蔵 ROM 領域 )
2: 0x00060000 0x03FF6FFF 8 ( ノン・マップ領域 )
3: 0x03FF7000 0x03FFEFFF 32 ( 内蔵 RAM 領域 )
4: 0x03FFF000 0x03FFFFFF 8 (SFR)
>>>
```

debugger.Map.Set

メモリ・マッピングの設定を行います。

[指定形式]

```
debugger.Map.Set(mapType, address1, address2, accessSize = 8, cs = "")
```

[引数]

引数	説明	
mapType	メモリ種別を指定します。 指定可能なメモリ種別を以下に示します。	
	種類	説明
	MapType.EmulationRom	エミュレーション ROM 領域
	MapType.EmulationRam	エミュレーション RAM 領域
	MapType.Target	ターゲット・メモリ領域
	MapType.TargetRom	ターゲット ROM 領域
	MapType.Stack	スタック領域
	MapType.Protect	I/O プロテクト領域
address1	マップ開始アドレスを指定します。	
address2	マップ終了アドレスを指定します。	
accessSize	アクセス・サイズ（単位：ビット）を指定します（デフォルト：8）。 V850 の場合は，8，16，32 のいずれかを指定します。 78K0R【IECUBE】の場合は，8，16 のどちらかを指定します。	
cs	チップ・セレクトを指定します（デフォルト：指定なし）。 IECUBE【V850E1】でエミュレーション・メモリ（代替 ROM/RAM) をマッピングする場合は，cs0，cs1，cs2，cs3，cs4，cs5，cs6，cs7 のいずれかのチップ・セレクトを文字列で指定します。 ただし，V850ES シリーズの品種の場合は，チップ・セレクトの割り当てが固定，またはチップ・セレクト機能がないので，省略することができます。 チップ・セレクトを指定する場合は，*accessSize* を省略することはできません。	

[戻り値]

メモリ・マッピングの設定に成功した場合　：True
メモリ・マッピングの設定に失敗した場合　：False

[詳細説明]

- *mapType* で指定したメモリ種別で，メモリ・マッピングの設定を行います。

［使用例］

```
>>>debugger.Map.Set(MapType.EmulationRom, 0x100000, 0x10ffff)
True
>>>
```

debugger.Memory.Copy

メモリをコピーします。

[指定形式]

```
debugger.Memory.Copy(address1, address2, address3)
```

[引数]

引数	説明
address1	コピー元の開始アドレスを指定します。
address2	コピー元の終了アドレスを指定します。
address3	コピー先のアドレスを指定します。

[戻り値]

メモリのコピーに成功した場合　：True
メモリのコピーに失敗した場合　：False

[詳細説明]

- *address1* から *address2* までの間を *address3* にコピーします。

[使用例]

```
>>>debugger.Memory.Copy(0x1000, 0x2000, 0x3000)
True
>>>
```

debugger.Memory.Fill

メモリを補填します。

[指定形式]

```
debugger.Memory.Fill(address1, address2, value, memoryOption = MemoryOption.Byte)
```

[引数]

引数	説明	
address1	補填開始アドレスを指定します。	
address2	補填終了アドレスを指定します。	
value	補填する値を指定します。	
memoryOption	補填する単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	MemoryOption.Byte	バイト単位（8 ビット）（デフォルト）
	MemoryOption.HalfWord	ハーフ・ワード単位（16 ビット）【RH850，RX，V850】
	MemoryOption.Word	ワード単位（RL78,78K：16 ビット，RH850，RX，V850：32 ビット）

[戻り値]

メモリの補填に成功した場合　：True
メモリの補填に失敗した場合　：False

[詳細説明]

- *address1* から *address2* までの間を *value* で補填します。
- *memoryOption* を指定した場合，指定した内容に従って補填します。

[使用例]

```
>>>debugger.Memory.Fill(0x1000, 0x2000, 0xFF)
True
>>>debugger.Memory.Fill(0x2000, 0x3000, 0x0A, MemoryOption.Word)
False
>>>
```

debugger.Memory.Read

メモリを参照します。

[指定形式]

```
debugger.Memory.Read(address, memoryOption = MemoryOption.Byte)
```

[引数]

引数	説明	
address	参照するアドレスを指定します。	
memoryOption	表示する単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	MemoryOption.Byte	バイト単位（8 ビット）（デフォルト）
	MemoryOption.HalfWord	ハーフ・ワード単位（16 ビット）【RH850，RX，V850】
	MemoryOption.Word	ワード単位（RL78,78K：16 ビット，RH850，RX，V850：32 ビット）

[戻り値]

参照したメモリ値（数値）

[詳細説明]

- *address* で指定したアドレスのメモリ値を，*memoryOption* に従って 16 進数で表示します。

[使用例]

```
>>>debugger.Memory.Read(0x100)
0x10
>>>value = debugger.Memory.Read(0x100)
0x10
>>>print value
16
>>>debugger.Memory.Read(0x100, MemoryOption.HalfWord)
0x0010
>>>
```

debugger.Memory.ReadRange

指定した個数のメモリを参照します。

[指定形式]

```
debugger.Memory.ReadRange(address, count, memoryOption = MemoryOption.Byte)
```

[引数]

引数	説明	
address	参照する開始アドレスを指定します。	
count	参照するメモリの個数を指定します。	
memoryOption	表示する単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	MemoryOption.Byte	バイト単位（8 ビット）（デフォルト）
	MemoryOption.HalfWord	ハーフ・ワード単位（16 ビット）【RH850，RX，V850】
	MemoryOption.Word	ワード単位（RL78,78K：16 ビット，RH850，RX，V850：32 ビット）

[戻り値]

参照したメモリ値（数値）のリスト
メモリの取得に失敗した場合は，None が設定されます。

[詳細説明]

- *address* で指定したアドレスから *count* で指定した個数のメモリ値を，*memoryOption* に従って 16 進数で表示します。
- メモリの取得に失敗した場合は，“?” を表示します（8 ビットの場合は 0x??，16 ビットの場合は 0x????，32 ビットの場合は 0x????????）。

[使用例]

```
>>>debugger.Memory.ReadRange(0x100, 3, MemoryOption.Word)
0x00000011 0x0000ff30 0x0000ff40
>>>mem = debugger.Memory.ReadRange(0x1ffffd, 5, MemoryOption.Byte)
0x23 0x43 0x32 0x?? 0x??
>>>print mem
[35, 67, 50, None, None]
```

debugger.Memory.Write

メモリに書き込みます。

[指定形式]

```
debugger.Memory.Write(address, value, memoryOption = MemoryOption.Byte)
```

[引数]

引数	説明	
address	設定するアドレスを指定します。	
value	設定する値を指定します。	
memoryOption	設定する単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	MemoryOption.Byte	バイト単位（8 ビット）（デフォルト）
	MemoryOption.HalfWord	ハーフ・ワード単位（16 ビット）【RH850，RX，V850】
	MemoryOption.Word	ワード単位（RL78,78K：16 ビット，RH850，RX，V850：32 ビット）

[戻り値]

メモリの書き込みに成功した場合　：True
メモリの書き込みに失敗した場合　：False

[詳細説明]

- *address* で指定したアドレスに，*memoryOption* に従って *value* を設定します。

[使用例]

```
>>>debugger.Memory.Read(0x100)
0x10
>>>debugger.Memory.Write(0x100, 0xFF)
True
>>>debugger.Memory.Read(0x100)
0xFF
>>>debugger.Memory.Write(0x100, 0xFE, MemoryOption.HalfWord)
False
>>>
```

debugger.Memory.WriteRange

複数のデータをメモリに書き込みます。

[指定形式]

```
debugger.Memory.WriteRange(address, valuelist, memoryOption = MemoryOption.Byte)
```

[引数]

引数	説明	
address	書き込む開始アドレスを指定します。	
valuelist	設定する値のリストを指定します。	
memoryOption	設定する単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	MemoryOption.Byte	バイト単位（8 ビット）（デフォルト）
	MemoryOption.HalfWord	ハーフ・ワード単位（16 ビット）【RH850，RX，V850】
	MemoryOption.Word	ワード単位（RL78,78K：16 ビット，RH850，RX，V850：32 ビット）

[戻り値]

メモリの書き込みに成功した場合　：True
メモリの書き込みに失敗した場合　：False

[詳細説明]

- *address* で指定したアドレスから，*memoryOption* に従って *valuelist* で指定した値のリストを書き込みます。

[使用例]

```
>>> mem = [0x10, 0x20, 0x30]
>>>debugger.Memory.WriteRange(0x100, mem, MemoryOption.Byte)
True
>>>debugger.Memory.ReadRange(0x100, 3, MemoryOption.Byte)
0x10 0x20 0x30
>>>debugger.Memory.WriteRange(0x100, mem, MemoryOption.Word)
True
>>>debugger.Memory.ReadRange(0x100, 3, MemoryOption.Word)
0x00000010 0x00000020 0x00000030
```

debugger.Next

プロシージャ・ステップ実行を行います。

[指定形式]

```
debugger.Next(nextOption = NextOption.Source)
```

[引数]

引数	説明	
nextOption	実行する単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	NextOption.Source	ソースの行単位（デフォルト）
	NextOption.Instruction	命令単位

[戻り値]

なし

[詳細説明]

- プロシージャ・ステップ実行を行います。
  関数呼び出しを行っている場合は，関数実行後に停止します。

[使用例]

```
>>>debugger.Next()
>>>debugger.Next(NextOption.Instruction)
>>>
```

debugger.Register.GetValue

レジスタ，I/O レジスタ，SFR を参照します。

[指定形式]

```
debugger.Register.GetValue(regName)
```

[引数]

引数	説明
regName	参照するレジスタ名を指定します。

[戻り値]

レジスタ値（数値）

[詳細説明]

- *regName* で指定したレジスタ値を表示します。

[使用例]

```
>>>debugger.Register.GetValue("pc")
0x100
>>>debugger.Register.GetValue("A:RB1")
0x20
>>>debugger.Register.SetValue("pc", 0x200)
True
>>>debugger.Register.GetValue("pc")
0x200
>>>
```

debugger.Register.SetValue

レジスタ，I/O レジスタ，SFR に値を設定します。

[指定形式]

```
debugger.Register.SetValue(regName, value)
```

[引数]

引数	説明
regName	設定するレジスタ名を指定します。
value	設定する値を指定します。

[戻り値]

値の設定に成功した場合　：True
値の設定に失敗した場合　：False

[詳細説明]

- *regName* で指定したレジスタに *value* で指定した値を設定します。

[使用例]

```
>>>debugger.Register.GetValue("pc")
0x100
>>>debugger.Register.GetValue("A:RB1")
0x20
>>>debugger.Register.SetValue("pc", 0x200)
True
>>>debugger.Register.GetValue("pc")
0x200
>>>
```

debugger.Reset

CPU をリセットします。

[指定形式]

```
debugger.Reset()
```

[引数]

なし

[戻り値]

なし

[詳細説明]

- CPU をリセットします。

[使用例]

```
>>>debugger.Reset()
>>>
```

debugger.ReturnOut

現在の関数を呼び出したプログラムに戻るまで実行します。

[指定形式]

```
debugger.ReturnOut()
```

[引数]

なし

[戻り値]

なし

[詳細説明]

- 現在の関数を呼び出したプログラムに戻るまで実行します。

[使用例]

```
>>>debugger.ReturnOut()
>>>
```

debugger.Run

プログラムをリセット後に実行します。

[指定形式]

```
debugger.Run(runOption = RunOption.Normal)
```

[引数]

引数	説明	
runOption	オプションを指定します。 指定可能なオプションを以下に示します。	
	種類	説明
	RunOption.WaitBreak	プログラムが停止するまで待機します。
	RunOption.Normal	ブレークポイントは有効で，プログラムが停止するまで待機しません（デフォルト）。

[戻り値]

なし

[詳細説明]

- プログラムをリセット後に実行します。
  runOption に RunOption.WaitBreak を指定した場合，プログラムの停止まで待機します。

[使用例]

```
>>>debugger.Run()
>>>debugger.Run(RunOption.WaitBreak)
```

debugger.Step

ステップ実行を行います。

[指定形式]

```
debugger.Step(stepOption = StepOption.Source)
```

[引数]

引数	説明	
stepOption	実行する単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	StepOption.Source	ソースの行単位（デフォルト）
	StepOption.Instruction	命令単位

[戻り値]

なし

[詳細説明]

- ステップ実行を行います。
  関数呼び出しを行っている場合は，関数の先頭で停止します。

[使用例]

```
>>>debugger.Step()
>>>debugger.Step(StepOption.Instruction)
```

debugger.Stop

デバッグ・ツールの実行を停止します。

[指定形式]

```
debugger.Stop()
```

[引数]

なし

[戻り値]

なし

[詳細説明]

- デバッグ・ツールの実行を停止します。
  プログラムを強制的に停止します。

[使用例]

```
>>>debugger.Stop()
>>>
```

debugger.Timer.Clear

条件タイマの計測結果をクリアします。

[指定形式]

```
debugger.Timer.Clear()
```

[引数]

なし

[戻り値]

条件タイマの計測結果のクリアに成功した場合　：True
条件タイマの計測結果のクリアに失敗した場合　：False

[詳細説明]

- 条件タイマの計測結果をクリアします。

[使用例]

```
>>>debugger.Timer.Get()
1 Total: 2000 ns, Pass Count: 4 , Average: 500 ns, Max: 800 ns, Min: 300 ns
>>>debugger.Timer.Clear()
True
>>>debugger.Timer.Get()
1 Total: 0 ns, Pass Count: 0 , Average: 0 ns, Max: 0 ns, Min: 0 ns
>>>
```

debugger.Timer.Delete

条件タイマを削除します。

[指定形式]

```
debugger.Timer.Delete(timerNumber = "")
```

[引数]

引数	説明
timerNumber	削除するタイマ・イベント番号を指定します。

[戻り値]

タイマの削除に成功した場合　：True
タイマの削除に失敗した場合　：False

[詳細説明]

- *timerNumber* で指定したタイマ・イベント番号のタイマを削除します。
- *timerNumber* を指定しない場合は，すべてのタイマ・イベント番号のタイマを削除します。

[使用例]

```
>>>debugger.Timer.Delete(1)
True
>>>
```

debugger.Timer.Disable

条件タイマを無効にします。

[指定形式]

```
debugger.Timer.Disable(timerNumber = "")
```

[引数]

引数	説明
timerNumber	無効にするタイマ・イベント番号を指定します。

[戻り値]

タイマの無効に成功した場合　：True
タイマの無効に失敗した場合　：False

[詳細説明]

- *timerNumber* で指定したタイマ・イベント番号のタイマを無効にします。
- *timerNumber* を指定しない場合は，すべてのタイマ・イベント番号のタイマを無効にします。

[使用例]

```
>>>debugger.Timer.Disable(1)
True
>>>
```

debugger.Timer.Enable

条件タイマを有効にします。

[指定形式]

```
debugger.Timer.Ensable(timerNumber = "")
```

[引数]

引数	説明
timerNumber	有効にするタイマ・イベント番号を指定します。

[戻り値]

タイマの有効に成功した場合　：True
タイマの有効に失敗した場合　：False

[詳細説明]

- *traceNumber* で指定したタイマ・イベント番号のタイマを有効にします。
- *traceNumber* を指定しない場合は，すべてのタイマ・イベント番号のタイマを有効にします。

[使用例]

```
>>>debugger.Timer.Enable(1)
True
>>>
```

debugger.Timer.Get

条件タイマの計測結果を参照します。

[指定形式]

```
debugger.Timer.Get()
```

[引数]

なし

[戻り値]

条件タイマ情報のリスト（詳細は TimerInfo クラスを参照してください）

[詳細説明]

- 条件タイマの計測結果を，以下の形式で表示します。

タイマ・イベント番号 Total: *総実行時間* ns, Pass Count: *パス・カウント* , Average: *平均実行時間* ns, Max: *最大実行時間* ns, Min: *最少実行時間* ns

[使用例]

```
>>>debugger.Timer.Get()
1 Total: 2000 ns, Pass Count: 4 , Average: 500 ns, Max: 800 ns, Min: 300 ns
>>>
```

debugger.Timer.Information

条件タイマ情報を表示します。

[指定形式]

```
debugger.Timer.Information()
```

[引数]

なし

[戻り値]

条件タイマ・イベント情報のリスト（詳細は TimerEventInfo クラスを参照してください）

[詳細説明]

- 条件タイマ情報を，以下の形式で表示します。

```
タイマ・イベント番号 タイマ名 状態 開始アドレス - 終了アドレス
```

[使用例]

```
>>>ti = debugger.Timer.Information()
1 Python タイマ 0001 Enable main - sub
>>>print ti[0].Number
1
>>>print ti[0].Name
Python タイマ 0001
>>>
```

debugger.Timer.Set

条件タイマを設定します。

[指定形式]

```
debugger.Timer.Set(TimerCondition)
```

[引数]

引数	説明
TimerCondition	条件タイマの条件を指定します。 条件タイマの作成については，TimerCondition クラスを参照してください。

[戻り値]

設定したタイマ・イベント番号（数値）

[詳細説明]

- *TimerCondition* で指定されている内容に従って，条件タイマを設定します。
- 設定した条件タイマは，以下の名前で登録されます。
  *数字*は 4 桁の 10 進数です。

```
Python タイマ 数字
```

[使用例]

```
>>>tc = TimerCondition()
>>>tc.StartAddress = "main"
>>>tc.EndAddress = "chData"
>>>tc.EndData = 0x20
>>>tc.EndTimerType = TimerType.Write
>>>ts_number = debugger.Timer.Set(tc)
1
>>>print ts_number
1
```

debugger.Trace.Clear

トレース・メモリをクリアします。

備考　　debugger.XTrace.Clear と同じ機能を提供します。

[指定形式]

```
debugger.Trace.Clear()
```

[引数]

なし

[戻り値]

トレース・メモリのクリアに成功した場合　：True
トレース・メモリのクリアに失敗した場合　：False

[詳細説明]

- トレース・メモリをクリアします。

[使用例]

```
>>>debugger.Trace.Clear()
False
>>>
```

debugger.Trace.Delete

条件トレースを削除します。

[指定形式]

```
debugger.Trace.Delete(traceNumber = "")
```

[引数]

引数	説明
traceNumber	削除するトレース・イベント番号を指定します。

[戻り値]

トレースの削除に成功した場合　：True
トレースの削除に失敗した場合　：False

[詳細説明]

- *traceNumber* で指定したトレース・イベント番号のトレースを削除します。
- *traceNumber* を指定しない場合は，すべてのトレース・イベント番号のトレースを削除します。

[使用例]

```
>>>debugger.Trace.Delete(1)
True
>>>
```

debugger.Trace.Disable

条件トレースを無効にします。

[指定形式]

```
debugger.Trace.Disable(traceNumber = "")
```

[引数]

引数	説明
traceNumber	無効にするトレース・イベント番号を指定します。

[戻り値]

トレースの無効に成功した場合　: True
トレースの無効に失敗した場合　: False

[詳細説明]

- *traceNumber* で指定したトレース・イベント番号のトレースを無効にします。
- *traceNumber* を指定しない場合は，すべてのトレース・イベント番号のトレースを無効にします。

[使用例]

```
>>>debugger.Trace.Disable(1)
True
>>>
```

debugger.Trace.Enable

条件トレースを有効にします。

[指定形式]

```
debugger.Trace.Enable(traceNumber = "")
```

[引数]

引数	説明
traceNumber	有効にするトレース・イベント番号を指定します。

[戻り値]

トレースの有効に成功した場合　: True
トレースの有効に失敗した場合　: False

[詳細説明]

- *traceNumber* で指定したトレース・イベント番号のトレースを有効にします。
- *traceNumber* を指定しない場合は，すべてのトレース・イベント番号のトレースを有効にします。

[使用例]

```
>>>debugger.Trace.Enable(1)
True
>>>
```

debugger.Trace.Get

トレース・データをダンプします。

備考　　debugger.XTrace.Dump と同じ機能を提供します。

[指定形式]

```
debugger.Trace.Get(frameCount, fileName = "", append = False)
```

[引数]

引数	説明
frameCount	ダンプ数を指定します。
fileName	ダンプするファイル名を指定します（デフォルト：指定なし）。
append	トレース・データをファイルに追記するかどうかを指定します。 True　：トレース・データをファイルに追記します。 False　：トレース・データをファイルに追記しません（デフォルト）。

[戻り値]

トレース情報のリスト（詳細は TraceInfo クラスを参照してください）

[詳細説明]

- *frameCount* で指定した数分のトレース・データをダンプします。
- *fileName* を指定した場合，トレース・データをファイルに書き込みます。
- *append* に “True” を指定した場合，トレース・データをファイルに追記します。

[使用例]

```
>>>debugger.Trace.Get(3)
    1851  00h00min00s003ms696µs000ns  0x000003be  cmp r11, r14
    1852  00h00min00s003ms700µs000ns  0x000003c0  blt _func_static3+0x2c
    1853  00h00min00s003ms702µs000ns  0x000003c2  jarl _errfunc, lp
>>>debugger.XTrace.Dump(10, "C:/test/TestTrace.txt")
>>>
```

debugger.Trace.Information

条件トレース情報を表示します。

[指定形式]

```
debugger.Trace.Information()
```

[引数]

なし

[戻り値]

条件トレース情報のリスト（詳細は TraceEventInfo クラスを参照してください）

[詳細説明]

- 条件トレース情報を，以下の形式で表示します。

```
トレース・イベント番号 トレース 状態 開始アドレス - 終了アドレス
```

[使用例]

```
>>>ti = debugger.Trace.Information()
1 トレース Enable main - sub
>>>print ti[0].Number
1
>>>print ti[0].Name
トレース
>>>
```

debugger.Trace.Set

条件トレースを設定します。

[指定形式]

```
debugger.Trace.Set(TraceCondition)
```

[引数]

引数	説明
TraceCondition	条件トレースの条件を指定します。 条件トレースの作成については，TraceCondition クラスを参照してください。

[戻り値]

設定したトレース・イベント番号（数値）

[詳細説明]

- *TraceCondition* で指定されている内容に従って，条件トレースを設定します。
- 設定した条件トレースは，以下の名前で登録されます。

```
トレース
```

[使用例]

```
>>>tc = TraceCondition()
>>>tc.StartAddress = "main"
>>>tc.EndAddress = "chData"
>>>tc.EndData = 0x20
>>>tc.EndTraceType = TraceType.Write
>>>ts_number = debugger.Trace.Set(tc)
1
>>>print ts_number
1
```

debugger.Upload.Binary

メモリ・データをバイナリ形式で保存します。

[指定形式]

```
debugger.Upload.Binary(fileName, address1, address2, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
address1	アップロード開始アドレスを指定します。
address2	アップロード終了アドレスを指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- *address1* から *address2* までのメモリ・データをバイナリ形式で保存します。

[使用例]

```
>>>debugger.Upload.Binary("C:/test/testBinary.bin", 0x1000, 0x2000, True)
True
>>>
```

debugger.Upload.Coverage

カバレッジ・データを保存します。【シミュレータ】

[指定形式]

```
debugger.Upload.Coverage(fileName, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- カバレッジ・データをファイルに保存します。

[使用例]

```
>>>debugger.Upload.Coverage("C:/test/coverageData.csrcv")
True
>>>
```

debugger.Upload.Intel

メモリ・データをインテル形式で保存します。

[指定形式]

```
debugger.Upload.Intel(fileName, address1, address2, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
address1	アップロード開始アドレスを指定します。
address2	アップロード終了アドレスを指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- *address1* から *address2* までのメモリ・データをインテル形式で保存します。

[使用例]

```
>>>debugger.Upload.Intel("C:/test/testIntel.hex", 0x1000, 0x2000, True)
True
>>>
```

debugger.Upload.IntelIdTag

メモリ・データを ID タグ付きインテル形式で保存します。

[指定形式]

```
debugger.Upload.IntelIdTag(fileName, address1, address2, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
address1	アップロード開始アドレスを指定します。
address2	アップロード終了アドレスを指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- *address1* から *address2* までのメモリ・データを ID タグ付きインテル形式で保存します。

[使用例]

```
>>>debugger.Upload.IntelIdTag("C:/test/testIdTagIntel.hex", 0x1000, 0x2000, True)
True
>>>
```

debugger.Upload.Motorola

メモリ・データをモトローラ形式で保存します。

[指定形式]

```
debugger.Upload.Motorola(fileName, address1, address2, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
address1	アップロード開始アドレスを指定します。
address2	アップロード終了アドレスを指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- *address1* から *address2* までのメモリ・データをモトローラ形式で保存します。

[使用例]

```
>>>debugger.Upload.Motorola("C:/test/testMotorola.hex", 0x1000, 0x2000, True)
True
>>>
```

debugger.Upload.MotorolaIdTag

メモリ・データを ID タグ付きモトローラ形式で保存します。

[指定形式]

```
debugger.Upload.MotorolaIdTag(fileName, address1, address2, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
address1	アップロード開始アドレスを指定します。
address2	アップロード終了アドレスを指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- *address1* から *address2* までのメモリ・データを ID タグ付きモトローラ形式で保存します。

[使用例]

```
>>>debugger.Upload.MotorolaIdTag("C:/test/testIdTagMotorola.hex", 0x1000, 0x2000,
True)
True
>>>
```

debugger.Upload.Tektronix

メモリ・データをテクトロニクス形式で保存します。

[指定形式]

```
debugger.Upload.Tektronix(fileName, address1, address2, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
address1	アップロード開始アドレスを指定します。
address2	アップロード終了アドレスを指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- *address1* から *address2* までのメモリ・データをテクトロニクス形式で保存します。

[使用例]

```
>>>debugger.Upload.Tektronix("C:/test/testTektronix.hex", 0x1000, 0x2000, True)
True
>>>
```

debugger.Upload.TektronixIdTag

メモリ・データを ID タグ付きテクトロニクス形式で保存します。

[指定形式]

```
debugger.Upload.TektronixIdTag(fileName, address1, address2, force = False)
```

[引数]

引数	説明
fileName	ファイル名を指定します。
address1	アップロード開始アドレスを指定します。
address2	アップロード終了アドレスを指定します。
force	上書きをするかどうかを指定します。 True ：上書きします。 False ：上書きしません（デフォルト）。

[戻り値]

アップロードに成功した場合 ：True
アップロードに失敗した場合 ：False

[詳細説明]

- *address1* から *address2* までのメモリ・データを ID タグ付きテクトロニクス形式で保存します。

[使用例]

```
>>>debugger.Upload.TektronixIdTag("C:/test/testIdTagTektronix.hex", 0x1000, 0x2000,
True)
True
>>>
```

debugger.Watch.GetValue

変数値を参照します。

[指定形式]

```
debugger.Watch.GetValue(variableName, encode = Encoding.Default, watchOption =
WatchOption.Auto)
```

[引数]

引数	説明	
variableName	参照する変数名，レジスタ名，I/O レジスタ名 /SFR レジスタ名を指定します。	
encode	文字列表示時のエンコードを指定します。 デフォルトでは，システムのエンコードを使用します。 エンコード名は，.NET の仕様に準拠します。 例）Encoding.utf-8，Encoding.euc-jp	
watchOption	オプションを指定します。 指定可能なオプションを以下に示します。	
	種類	説明
	WatchOption.Auto	自動判別して表示します（デフォルト）。
	WatchOption.Binary	2 進数で表示します。
	WatchOption.Octal	8 進数で表示します。
	WatchOption.Decimal	10 進数で表示します。
	WatchOption.SignedDecimal	符号あり 10 進数で表示します。
	WatchOption.UnsignedDecimal	符号なし 10 進数で表示します。
	WatchOption.Hexdecimal	16 進数で表示します。
	WatchOption.String	文字列で表示します。
	WatchOption.Sizeof	変数のサイズを 10 進数で表示します。
	WatchOption.Float	float 型で表示します。
	WatchOption.Double	double 型で表示します。

[戻り値]

表示した値を *watchOption* で指定した型で返します。
watchOption に “WatchOption.Auto” を指定した場合は，変数値にあわせた型で返します。
ただし，戻り値が double 型の場合は string 型で返します（*watchOption* に “WatchOption.Double” を指定した場合，および *watchOption* に “WatchOption.Auto” を指定して戻り値が double 型だった場合）。

[詳細説明]

- *variableName* で指定した変数値を表示します。
- *encode* を指定した場合，*encode* を使用してエンコードを行います。
- *watchOption* を指定した場合，*watchOption* に従って表示します。

注意 変数（*variableName*）にロード・モジュール名やファイル名を指定する場合は，ダブルクォーテーション（" "）で囲む必要がある場合があります。詳細については，「CS+ 統合開発環境 ユーザーズマニュアル デバッグ・ツール編」を参照してください。

例 ファイル名 C:¥path¥test.c，変数 var を指定する場合

```
"¥"C:/path/test.c¥"#var"
```

または

```
"¥"C:¥¥path¥¥test.c¥"#var"
```

［使用例］

```
>>>debugger.Watch.GetValue("testVal")
128
>>>debugger.Watch.GetValue("testVal", WatchOption.Hexdecimal)
0x80
>>>debugger.Watch.GetValue("testVal", WatchOption.Binary)
0b10000000
```

debugger.Watch.SetValue

変数値を設定します。

[指定形式]

```
debugger.Watch.SetValue(variableName, value)
```

[引数]

引数	説明
variableName	設定する変数名，レジスタ名，I/O レジスタ名 /SFR レジスタ名を指定します。
value	設定する値を指定します。

[戻り値]

変数値の設定に成功した場合　：True
変数値の設定に失敗した場合　：False

[詳細説明]

- *variableName* で指定した変数，レジスタ，I/O レジスタ /SFR レジスタに *value* で指定した値を設定します。

注意　変数（*variableName*）にロード・モジュール名やファイル名を指定する場合は，ダブルクォーテーション（" "）で囲む必要がある場合があります。詳細については，「CS+ 統合開発環境 ユーザーズマニュアル デバッグ・ツール編」を参照してください。

例　ファイル名 C:¥path¥test.c，変数 var を指定する場合

```
"¥"C:/path/test.c¥"#var"
```

または

```
"¥"C:¥¥path¥¥test.c¥"#var"
```

[使用例]

```
>>>debugger.Watch.GetValue("testVal")
128
>>>debugger.Watch.GetValue("testVal", WatchOption.Hexdecimal)
0x80
>>>debugger.Watch.GetValue("testVal", WatchOption.Binary)
0b10000000
>>>debugger.Watch.SetValue("testVal", 100)
True
>>>debugger.Watch.GetValue("testVal")
100
>>>debugger.Watch.GetValue("testVal", WatchOption.Hexdecimal)
0x64
>>>debugger.Watch.GetValue("testVal", WatchOption.Binary)
0b1100100
>>>debugger.Watch.SetValue("testVal", 0x256)
True
>>>debugger.Watch.GetValue("testVal", WatchOption.Hexdecimal)
0x256
```

debugger.Where

スタックのバック・トレースを表示します。

[指定形式]

```
debugger.Where()
```

[引数]

なし

[戻り値]

バック・トレースのリスト（詳細は StackInfo クラスを参照してください）

[詳細説明]

- スタックのバック・トレースを表示します。

注意 「--- Information below might be inaccurate.」を表示した場合，それ以降の表示は信用できない可能性があります。【RL78】【78K0R】

[使用例]

```
>>>debugger.Where()
1: test2.c#sub2#13
--- Information below might be inaccurate.
2:func.c#func#34
>>>
```

debugger.Whereami

ロケーションを表示します。

[指定形式]

```
debugger.Whereami(address)
```

[引数]

引数	説明
address	ロケーション表示するアドレスを指定します。

[戻り値]

ロケーションの文字列

[詳細説明]

- *address* で指定したアドレスに対するロケーションを表示します。
- 通常は，以下の形式でロケーションを表示します。

```
ファイル名 # 関数名 at ファイル名 # 行番号
```

ただし，アドレスに対する関数，または行番号が見つからない場合は，以下の形式でロケーションを表示します。

```
at シンボル名 + オフセット値
```

シンボルが見つからない場合は，以下の形式でロケーションを表示します。

```
at アドレス値
```

- *address* を省略した場合，pc 値のロケーションを表示します。

[使用例]

```
>>>debugger.Whereami()
foo.c#func at foo.c#100
>>>debugger.Whereami(0x100)
foo.c#main at foo.c#20
>>>
```

debugger.XCoverage.Clear

カバレッジ・メモリをクリアします。【IECUBE】【IECUBE2】【シミュレータ】

[指定形式]

```
debugger.XCoverage.Clear()
```

[引数]

なし

[戻り値]

カバレッジ・メモリのクリアに成功した場合　：True
カバレッジ・メモリのクリアに失敗した場合　：False

[詳細説明]

- カバレッジ・メモリをクリアします。

[使用例]

```
>>>debugger.XCoverageClear()
True
>>>
```

debugger.XCoverage.GetCoverage

カバレッジを取得します。【IECUBE】【IECUBE2】【シミュレータ】

[指定形式]

```
debugger.XCoverage.GetCoverage(funcName, progName = "", fileName = "")
```

[引数]

引数	説明
funcName	カバレッジを取得する関数名を指定します。
progName	関数が含まれているロード・モジュール名を指定します。 単数ロード・モジュールの場合は省略可能です（デフォルト）。
fileName	関数が含まれているファイル名を指定します。 グローバル関数の場合は省略可能です（デフォルト）。

注意 2 つ以上の引数を指定する場合は，3 つの引数を指定する必要があります。

[戻り値]

% を除いた値（数値）
関数の実行結果には，“%” を付けて表示します。

[詳細説明]

- *funcName* で指定した関数のカバレッジを取得します。
- 複数ロード・モジュールの場合は，*progName* を指定してください。
- スタティック関数の場合は，*fileName* を指定してください。

注意 ロード・モジュール名（*progName*）やファイル名（*fileName*）を指定する場合は，ダブルクォーテーション（" "）で囲む必要がある場合があります。詳細については，「CS+ 統合開発環境 ユーザーズマニュアル デバッグ・ツール編」を参照してください。

例 ファイル名 C:¥path¥test.c を指定する場合

```
"¥"C:/path/test.c¥""
```

または

```
"¥"C:¥¥path¥¥test.c¥""
```

[使用例]

```
>>>debugger.XCoverage.GetCoverage("TestInit", "C:/test/Test.out", "C:/test/Test.c")
81.50%
>>>
```

debugger.XRunBreak.Delete

XRunBreak 情報を削除します。【V850 シミュレータ】

[指定形式]

```
debugger.XRunBreak.Delete()
```

[引数]

なし

[戻り値]

XRunBreak 情報の削除に成功した場合 ：True
XRunBreak 情報の削除に失敗した場合 ：False

[詳細説明]

- XRunBreak 情報を削除します。

[使用例]

```
>>>debugger.XRunBreak.Refer()
None
>>>debugger.XRunBreak.Set(1, TimeType.S, True)
True
>>>debugger.XRunBreak.Refer()
1Second Periodic
>>>debugger.XRunBreak.Delete()
True
>>>debugger.XRunBreak.Refer()
None
```

debugger.XRunBreak.Refer

XRunBreak の設定情報を表示します。【V850 シミュレータ】

[指定形式]

```
debugger.XRunBreak.Refer()
```

[引数]

なし

[戻り値]

周期時間の数値と周期情報（TimeType）のリスト（詳細は XRunBreakInfo クラスを参照してください）

[詳細説明]

- 設定されている XRunBreak の周期情報（周期時間 [Periodic]）を表示します。
- XRunBreak の設定が存在しない場合は，“None” を表示します。

[使用例]

```
>>>debugger.XRunBreak.Refer()
None
>>>debugger.XRunBreak.Set(1, TimeType.S, True)
True
>>>debugger.XRunBreak.Refer()
1Second Periodic
```

debugger.XRunBreak.Set

XRunBreak 情報を設定します。【V850 シミュレータ】

[指定形式]

```
debugger.XRunBreak.Set(time, timeType = TimeType.Ms, periodic = False)
```

[引数]

引数	説明	
time	ブレーク時間を指定します。	
timeType	ブレーク時間の単位を指定します。 指定可能な単位を以下に示します。	
	種類	説明
	TimeType.Min	分単位
	TimeType.S	秒単位
	TimeType.Ms	ミリ秒単位（デフォルト）
	TimeType.Us	マイクロ秒単位
	TimeType.Ns	ナノ秒単位
periodic	指定時間毎にコールバックを呼び出すかどうかを指定します。 True ：指定時間毎に呼び出します。 False ：1 回のみ呼び出します（デフォルト）。	

[戻り値]

XRunBreak 情報の設定に成功した場合 ：True
XRunBreak 情報の設定に失敗した場合 ：False

[詳細説明]

- XRunBreak 情報を設定します。
- XRunBreak のコール間隔は，シミュレータに依存します。
- 指定時間経過後に処理する Python 関数は Hook 関数で登録します。詳細は「Hook」を参照してください。

注意 XRunBreak 情報を設定後のプログラム実行中に，
・CPU リセット
・CPU リセット後，プログラムを実行
・ブレークポイントの設定
を行う場合は，一度プラグラムを停止してから行ってください。

［使用例］

```
>>>debugger.XRunBreak.Refer()
None
>>>debugger.XRunBreak.Set(1, TimeType.S, True)
True
>>>debugger.XRunBreak.Refer()
1Second Periodic
```

debugger.XTime

Go-Break 間の時間情報を表示します。

[指定形式]

```
debugger.XTime()
```

[引数]

なし

[戻り値]

時間情報のリスト（詳細は XTimeInfo クラスを参照してください）

[詳細説明]

- Go-Break 間の時間情報を nsec 単位で表示します。

[使用例]

```
>>>debugger.XTime()
9820214200nsec
>>>
```

debugger.XTrace.Clear

トレース・メモリをクリアします。【IECUBE】【IECUBE2】【シミュレータ】

[指定形式]

```
debugger.XTrace.Clear()
```

[引数]

なし

[戻り値]

トレース・メモリのクリアに成功した場合　：True
トレース・メモリのクリアに失敗した場合　：False

[詳細説明]

- トレース・メモリをクリアします。

[使用例]

```
>>>debugger.XTrace.Clear()
False
>>>
```

debugger.XTrace.Dump

トレース・データをダンプします。【IECUBE】【IECUBE2】【シミュレータ】

[指定形式]

```
debugger.XTrace.Dump(frameCount, fileName = "", append = False)
```

[引数]

引数	説明
frameCount	ダンプ数を指定します。
fileName	ダンプするファイル名を指定します（デフォルト：指定なし）。
append	トレース・データをファイルに追記するかどうかを指定します。 True ：トレース・データをファイルに追記します。 False ：トレース・データをファイルに追記しません（デフォルト）。

[戻り値]

トレース情報のリスト（詳細は TraceInfo クラスを参照してください）

[詳細説明]

- *frameCount* で指定した数分のトレース・データをダンプします。
- *fileName* を指定した場合，トレース・データをファイルに書き込みます。
- *append* に “True” を指定した場合，トレース・データをファイルに追記します。

[使用例]

```
>>>debugger.XTrace.Dump(3)
    1851  00h00min00s003ms696μs000ns  0x000003be  cmp r11, r14
    1852  00h00min00s003ms700μs000ns  0x000003c0  blt _func_static3+0x2c
    1853  00h00min00s003ms702μs000ns  0x000003c2  jarl _errfunc, lp
>>>debugger.XTrace.Dump(10, "C:/test/TestTrace.txt")
>>>
```

B.3.6 CS+ Python クラス

以下に，CS+ Python クラスの一覧を示します。

表 B.6 CS+ Python クラス

クラス名	機能概要
ActionEventCondition	アクション・イベントの条件を作成します。
ActionEventInfo	アクション・イベント情報を保持します。
ActionInfo	アクション・イベントの結果情報を保持します。
BreakCondition	ブレーク条件を作成します。
BreakpointInfo	ブレークポイント情報を保持します。
BuildCompletedEventArgs	ビルド完了時のパラメータを保持します。
DisassembleInfo	逆アセンブル情報を保持します。
DownloadInfo	ダウンロード情報を保持します。
FunctionInfo	関数情報を保持します。
IORInfo	IOR，SFR の情報を保持します。
MapInfo	マップ情報を保持します。
StackInfo	スタック情報を保持します。
TimerCondition	条件タイマの条件を作成します。
TimerEventInfo	条件タイマ・イベント情報を保持します。
TimerInfo	条件タイマ情報を保持します。
TraceCondition	条件トレースの条件を作成します。
TraceEventInfo	条件トレース・イベント情報を保持します。
TraceInfo	トレース情報を保持します。
VariableInfo	変数情報を保持します。
XRunBreakInfo	XRunBreak 情報を保持します。
XTimeInfo	タイマ情報を保持します。

ActionEventCondition

アクション・イベントの条件を作成します。

[型]

```
class ActionEventCondition:
        Address = ""
        Output = ""
        Expression = ""
        Vector = 0
        Priority = 1
        ActionEventType = ActionEventType.Printf
```

[変数]

変数	ActionEventType の指定	説明
Address	ActionEventType.Printf	アクション・イベントのアドレスを指定します。 必ず指定してください。
	ActionEventType.Interrupt	アクション・イベントのアドレスを指定します。 必ず指定してください。
Output	ActionEventType.Printf	出力する際に付与する文字列を指定します。
	ActionEventType.Interrupt	無視されます。
Expression	ActionEventType.Printf	変数式を指定します。 カンマで区切ることにより，10 個まで指定することができます。
	ActionEventType.Interrupt	無視されます。
Vector	ActionEventType.Printf	無視されます。
	ActionEventType.Interrupt	割り込みベクタ番号を指定します。【RX シミュレータ】 0 ～ 255 の範囲で指定してください。
Priority	ActionEventType.Printf	無視されます。
	ActionEventType.Interrupt	割り込み優先順位を指定します。【RX シミュレータ】 指定可能な範囲はシリーズによって異なります。詳細については，「CS+ 統合開発環境 ユーザーズマニュアル RX デバッグ・ツール編」を参照してください。
ActionEventType	アクション・イベントの種類を指定します。 指定可能な種類を以下に示します。	
	種類	説明
	ActionEventType.Printf	Printf イベント（デフォルト）
	ActionEventType.Interrupt	割り込みイベント

[詳細説明]

- ActionEventCondition は class 形式になっており，アクション・イベントの条件を変数に指定します。
  アクション・イベントの条件を作成するには，インスタンスを生成し，生成したインスタンスに対して条件を設定します。

[使用例]

```
>>>ae = ActionEventCondition()                    …Printf イベントの場合
>>>ae.Address = 0x3000
>>>ae.Output = "chData = "
>>>ae.Expression = "chData"
>>>ae.ActionEventType = ActionEventType.Printf
>>>debugger.ActionEvent.Set(ae)
1
>>>
>>>ae = ActionEventCondition()                    … 割り込みイベントの場合
>>>ae.Address = 0x4000
>>>ae.Vector = 10
>>>ae.Priority = 2
>>>ae.ActionEventType = ActionEventType.Interrupt
>>>debugger.ActionEvent.Set(ae)
2
>>>
```

ActionEventInfo

アクション・イベント情報（debugger.ActionEvent.Information 関数の戻り値）を保持します。

［型］

```
class ActionEventInfo:
        Number = 0
        Name = ""
        Enable = True
        Address = ""
        Output = ""
        Expression = ""
        Vector = 0
        Priority = 1
        ActionEventType = ActionEventType.Printf
```

［変数］

変数	説明	
Number	アクション・イベント番号が格納されます。	
Name	アクション・イベント名が格納されます。	
Enable	アクション・イベントが有効かどうかが格納されます。 True ：有効 False ：無効	
Address	アクション・イベントのアドレスが格納されます。	
Output	出力する際に付与する文字列が格納されます。 **注意** ActionEventType が ActionEventType.Printf の場合のみ参照してください。	
Expression	変数式（文字列）が格納されます。 **注意** ActionEventType が ActionEventType.Printf の場合のみ参照してください。	
Vector	割り込みベクタ番号（数値）が格納されます。 **注意** ActionEventType が ActionEventType.Interrupt の場合のみ参照してください。	
Priority	割り込み優先順位（数値）が格納されます。 **注意** ActionEventType が ActionEventType.Interrupt の場合のみ参照してください。	
ActionEventType	アクション・イベントの種類が格納されます。	
	種類	説明
	ActionEventType.Printf	Printf イベント
	ActionEventType.Interrupt	割り込みイベント

［詳細説明］

- ActionEventInfo は class 形式になっており，debugger.ActionEvent.Information 関数を実行した場合に戻り値として渡されます。

［使用例］

```
>>>info = debugger.ActionEvent.Information()
1 Python アクションイベント 0001 Enable main - sub
>>>print info[0].Number
1
>>>print info[0].Name
Python アクション・イベント 0001
>>>print info[0].Enable
True
>>>
```

ActionInfo

アクション・イベントの結果情報（debugger.ActionEvent.Get 関数の戻り値）を保持します。

[型]

```
class ActionEventInfo:
        Number = 0
        Name = ""
        Address = ""
        Output = ""
        Expression = ""
        ActionEventType = ActionEventType.Printf
        HostDate = ""
```

[変数]

変数	説明	
Number	アクション・イベント番号（数値）が格納されます。	
Name	アクション・イベント名（文字列）が格納されます。	
Address	アクション・イベントのアドレスが格納されます。	
Output	出力する際に付与する文字列が格納されます。	
Expression	変数式（文字列）が格納されます。	
ActionEventType	アクション・イベントの種類が格納されます。	
	種類	説明
	ActionEventType.Printf	Printf イベント
HostDate	アクション・イベントが発生したホスト PC の時刻が格納されます。 ホスト PC の時刻ですので，注意が必要です。	

[詳細説明]

- ActionInfo は class 形式になっており，debugger.ActionEvent.Get 関数を実行した場合に戻り値として渡されます。

[使用例]

```
>>>ae = ActionEventCondition()
>>>ae.Address = "main"
>>>ae.Output = "result "
>>>ae.Expression = "chData"
>>>ae.ActionEventType = ActionEventType.Printf
>>>ae_number = debugger.ActionEvent.Set(ae)
        :
>>>out = debugger.ActionEvent.Get()
result chData=0x64
result chData=0x65
result chData=0x66
>>>print out[0].Address
main
```

BreakCondition

ブレーク条件を作成します。

[型]

```
class BreakCondition:
        Address = ""
        Data = None
        AccessSize = None
        BreakType = BreakType.Hardware
```

[変数]

変数	説明	
Address	ブレークを設定するアドレスを指定します。 必ず指定してください。	
Data	データのブレーク条件を設定する数値を指定します。 “None” を指定した場合，データ条件は無視されます。	
AccessSize	アクセス・サイズ（8，16，32 のいずれか）を指定します。 “None” を指定した場合，すべてのアクセス・サイズを指定したことになります。	
BreakType	ブレークの種類を指定します。 指定可能な種類を以下に示します。	
	種類	説明
	BreakType.Software	ソフトウエア・ブレーク（シミュレータ以外）
	BreakType.Hardware	ハードウエア・ブレーク（デフォルト）
	BreakType.Read	データ・リード・ブレーク
	BreakType.Write	データ・ライト・ブレーク
	BreakType.Access	データ・アクセス・ブレーク

[詳細説明]

- BreakCondition は class 形式になっており，ブレーク条件を変数に指定します。
  ブレーク条件を作成するには，インスタンスを生成し，生成したインスタンスに対して条件を設定します。

［使用例］

```
>>>executeBreak = BreakCondition()              … インスタンスを生成
>>>executeBreak.Address = "main"
>>>executeBreak.BreakType = BreakType.Software
>>>debugger.Breakpoint.Set(executeBreak)        … ブレークポイント設定関数の引数に指定
>>>
>>>dataBreak = BreakCondition()                 … インスタンスを生成
>>>dataBreak.Address = "chData"
>>>dataBreak.Data = 0x10
>>>dataBreak.BreakType = BreakType.Access
>>>debugger.Breakpoint.Set(dataBreak)           … ブレークポイント設定関数の引数に指定
>>>
>>>executeBreak.Address = "sub + 0x10"          … ブレーク条件を再利用
>>>debugger.Breakpoint.Set(executeBreak)        … ブレークポイント設定関数の引数に指定
>>>
```

BreakpointInfo

ブレークポイント情報（debugger.Breakpoint.Information 関数の戻り値）を保持します。

[型]

```
class BreakpointInfo:
        Number = 0
        Name = None
        Enable = True
        BreakType = BreakType.Hardware
        Address1 = None
        Address2 = None
        Address3 = None
        Address4 = None
```

[変数]

変数	説明	
Number	イベント番号が格納されます。	
Name	ブレークポイント名が格納されます。	
Enable	ブレークポイントが有効かどうかが格納されます。 True ：有効 False ：無効	
BreakType	ブレークの種類が格納されます。	
	種類	説明
	BreakType.Software	ソフトウエア・ブレーク（シミュレータ以外）
	BreakType.Hardware	ハードウエア・ブレーク
	BreakType.Read	データ・リード・ブレーク
	BreakType.Write	データ・ライト・ブレーク
	BreakType.Access	データ・アクセス・ブレーク
Address1	アドレス情報 1 が文字列として格納されます。	
Address2	アドレス情報 2 が文字列として格納されます（組み合わせブレーク時のみ）。	
Address3	アドレス情報 3 が文字列として格納されます（組み合わせブレーク時のみ）。	
Address4	アドレス情報 4 が文字列として格納されます（組み合わせブレーク時のみ）。	

[詳細説明]

- BreakpointInfo は class 形式になっており，debugger.Breakpoint.Information 関数を実行した場合に戻り値として渡されます。

［使用例］

```
>>>info = debugger.Breakpoint.Information()
   1 ブレーク 0001 Enable  test1.c#_main+2
   2 ブレーク 0002 Disable test2.c#_sub4+10
>>>print info[0].Number
1
>>>print info[0].Name
ブレーク 0001
>>>print info[0].BreakType
Hardware
>>>print info[0].Enable
True
>>>print info[0].Address1
test1.c#_main+2
>>>print info[0].Address2
None
>>>print info[1].Number
2
>>>print info[1].Name
ブレーク 0002
>>>print info[1].BreakType
Hardware
>>>print info[1].Enable
False
>>>print info[1].Address1
test2.c#_sub4+10
>>>print info[1].Address2
None
>>>
```

BuildCompletedEventArgs

ビルド完了時のパラメータを保持します。

[型]

```
class BuildCompletedEventArgs:
        Error = None
        Cancelled = False
        HasBuildError = False
        HasBuildWarning = False
```

[変数]

変数	説明
Error	ビルドで例外が発生した場合，エラーの内容（System.Exception）が格納されます。
Cancelled	ビルドの実行がキャンセルされたかどうかが格納されます。
HasBuildError	ビルドでエラーが発生したかどうかが格納されます。
HasBuildWarning	ビルドでワーニングが発生したかどうかが格納されます。

[詳細説明]

- BreakCompletedEventArgs は class 形式になっており，build.BuildCompleted イベントが発生した場合のみ引数として渡されます。
  そのため，この class のインスタンスを生成することはできません。

[使用例]

```
>>>def buildCompleted(sender, e):
... print "Error = {0}".format(e.Error)
... print "BuildError = " + e.HasBuildError.ToString()
... print "BuildWarning = " + e.HasBuildWarning.ToString()
... print "BuildCancelled = " + e.Cancelled.ToString()
...
>>>build.BuildCompleted += buildCompleted    … イベントの接続
>>>build.All(True)
Error = None
BuildError = False
BuildWarning = False
BuildCancelled = False
True
>>>                                        … 例外が発生した場合，下記のように表示されます
>>>build.All(True)
Error = System.Exception: ビルド中にエラーが発生しました。(E0203001)
BuildError = False
BuildWarning = False
BuildCancelled = False
False
>>>
>>>                                        … ビルド・エラーが発生した場合，下記のように表示されます
>>>build.All(True)
Error = None
```

```
BuildError = True
BuildWarning = False
BuildCancelled = False
False
>>>
```

DisassembleInfo

逆アセンブル情報（debugger.Assemble.Disassemble 関数の戻り値）を保持します。

[型]

```
class DisassembleInfo:
        Address = 0
        Code = None
        Mnemonic = None
```

[変数]

変数	説明
Address	アドレスが格納されます。
Code	コード情報がバイト単位のコレクションとして格納されます。
Mnemonic	ニーモニック情報が格納されます。

[詳細説明]

- DisassembleInfo は class 形式になっており，debugger.Assemble.Disassemble 関数の戻り値の構造です。

[使用例]

```
>>>info = debugger.Assemble.Disassemble("main", 4)      … 逆アセンブルの実行
0x000002DC      B51D      br _main+0x36
0x000002DE      0132      mov0x1, r6
0x000002E0      60FF3800  jarl _func_static1, lp
0x000002E4      63570100  st.w r10, 0x0[sp]
>>>print info[0].Address
732
>>>print info[0].Code[0]
181
>>>print info[0].Code[1]
29
>>>print Mnemonic
br _main+0x36
>>>print info[3].Address
740
>>>print info[3].Code[0]
99
>>>print info[3].Code[1]
87
>>>print info[3].Code[2]
1
>>>print info[3].Code[3]
0
>>>print info[3].Mnemonic
st.w r10, 0x0[sp]
>>>
```

DownloadInfo

ダウンロード情報（debugger.Download.Information 関数の戻り値）を保持します。

[型]

```
class DownloadInfo:
        Number = None
        Name = None
        ObjectDownload = True
        SymbolDownload = False
```

[変数]

変数	説明
Number	ダウンロード番号が格納されます。
Name	ファイル名が格納されます。
ObjectDownload	オブジェクト情報をダウンロードしているかどうかが格納されます。 True ：オブジェクト情報をダウンロードしている False ：オブジェクト情報をダウンロードしていない
SymbolDownload	シンボル情報をダウンロードしているかどうかが格納されます。 True ：シンボル情報をダウンロードしている False ：シンボル情報をダウンロードしていない

[詳細説明]

- DownloadInfo は class 形式になっており，debugger.Download.Information 関数の戻り値の構造です。

[使用例]

```
>>>info = debugger.Download.Information()
   1: DefaultBuild¥sample.out
>>>print info[0].Number
1
>>>print info[0].Name
DefaultBuild¥sample.out
>>>print info[0].ObjectDownload
True
>>>print info[0].SymbolDownload
True
>>>
```

FunctionInfo

関数情報（project.GetFunctionList 関数の戻り値）を保持します。

[型]

```
class FunctionInfo:
        FunctionName = None
        FileName = None
        ReturnType = None
        StartAddress = None
        EndAddress = None
```

[変数]

変数	説明
FunctionName	関数名が格納されます。
FileName	関数が定義されているファイル名がフルパスでが格納されます。
ReturnType	戻り値の型が格納されます。
StartAddress	関数の開始アドレスが格納されます。
EndAddress	関数の終了アドレスが格納されます。

[詳細説明]

- FunctionInfo は class 形式になっており，project.GetFunctionList 関数の戻り値の構造です。

[使用例]

```
>>>info = project.GetFunctionList()
func1 int 0x00200 0x00224 C:¥project¥src¥test1.c
func2 int 0x00225 0x002ff C:¥project¥src¥test2.c
>>>print info[0].FunctionName
func1
>>>print info[1].FileName
C:¥project¥src¥test2.c
>>>print info[0].StartAddress
512
>>>
```

IORInfo

IOR，SFR の情報（debugger.GetIORList 関数の戻り値）を保持します。

[型]

```
class IORInfo:
        IORName = ""
        Value = ""
        Type = ""
        Size = ""
        Address = ""
        Category = ""
```

[変数]

変数	説明
IORName	IOR，SFR の名前が格納されます。
Value	値が格納されます。
Type	型が格納されます。
Size	サイズが格納されます。 サイズがバイト単位の場合はバイト数，ビット単位の場合はビット数（bits）が格納されます。
Address	アドレスが格納されます。
Category	カテゴリが格納されます。

[詳細説明]

- IORInfo は class 形式になっており，debugger.GetIORList 関数を実行した場合に戻り値として渡されます。

[使用例]

```
>>> ior = project.GetIORList()
AD0.ADDRA 0x0000 IOR 2 0x00088040
AD0.ADDRB 0x0000 IOR 2 0x00088042
AD0.ADDRC 0x0000 IOR 2 0x00088044
        :
>>> print ior[0].IORName
AD0.ADDRA
>>> print funcinfo[0].Type
IOR
>>> print funcinfo[0].Address
557120
```

MapInfo

マップ情報（debugger.Map.Information 関数の戻り値）を保持します。

[型]

```
class MapInfo:
        Number = 0
        StartAddress = 0
        EndAddress = 0
        AccessSize = 0
        MapTypeName = None
```

[変数]

変数	説明
Number	番号が格納されます。
StartAddress	マップ領域の開始アドレスが格納されます。
EndAddress	マップ領域の終了アドレスが格納されます。
AccessSize	マップ領域のアクセス・サイズが格納されます。
MapTypeName	マップ領域の型名が格納されます。

[詳細説明]

- MapInfo は class 形式になっており，debugger.Map.Information 関数の戻り値の構造です。

[使用例]

```
>>>info = debugger.Map.Information()    …Map.Information 関数の実行
   1: 0x00000000 0x0003FFFF 32 ( 内蔵 ROM 領域 )
   2: 0x00040000 0x00048FFF  8 ( ノン・マップ領域 )
   3: 0x00049000 0x001003FF  8 ( エミュレーション ROM 領域 )
   4: 0x00100400 0x03FF8FFF  8 ( ノン・マップ領域 )
   5: 0x03FF9000 0x03FFEFFF 32 ( 内蔵 RAM 領域 )
   6: 0x03FFF000 0x03FFFFFF  8 (I/O レジスタ領域 )
>>>print info[0].StartAddress
0
>>>print info[0].EndAddress
262143
>>>print info[0].AccessSize
32
>>>print info[0].MapTypeName
内蔵 ROM 領域
>>>print info[5].StartAddress
67104768
>>>print info[5].EndAddress
67108863
>>>print info[5].AccessSize
8
>>>print info[5].MapTypeName
I/O レジスタ領域
>>>
```

StackInfo

スタック情報（debugger.Where 関数の戻り値）を保持します。

[型]

```
class StackInfo:
        Number = None
        AddressInfoText = None
```

[変数]

変数	説明
Number	スタック番号が格納されます。
AddressInfoText	スタックのアドレス情報が文字列で格納されます。

[詳細説明]

- StackInfo は class 形式になっており，debugger.Where 関数の戻り値の構造です。

[使用例]

```
>>>info = debugger.Where()
   1: test2.c#
   2: test1.c#main#41
>>>print info[0].Number
1
>>>print info[0].AddressInfoText
test2.c#
>>>info = debugger.Where
1: test2.c#
--- Information below might be inaccurate.
2: test1.c#main#41
>>>print a[1].Number
None
>>>print a[1].AddressInfoText
--- Information below might be inaccurate.
>>>
```

TimerCondition

条件タイマの条件を作成します。

[型]

```
class TimerCondition:
        StartAddress = ""
        StartData = ""
        StartTimerType = TimerType.Execution
        EndAddress = ""
        EndData = ""
        EndTimerType = TimerType.Execution
```

[変数]

変数	説明	
StartAddress	タイマ測定を開始するアドレスを指定します。 必ず指定してください。	
StartData	タイマ測定を開始するアドレスのデータ条件（数値）を指定します。 StartTimerType に “TimerType.Execution” を指定した場合，本指定は無視されます。	
StartTimerType	タイマ測定を開始するタイマの種類を指定します。 指定可能な種類を以下に示します。	
	種類	説明
	TimerType.Execution	実行時にタイマの開始を行います（デフォルト）。
	TimerType.Read	データ・リード時にタイマの開始を行います。
	TimerType.Write	データ・ライト時にタイマの開始を行います。
	TimerType.Access	データ・アクセス時にタイマの開始を行います。
EndAddress	タイマ測定を終了するアドレスを指定します。 必ず指定してください。	
EndData	タイマ測定を終了するアドレスのデータ条件（数値）を指定します。 EndTimerType に “TimerType.Execution” を指定した場合，本指定は無視されます。	
EndTimerType	タイマ測定を終了するタイマの種類を指定します。 指定可能な種類を以下に示します。	
	種類	説明
	TimerType.Execution	実行時にタイマの終了を行います（デフォルト）。
	TimerType.Read	データ・リード時にタイマの終了を行います。
	TimerType.Write	データ・ライト時にタイマの終了を行います。
	TimerType.Access	データ・アクセス時にタイマの終了を行います。

[詳細説明]

- TimerCondition は class 形式になっており，条件タイマの条件を変数に指定します。
  条件タイマの条件を作成するには，インスタンスを生成し，生成したインスタンスに対して条件を設定します。

［使用例］

```
>>>execute_timer = TimerCondition()              … インスタンスを生成
>>>execute_timer.StartAddress = "main"
>>>execute_timer.StartTimerType = TimerType.Execution
>>>execute_timer.EndAddress = "sub"
>>>execute_timer.EndTimerType = TimerType.Execution
>>>debugger.Timer.Set(execute_timer)             … 条件タイマ設定関数の引数に指定
1
>>>
```

TimerEventInfo

条件タイマ・イベント情報（debugger.Timer.Information 関数の戻り値）を保持します。

［型］

```
class TimerEventInfo:
        Number = 0
        Name = ""
        Enable = True
        StartAddress = ""
        StartData = ""
        StartTimerType = TimerType.Execution
        EndAddress = ""
        EndData = ""
        EndTimerType = TimerType.Execution
```

［変数］

変数	説明	
Number	タイマ・イベント番号が格納されます。	
Name	タイマ名が格納されます。	
Enable	タイマが有効かどうかが格納されます。 True ：有効 False ：無効	
StartAddress	タイマ測定を開始するアドレスが格納されます。	
StartData	タイマ測定を開始するアドレスのデータ条件（数値）が格納されます。	
StartTimerType	タイマ測定を開始するタイマの種類が格納されます。	
	種類	説明
	TimerType.Execution	実行時にタイマの開始を行います。
	TimerType.Read	データ・リード時にタイマの開始を行います。
	TimerType.Write	データ・ライト時にタイマの開始を行います。
	TimerType.Access	データ・アクセス時にタイマの開始を行います。
EndAddress	タイマ測定を終了するアドレスが格納されます。	
EndData	タイマ測定を終了するアドレスのデータ条件（数値）が格納されます。	
EndTimerType	タイマ測定を終了するタイマの種類が格納されます。	
	種類	説明
	TimerType.Execution	実行時にタイマの終了を行います。
	TimerType.Read	データ・リード時にタイマの終了を行います。
	TimerType.Write	データ・ライト時にタイマの終了を行います。
	TimerType.Access	データ・アクセス時にタイマの終了を行います。

[詳細説明]

- TimerEventInfo は class 形式になっており，debugger.Timer.Information 関数を実行した場合に戻り値として渡されます。

[使用例]

```
>>>info = debugger.Timer.Information()
1 Python タイマ 0001 Enable main - sub
>>>print info[0].Number
1
>>>print info[0].Name
Python タイマ 0001
>>>print info[0].Enable
True
>>>
```

TimerInfo

条件タイマ情報（debugger.Timer.Get 関数の戻り値）を保持します。

［型］

```
class TimerInfo:
        Number = 0
        MaxTime = 0
        MaxClockCount = 0
        IsMaxOverflow = False
        MinTime = 0
        MinClockCount = 0
        IsMinOverflow = False
        AverageTime = 0
        AverageClockCount = 0
        IsAverageOverflow = False
        TotalTime = 0
        TotalClockCount = 0
        IsTotalOverflow = False
        PassCount = 0
        IsPassCountOverflow = False
```

［変数］

変数	説明
Number	タイマ・イベント番号が格納されます。
MaxTime	最大実行時間が格納されます。
MaxClockCount	最大実行クロック数が格納されます。
IsMaxOverflow	最大実行時間／クロック数がオーバフローしたかどうかが格納されます。 True ：最大実行時間／クロック数がオーバフローした False ：最大実行時間／クロック数がオーバフローしていない
MinTime	最少実行時間が格納されます。
MinClockCount	最少実行クロック数が格納されます。
IsMinOverflow	最少実行時間／クロック数がオーバフローしたかどうかが格納されます。 True ：最少実行時間／クロック数がオーバフローした False ：最少実行時間／クロック数がオーバフローしていない
AverageTime	平均実行時間が格納されます。
AverageClockCount	平均実行クロック数が格納されます。
IsAverageOverflow	平均実行時間／クロック数がオーバフローしたかどうかが格納されます。 True ：平均実行時間／クロック数がオーバフローした False ：平均実行時間／クロック数がオーバフローしていない
TotalTime	総実行時間が格納されます。
TotalClockCount	総実行クロック数が格納されます。
IsTotalOverflow	総実行時間／クロック数がオーバフローしたかどうかが格納されます。 True ：総実行時間／クロック数がオーバフローした False ：総実行時間／クロック数がオーバフローしていない
PassCount	パス・カウントが格納されます。

変数	説明
IsPassCountOverflow	パス・カウントがオーバフローしたかどうかが格納されます。 True ：パス・カウントがオーバフローした False ：パス・カウントがオーバフローしていない

[詳細説明]

- TimerInfo は class 形式になっており，debugger.Timer.Get 関数を実行した場合に戻り値として渡されます。

[使用例]

```
>>>info = debugger.Timer.Get()
1 Total: 2000 ns, Pass Count: 4 , Average: 500 ns, Max: 800 ns, Min: 300 ns
>>>print info[0].Number
1
>>>print info[0].MaxTime
800
>>>print info[0].PassCount
4
>>>print info[0].IsMaxOverflow
False
>>>
```

TraceCondition

条件トレースの条件を作成します。

[型]

```
class TraceCondition:
        StartAddress = ""
        StartData = ""
        StartTraceType = TraceType.Execution
        EndAddress = ""
        EndData = ""
        EndTraceType = TraceType.Execution
```

[変数]

変数	説明	
StartAddress	トレースを開始するアドレスを指定します。 必ず指定してください。	
StartData	トレースを開始するアドレスのデータ条件（数値）を指定します。 StartTraceType に “TraceType.Execution” を指定した場合，本指定は無視されます。	
StartTraceType	トレースを開始するトレースの種類を指定します。 指定可能な種類を以下に示します。	
	種類	説明
	TraceType.Execution	実行時にトレースの開始を行います（デフォルト）。
	TraceType.Read	データ・リード時にトレースの開始を行います。
	TraceType.Write	データ・ライト時にトレースの開始を行います。
	TraceType.Access	データ・アクセス時にトレースの開始を行います。
EndAddress	トレースを終了するアドレスを指定します。 必ず指定してください。	
EndData	トレースを終了するアドレスのデータ条件（数値）を指定します。 EndTraceType に “TraceTypeType.Execution” を指定した場合，本指定は無視されます。	
EndTraceType	トレースを終了するトレースの種類を指定します。 指定可能な種類を以下に示します。	
	種類	説明
	TraceType.Execution	実行時にトレースの終了を行います（デフォルト）。
	TraceType.Read	データ・リード時にトレースの終了を行います。
	TraceType.Write	データ・ライト時にトレースの終了を行います。
	TraceType.Access	データ・アクセス時にトレースの終了を行います。

[詳細説明]

- TraceCondition は class 形式になっており，条件トレースの条件を変数に指定します。
  条件トレースの条件を作成するには，インスタンスを生成し，生成したインスタンスに対して条件を設定します。

［使用例］

```
>>>execute_trace = TraceCondition()              … インスタンスを生成
>>>execute_trace.StartAddress = "main"
>>>execute_trace.StartTraceType = TraceType.Execution
>>>execute_trace.EndAddress = "sub"
>>>execute_trace.EndTraceType = TraceType.Execution
>>>debugger.Trace.Set(execute_trace)             … 条件トレース設定関数の引数に指定
1
>>>
```

TraceEventInfo

条件トレース・イベント情報（debugger.Trace.Information 関数の戻り値）を保持します。

［型］

```
class TraceEventInfo:
        Number = 0
        Name = ""
        Enable = True
        StartAddress = ""
        StartData = ""
        StartTraceType = TraceType.Execution
        EndAddress = ""
        EndData = ""
        EndTraceType = TraceType.Execution
```

［変数］

<table>
<tr><th>変数</th><th colspan="2">説明</th></tr>
<tr><td>Number</td><td colspan="2">トレース・イベント番号が格納されます。</td></tr>
<tr><td>Name</td><td colspan="2">トレース名が格納されます。</td></tr>
<tr><td>Enable</td><td colspan="2">トレースが有効かどうかが格納されます。
True ：有効
False ：無効</td></tr>
<tr><td>StartAddress</td><td colspan="2">トレースを開始するアドレスが格納されます。</td></tr>
<tr><td>StartData</td><td colspan="2">トレースを開始するアドレスのデータ条件（数値）が格納されます。</td></tr>
<tr><td rowspan="5">StartTraceType</td><td colspan="2">トレースを開始するトレースの種類が格納されます。</td></tr>
<tr><td>種類</td><td>説明</td></tr>
<tr><td>TraceType.Execution</td><td>実行時にトレースの開始を行います。</td></tr>
<tr><td>TraceType.Read</td><td>データ・リード時にトレースの開始を行います。</td></tr>
<tr><td>TraceType.Write</td><td>データ・ライト時にトレースの開始を行います。</td></tr>
<tr><td>EndAddress</td><td>TraceType.Access</td><td>データ・アクセス時にトレースの開始を行います。</td></tr>
<tr><td>トレースを終了するアドレスが格納されます。</td><td colspan="2"></td></tr>
<tr><td>EndData</td><td colspan="2">トレースを終了するアドレスのデータ条件（数値）が格納されます。</td></tr>
<tr><td rowspan="5">EndTraceType</td><td colspan="2">トレースを終了するトレースの種類が格納されます。</td></tr>
<tr><td>種類</td><td>説明</td></tr>
<tr><td>TraceType.Execution</td><td>実行時にトレースの終了を行います。</td></tr>
<tr><td>TraceType.Read</td><td>データ・リード時にトレースの終了を行います。</td></tr>
<tr><td>TraceType.Write</td><td>データ・ライト時にトレースの終了を行います。</td></tr>
<tr><td>TraceType.Access</td><td>データ・アクセス時にトレースの終了を行います。</td></tr>
</table>

[詳細説明]

- TraceEventInfo は class 形式になっており，debugger.Trace.Information 関数を実行した場合に戻り値として渡されます。

[使用例]

```
>>>info = debugger.Trace.Information()
1 トレース Enable main - sub
>>>print info[0].Number
1
>>>print info[0].Name
トレース
>>>print info[0].Enable
True
>>>
```

TraceInfo

トレース情報（debugger.XTrace.Dump 関数の戻り値）を保持します。

［型］

```
class TraceInfo:
        FrameNumber = None
        Timestamp = None
        FetchAddress = None
        Mnemonic = None
        ReadAddress = None
        ReadData = None
        WriteAddress = None
        WriteData = None
        VectorAddress = None
        VectorData = None
        IsDma = True
```

［変数］

変数	説明
FrameNumber	フレーム番号情報が格納されます。
Timestamp	タイムスタンプ情報が格納されます。
FetchAddress	フェッチ・アドレス情報が格納されます。
Mnemonic	ニーモニック情報が格納されます。
ReadAddress	リード・アドレス情報が格納されます。
ReadData	リード・データ情報が格納されます。
WriteAddress	ライト・アドレス情報が格納されます。
WriteData	ライト・データ情報が格納されます。
VectorAddress	ベクタ・アドレス情報が格納されます。
VectorData	ベクタ・データが格納されます。
IsDma	データが DMA かどうかを格納します。 True ：データが DMA False ：データが DMA 以外

［詳細説明］

- TraceInfo は class 形式になっており，debugger.XTrace.Dump 関数の戻り値の構造です。

［使用例］

```
>>>info = debugger.XTrace.Dump(10)
    853   00h00min00s001ms704us000ns  0x000002c2 movhi 0xffff, gp, r1
    854   00h00min00s001ms706us000ns  0x000002c6 id.w 0x7ff4[r1], r6
    855   00h00min00s001ms706us000ns                                0x03ff9000 R
0x00000000
    856   00h00min00s001ms706us000ns  0x000002ca movhi 0xffff, gp, r1
    857   00h00min00s001ms710us000ns  0x000002ce movea 0x7ff8, r1, r7
    858   00h00min00s001ms712us000ns  0x000002d2 jarl _main+0x36
    859   00h00min00s001ms716us000ns  0x000002dc br _main+0x36
    860   00h00min00s001ms720us000ns  0x00000312 prepare lp, 0x4
    861   00h00min00s001ms720us000ns                                0x03ff9308 W
0x000002d6
    862   00h00min00s001ms724us000ns  0x00000316 br _main+0x2
>>>print info[0].FrameNumber
853
>>>print info[0].Timestamp
1704000
>>>print info[0].FetchAddress
706
>>>print info[0].Mnemonic
movhi 0xffff, gp, r1
>>>print info[0].ReadAddress
None
>>>print info[0].ReadData
None
>>>print info[0].IsDma
False
>>>
>>>print info[2].FrameNumber
855
>>> print info[2].Timestamp
1706000
>>>print info[2].FetchAddress
None
>>>print info[2].Mnemonic
None
>>>print info[2].ReadAddress
67080192
```

VariableInfo

変数情報（project.GetVariableList 関数の戻り値）を保持します。

[型]

```
class VariableInfo:
        VariableName = None
        FileName = None
        Attribute = None
        Type = None
        Address = None
        Size = None
```

[変数]

変数	説明
VariableName	変数名が格納されます。
FileName	変数が定義されているファイル名がフルパスでが格納されます。
Attribute	属性が格納されます。
Type	型が格納されます。
Address	アドレスが格納されます。
Size	サイズが格納されます。

[詳細説明]

- VariableInfo は class 形式になっており，project.GetVariableList 関数の戻り値の構造です。

[使用例]

```
>>>info = project.GetVariableList()
var1 volatile int 0x000014e4 4 C:¥project¥src¥test1.c
var2 static int 0x000014e8 4 C:¥project¥src¥test2.c
>>>print info[0].VariableName
var1
>>>print info[1].FileName
C:¥project¥src¥test2.c
>>>print info[0].Attribute
volatile
>>>print info[0].Type
int
>>>
```

XRunBreakInfo

XRunBreak 情報（debugger.XRunBreak.Refer 関数の戻り値）を保持します。

[型]

```
class XRunBreakInfo:
        Value = 0
        TimeType = Timetype.Min
        IsPeriodic = True
```

[変数]

変数	説明	
Value	イベントの発生間隔値が格納されます。	
ATimeType	発生間隔値の単位が格納されます。	
	種類	説明
	TimeType.Min	分単位
	TimeType.S	秒単位
	TimeType.Ms	ミリ秒単位
	TimeType.Us	マイクロ秒単位
	TimeType.Ns	ナノ秒単位
IsPeriodic	指定時間毎にコールバックされるかどうかが格納されます。	

[詳細説明]

- XRunBreakInfo は class 形式になっており，debugger.XRunBreak.Refer 関数の戻り値の構造です。

[使用例]

```
>>>debugger.XRunBreak.Set(10, TimeType.S, True)
>>>info = debugger.XRunBreak.Refer()
10Second Periodic
>>>print info.Value
10
>>>print info.TimeType
S
>>>print info.IsPeriodic
True
>>>
```

XTimeInfo

タイマ情報（debugger.XTime 関数の戻り値）を保持します。

[型]

```
class XTimeInfo:
        Value = 0
        IsCpuClock = False
        IsOverFlow = False
```

[変数]

変数	説明
Value	タイマの計測値が格納されます。
IsCpuClock	CPU クロックの計測かどうかが格納されます。 True ：CPU クロックの計測 False ：上記以外
IsOverFlow	オーバーフローが発生したかどうかが格納されます。 True ：オーバーフローが発生した False ：オーバーフローが発生していない

[詳細説明]

- XTimeInfo は class 形式になっており，debugger.XTime 関数の戻り値の構造です。

[使用例]

```
>>>info = debugger.XTime()
9820214200nsec
>>>print info.Value
9820214200
>>>print info.IsCpuClock
False
>>>print info.IsOverFlow
False
>>>
```

B.3.7 CS+ Python プロパティ（共通）

以下に，CS+ Python プロパティ（共通）の一覧を示します。

表 B.7 CS+ Python プロパティ（共通）

プロパティ名	機能概要
common.ExecutePath	実行している CS+ の exe ファイルのフォルダの絶対パスを参照します。
common.ConsoleClear	アクティブ・プロジェクト変更時に Python コンソールの表示をクリアする／しないを設定／参照します。
common.EnableRemotingStartup	CS+ の起動時に外部ツールと連携する機能を有効にする／しないを設定／参照します。
common.Output	CS+ 用 Python 関数の戻り値，またはエラー内容を参照します。
common.ThrowExcept	Python 関数の実行時に例外を発生させる／させないを設定／参照します。
common.UseRemoting	Python コンソールの起動中に外部ツールと連携する機能を有効にする／しないを設定／参照します。
common.Version	CS+ のバージョンを参照します。
common.ViewLine	Python コンソールの表示桁数を設定／参照します。
common.ViewOutput	CS+ 用 Python 関数の実行結果，またはエラー内容を Python コンソールに表示する／しないを設定／参照します。

common.ExecutePath

実行している CS+ の exe ファイルのフォルダの絶対パスを参照します。

[指定形式]

```
common.ExecutePath
```

[設定]

なし

[参照]

実行している CS+ の exe ファイルのフォルダの絶対パス

[詳細説明]

- 実行している CS+ の exe ファイル（CubeSuiteW+.exe，または CubeSuite+.exe）のフォルダの絶対パスを参照します。

[使用例]

```
>>>print common.ExecutePath
C:¥Program Files¥Renesas Electronics¥CS+¥CC
```

common.ConsoleClear

アクティブ・プロジェクト変更時に Python コンソールの表示をクリアする／しないを設定／参照します。

[指定形式]

```
common.ConsoleClear = bool
```

[設定]

設定	説明
bool	アクティブ・プロジェクト変更時に Python コンソールの表示をクリアするかどうかを設定します。 True ：Python コンソールの表示をクリアします（デフォルト）。 False ：Python コンソールの表示をクリアしません。

[参照]

現在の設定値

[詳細説明]

- アクティブ・プロジェクト変更時に Python コンソールの表示をクリアする／しないを設定／参照します。

[使用例]

```
>>>print common.ConsoleClear
True
>>>common.ConsoleClear = False
```

common.EnableRemotingStartup

CS+ の起動時に外部ツールと連携する機能を有効にする／しないを設定／参照します。

[指定形式]

```
common.EnableRemotingStartup = bool
```

[設定]

設定	説明
bool	CS+ の起動時に外部ツールと連携する機能を有効にするかどうかを設定します。 True ：外部ツールと連携する機能を有効にします（デフォルト）。 False ：外部ツールと連携する機能を無効にします。 起動中に外部ツールと連携する機能を有効／無効にする場合は common.UseRemoting プロパティを使用してください。

[参照]

現在の設定値

[詳細説明]

- CS+ の起動時に外部ツールと連携する機能を有効にする／しないを設定／参照します。

[使用例]

```
>>>print common.EnableRemotingStartup
False
>>>common.EnableRemotingStartup = True
```

common.Output

CS+ 用 Python 関数の実行結果，またはエラー内容を参照します。

[指定形式]

```
common.Output
```

[設定]

なし

[参照]

CS+ 用 Python 関数の実行結果，またはエラー・メッセージ（文字列）

注意 エラー・メッセージを参照できるのは，common.ThrowExcept プロパティで例外を発生させない（False）設定をしている場合のみです。

備考 次の CS+ 用 Python 関数が実行されるまで，参照内容を保持します。

[詳細説明]

- CS+ 用 Python 関数の実行結果，またはエラー内容を参照します。

[使用例]

```
>>>debugger.Memory.Read("data")
0x0
>>>print common.Output
0
```

common.ThrowExcept

Python 関数の実行時に例外を発生させる／させないを設定／参照します。

[指定形式]

```
common.ThrowExcept = bool
```

[設定]

設定	説明
bool	Python 関数の実行時に例外を発生させるかどうかを設定します。 True ：例外を発生させます。 False ：例外を発生させません（デフォルト）。

[参照]

現在の設定値

[詳細説明]

- Python 関数の実行時に例外を発生させる／させないを設定／参照します。
- try ～ except を使用したい場合は，*bool* に “True” を設定します。

[使用例]

```
>>>print common.ThrowExcept
False
>>>common.ThrowExcept = True
```

common.UseRemoting

CS+ の起動中に外部ツールと連携する機能を有効にする／しないを設定／参照します。

[指定形式]

```
common.UseRemoting = bool
```

[設定]

設定	説明
bool	CS+ の起動中に外部ツールと連携する機能を有効にするかどうかを設定します。 True ：外部ツールと連携する機能を有効にします（デフォルト）。 False ：外部ツールと連携する機能を無効にします。 起動時に common.EnableRemotingStartup プロパティが True の場合は True, False の場合は False が設定されます。

[参照]

現在の設定値

[詳細説明]

- CS+ の起動中に外部ツールと連携する機能を有効にする／しないを設定／参照します。

[使用例]

```
>>>print common.UseRemoting
False
>>>common.UseRemoting = True
```

common.Version

CS+ のバージョンを参照します。

[指定形式]

```
common.Version
```

[設定]

なし

[参照]

CS+ のバージョン

[詳細説明]

- CS+ のバージョンを参照します。

[使用例]

```
>>>print common.Version
V1.02.00  [01 Apr 2012]
```

common.ViewLine

Python コンソールの表示桁数を設定／参照します。

[指定形式]

```
common.ViewLine = number
```

[設定]

設定	説明
number	Python コンソールの表示桁数を数値で設定します（デフォルト：10000）。

[参照]

現在の設定値

[詳細説明]

- Python コンソールの表示桁数を設定／参照します。

[使用例]

```
>>>print common.ViewLine
10000
>>>common.ViewLine = 20000
```

common.ViewOutput

CS+ 用 Python 関数の実行結果，またはエラー内容を Python コンソールに表示する／しないを設定／参照します。

[指定形式]

```
common.ViewOutput = bool
```

[設定]

設定	説明
bool	CS+ 用 Python 関数の実行結果，またはエラー内容を Python コンソールに表示するかどうかを設定します。 True ：Python コンソールに表示します（デフォルト）。 False ：Python コンソールに表示しません。

[参照]

現在の設定値

[詳細説明]

- CS+ 用 Python 関数の実行結果，またはエラー内容を Python コンソールに表示する／しないを設定／参照します。

[使用例]

```
>>>print common.ViewOutput
False
>>>common.ViewOutput = True
```

B.3.8 CS+ Python プロパティ（プロジェクト用）

以下に，CS+ Python プロパティ（プロジェクト用）の一覧を示します。

表 B.8 CS+ Python プロパティ（プロジェクト用）

プロパティ名	機能概要
project.Device	アクティブ・プロジェクトのマイクロコントローラ名を参照します。
project.IsOpen	プロジェクトを読み込んでいるかどうかを確認します。
project.Kind	アクティブ・プロジェクトの種類を参照します。
project.Name	アクティブ・プロジェクト・ファイル名（パスなし）を参照します。
project.Nickname	アクティブ・プロジェクトのマイクロコントローラ名の愛称を参照します。
project.Path	アクティブ・プロジェクト・ファイル名（パス付き）を参照します。

project.Device

アクティブ・プロジェクトのマイクロコントローラ名を参照します。

[指定形式]

```
project.Device
```

[設定]

なし

[参照]

アクティブ・プロジェクトのマイクロコントローラ名

[詳細説明]

- アクティブ・プロジェクトのマイクロコントローラ名を参照します。

[使用例]

```
>>>print project.Device
R5F100LE
```

project.IsOpen

プロジェクトを読み込んでいるかどうかを確認します。

[指定形式]

```
project.IsOpen
```

[設定]

なし

[参照]

プロジェクトを読み込んでいる場合　：True
プロジェクトを読み込んでいない場合　：False

[詳細説明]

- プロジェクトが開いているかどうかを確認します。

[使用例]

```
>>>print project.IsOpen
True
>>>
```

project.Kind

アクティブ・プロジェクトの種類を参照します。

[指定形式]

```
project.Kind
```

[設定]

なし

[参照]

アクティブ・プロジェクトの種類

種類	説明
Application	アプリケーション用プロジェクト
Library	ライブラリ用プロジェクト
DebugOnly	デバッグ専用プロジェクト
Empty	空のアプリケーション用プロジェクト
CppApplication	C++ アプリケーション用プロジェクト
RI600V4	RI600V4 用プロジェクト
RI600PX	RI600PX 用プロジェクト
RI850V4	RI850V4 用プロジェクト
RI850MP	RI850MP 用プロジェクト
RI78V4	RI78V4 用プロジェクト
MulticoreBootLoader	マルチコア用ブート・ローダ・プロジェクト
MulticoreApplication	マルチコア用アプリケーション・プロジェクト

[詳細説明]

- アクティブ・プロジェクトの種類を参照します。

[使用例]

```
>>>print project.Kind
Application
>>>
```

project.Name

アクティブ・プロジェクト・ファイル名（パスなし）を参照します。

[指定形式]

```
project.Name
```

[設定]

なし

[参照]

アクティブ・プロジェクト・ファイル名（パスなし）

[詳細説明]

- アクティブ・プロジェクト・ファイル名（パスなし）を参照します。

[使用例]

```
>>>print project.Name
test.mtpj
```

project.Nickname

アクティブ・プロジェクトのマイクロコントローラ名の愛称を参照します。

[指定形式]

```
project.Nickname
```

[設定]

なし

[参照]

アクティブ・プロジェクトのマイクロコントローラ名の愛称

[詳細説明]

- アクティブ・プロジェクトのマイクロコントローラ名の愛称を参照します。

[使用例]

```
>>>print project.Nickname
RL78/G13 (ROM:64KB)
```

project.Path

アクティブ・プロジェクト・ファイル名（パス付き）を参照します。

[指定形式]

```
project.Path
```

[設定]

なし

[参照]

アクティブ・プロジェクト・ファイル名（パス付き）

[詳細説明]

- アクティブ・プロジェクト・ファイル名（パス付き）を参照します。

[使用例]

```
>>>print project.Path
C:¥project¥test.mtpj
```

B.3.9 CS+ Python プロパティ（ビルド・ツール用）

以下に，CS+ Python プロパティ（ビルド・ツール用）の一覧を示します。

表 B.9 CS+ Python プロパティ（ビルド・ツール用）

プロパティ名	機能概要
build.Compile.IncludePath	アクティブ・プロジェクトのコンパイル・オプションである，追加のインクルード・パスの設定／参照を行います。
build.Compile.Macro	アクティブ・プロジェクトのコンパイル・オプションである，定義マクロの設定／参照を行います。
build.IsBuilding	ビルド実行中かどうかを確認します。
build.Link.LibraryFile	アクティブ・プロジェクトのライブラリ・ファイルの設定／参照を行います。
build.Link.SectionAlignment	アクティブ・プロジェクトのリンク・オプションである，セクション・アライメントの設定／参照を行います。
build.Link.SectionROMtoRAM	アクティブ・プロジェクトのリンク・オプションである，ROM から RAM へマップするセクションの設定／参照を行います。
build.Link.SectionStartAddress	アクティブ・プロジェクトのリンク・オプションである，セクションの開始アドレスの設定／参照を行います。
build.Link.SectionSymbolFile	アクティブ・プロジェクトのリンク・オプションである，外部定義シンボルをファイル出力するセクションの設定／参照を行います。
build.ROMization.OutputObjectFile	アクティブ・プロジェクトの ROM 化プロセス・オプションである，ROM 化用オブジェクト・ファイルの出力の設定／参照を行います。
build.Version	コンパイラ・パッケージのバージョンを参照します。

build.Compile.IncludePath

アクティブ・プロジェクトのコンパイル・オプションである，追加のインクルード・パスの設定／参照を行います。

[指定形式]

```
build.Compile.IncludePath = dirlist
```

[設定]

設定	説明
dirlist	追加するインクルード・パスを文字列のリストで設定します。

[参照]

追加のインクルード・パスのリスト

[詳細説明]

- アクティブ・プロジェクトのコンパイル・オプションである，追加のインクルード・パスの設定／参照を行います。
- 設定を変更する場合は，参照したリストに対して追加／変更してください。

[使用例]

```
>>>incpath1 = build.Compile.IncludePath      … 現在の設定を参照してインクルード・パスを追加
>>>print incpath1
['include', 'C:¥project¥inc']
>>>incpath1.append('include2')
>>>build.Compile.IncludePath = incpath1
>>>print build.Compile.IncludePath
['include', 'C:¥project¥inc', 'include2']
>>>
>>>incpath2 = ['include1', 'include2']       … 複数のインクルード・パスを設定
>>>build.Compile.IncludePath = incpath2
>>>print build.Compile.IncludePath
['include1', 'include2']
```

build.Compile.Macro

アクティブ・プロジェクトのコンパイル・オプションである，定義マクロの設定／参照を行います。

[指定形式]

```
build.Compile.Macro = macrolist
```

[設定]

設定	説明
macrolist	定義マクロを文字列のリストで設定します。

[参照]

定義マクロのリスト

[詳細説明]

- アクティブ・プロジェクトのコンパイル・オプションである，定義マクロの設定／参照を行います。
- 設定を変更する場合は，参照したリストに対して追加／変更してください。

[使用例]

```
>>>macrolist = build.Compile.Macro          … 現在の設定を参照して定義マクロを追加
>>>print macrolist
['RL78']
>>>macrolist.append('78K')
>>>build.Compile.Macro = macrolist
>>>print build.Compile.Macro
['RL78', '78K']
>>>
>>>macrolist = ['macro1', 'macro2']         … 複数の定義マクロを設定
>>>build.Compile.Macro = macrolist
>>>print build.Compile.Macro
['macro1', 'macro2']
```

build.IsBuilding

ビルド実行中かどうかを確認します。

[指定形式]

```
build.IsBuilding
```

[設定]

なし

[参照]

ビルド実行中の場合　　：True
ビルドを行っていない場合　：False

[詳細説明]

- ビルド実行中かどうかを確認します。

[使用例]

```
>>>print build.IsBuilding
False
>>>
```

build.Link.LibraryFile

アクティブ・プロジェクトのライブラリ・ファイルの設定／参照を行います。

[指定形式]

```
build.Link.LibraryFile = filelist
```

[設定]

設定	説明
filelist	アクティブ・プロジェクトのライブラリ・ファイルを文字列のリストで設定します。

[参照]

ライブラリ・ファイルのリスト

[詳細説明]

- アクティブ・プロジェクトのライブラリ・ファイルの設定／参照を行います。
- 設定を変更する場合は，参照したリストに対して追加／変更してください。

[使用例]

```
>>>lib1 = build.Link.LibraryFile            … 現在の設定を参照してライブラリ・ファイルを追加
>>>print lib1
['test1.lib', 'test2.lib']
>>>lib1.append("test3.lib")
>>>build.Link.LibraryFile = lib1
>>>print build.Link.LibraryFile
['test1.lib', 'test2.lib', 'test3.lib']
>>>
>>>lib2 = ['test1.lib', 'test2.lib']        … 複数のライブラリ・ファイルを設定
>>>build.Link.LibraryFile = lib2
>>>print build.Link.LibraryFile
['test1.lib', 'test2.lib']
```

build.Link.SectionAlignment

アクティブ・プロジェクトのリンク・オプションである，セクション・アライメントの設定／参照を行います。【CC-RH】【CC-RX】【CC-RL】

[指定形式]

```
build.Link.SectionAlignment = sectionlist
```

[設定]

設定	説明
sectionlist	セクション・アライメントを文字列のリストで設定します。

[参照]

セクション・アライメントのリスト

[詳細説明]

- アクティブ・プロジェクトのリンク・オプションである，セクション・アライメントの設定／参照を行います。
- 設定を変更する場合は，参照したリストに対して追加／変更してください。

[使用例]

```
>>>sec1= build.Link.SectionAlignment     … 現在の設定を参照してセクション・アライメントを追加
>>>print sec1
['R_1']
>>>sec1.append('R_2')
>>>build.Link.SectionAlignment = sec1
>>>print build.Link.SectionAlignment
['R_1', 'R_2']
>>>
>>>sec2 = ['R_1', 'R_2']                 … 複数のセクション・アライメントを設定
>>>build.Link.SectionAlignment = sec2
>>>print build.Link.SectionAlignment
['R_1', 'R_2']
```

build.Link.SectionROMtoRAM

アクティブ・プロジェクトのリンク・オプションである，ROM から RAM へマップするセクションの設定／参照を行います。【CC-RH】【CC-RX】【CC-RL】

[指定形式]

```
build.Link.SectionROMtoRAM = sectionlist
```

[設定]

設定	説明
sectionlist	ROM から RAM へマップするセクションを文字列のリストで設定します。

[参照]

ROM から RAM へマップするセクションのリスト

[詳細説明]

- アクティブ・プロジェクトのリンク・オプションである，ROM から RAM へマップするセクションの設定／参照を行います。
- 設定を変更する場合は，参照したリストに対して追加／変更してください。

[使用例]

```
>>>sec = build.Link.SectionROMtoRAM      … 現在の設定を参照して ROM から RAM へマップするセクションを追加
>>>print sec
['D=R', 'D_1=R_1', 'D_2=R_2']
>>>sec.append('D_3=R_3')
>>>build.Link.SectionROMtoRAM = sec
>>>print build.Link.SectionROMtoRAM
['D=R', 'D_1=R_1', 'D_2=R_2', 'D_3=R_3']
```

build.Link.SectionStartAddress

アクティブ・プロジェクトのリンク・オプションである，セクションの開始アドレスの設定／参照を行います。【CC-RH】【CC-RX】【CC-RL】

[指定形式]

```
build.Link.SectionStartAddress = section
```

[設定]

設定	説明
section	セクションの開始アドレスを文字列で設定します。

[参照]

セクションの開始アドレス（文字列）

[詳細説明]

- アクティブ・プロジェクトのリンク・オプションである，セクションの開始アドレスの設定／参照を行います。
- 設定を変更する場合は，参照した文字列に対して追加／変更してください。

[使用例]

```
>>>sec= build.Link.SectionStartAddress  …現在の設定を参照してセクションの開始アドレスを変更
>>>print sec
B_1,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/0FFFF8000
>>>sec = "B_1/0200,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/0FFFF8000"
>>>build.Link.SectionStartAddress = sec
>>>print build.Link.SectionStartAddress
B_1/0200,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/0FFFF8000
```

build.Link.SectionSymbolFile

アクティブ・プロジェクトのリンク・オプションである，外部定義シンボルをファイル出力するセクションの設定／参照を行います。【CC-RH】【CC-RX】【CC-RL】

[指定形式]

```
build.Link.SectionSymbolFile = sectionlist
```

[設定]

設定	説明
sectionlist	外部定義シンボルをファイル出力するセクションを文字列のリストで設定します。

[参照]

外部定義シンボルをファイル出力するセクションのリスト

[詳細説明]

- アクティブ・プロジェクトのリンク・オプションである，外部定義シンボルをファイル出力するセクションの設定／参照を行います。
- 設定を変更する場合は，参照したリストに対して追加／変更してください。

[使用例]

```
>>>sec = build.Link.SectionSymbolFile    … 現在の設定を参照して外部定義シンボルをファイル出力するセクションを追加
>>>print sec
['R_1', 'R_2']
>>>sec.append('R_3')
>>>build.Link.SectionSymbolFile = sec
>>>print build.Link.SectionSymbolFile
['R_1', 'R_2', 'R_3']
```

build.ROMization.OutputObjectFile

アクティブ・プロジェクトのROM化プロセス・オプションである，ROM化用オブジェクト・ファイルの出力の設定／参照を行います。【CA850】【CX】【CA78K0R】

[指定形式]

```
build.ROMization.OutputObjectFile = bool
```

[設定]

設定	説明
bool	ROM化用オブジェクト・ファイルを出力するかどうかを設定します。 True ：ROM化用オブジェクト・ファイルを出力します。 False ：ROM化用オブジェクト・ファイルを出力しません。

[参照]

ROM化用オブジェクト・ファイルを出力する場合 ： True
ROM化用オブジェクト・ファイルを出力しない場合 ： False
サポートしていないコンパイラの場合 ： None

[詳細説明]

- アクティブ・プロジェクトのROM化プロセス・オプションである，ROM化用オブジェクト・ファイルの出力の設定／参照を行います。

[使用例]

```
>>>setting = build.ROMization.OutputObjectFile
>>>print setting
True
>>>build.ROMization.OutputObjectFile = False
>>>print build.ROMization.OutputObjectFile
False
```

build.Version

コンパイラ・パッケージのバージョンを参照します。

[指定形式]

```
build.Version
```

[設定]

なし

[参照]

アクティブ・プロジェクトで使用しているコンパイラ・パッケージのバージョン

[詳細説明]

- アクティブ・プロジェクトで使用しているコンパイラ・パッケージのバージョンを参照します。

[使用例]

```
>>>print build.Version
V2.00.00
```

B.3.10 CS+ Python プロパティ（デバッグ・ツール用）

以下に，CS+ Python プロパティ（デバッグ・ツール用）の一覧を示します。

表 B.10 CS+ Python プロパティ（デバッグ・ツール用）

プロパティ名	機能概要
debugger.ActionEvent.GetLine	アクション・イベント結果を保持する数を設定／参照します。
debugger.ADConvertDataInExecution	実行時のデータ収集の設定／参照を行います。
debugger.IsMulticore	アクティブ・プロジェクトのマイクロコントローラがマルチコアかどうかを確認します。
debugger.Memory.NoVerify	書き込み時のベリファイ設定を切り替えます。
debugger.Option.AccessStopExecution debugger.Option.AfterTraceMemoryFull debugger.Opiton.Coverage debugger.Option.OpenBreak debugger.Option.ReuseCoverageData debugger.Option.Timer debugger.Option.Trace debugger.Option.UseTraceData	デバッグ・ツールのオプションを設定／参照します。
debugger.ProcessorElement	マルチコアの PE を設定／参照します。
debugger.XTrace.Addup debugger.XTrace.Complement debugger.XTrace.Mode	トレース・オプションを設定／参照します。

debugger.ActionEvent.GetLine

アクション・イベント結果を保持する数を設定／参照します。

[指定形式]

```
debugger.ActionEvent.GetLine = number
```

[設定]

設定	説明
number	アクション・イベント結果を Python コンソール内で保持する数を設定します（デフォルト：10000）。

[参照]

現在の設定値

[詳細説明]

- アクション・イベント結果を Python コンソール内で保持する数を設定／参照します。
- 設定した数を超える場合は，アクション・イベントの結果を保持しません。古いアクション・イベントの結果から削除します。有効範囲は，5000 ～ 100000 です。

[使用例]

```
>>>print debugger.ActionEvent.GetLine
10000
>>>debugger.ActionEvent.GetLine = 50000
>>>print debugger.ActionEvent.GetLine
50000
```

debugger.ADConvertDataInExecution

実行時のデータ収集の設定／参照を行います。【Smart Analog】

[指定形式]

```
debugger.ADConvertDataInExecution = adConvertDataInExecution
```

[設定]

設定	説明
adConvertDataInExecution	実行中にデータ収集するかどうかを設定します。 True ：実行中にデータ収集します。 False ：実行中にデータ収集しません。

[参照]

現在実行中のデータ収集の設定

[詳細説明]

- 実行時のデータ収集の設定／参照を行います。

[使用例]

```
>>>print debugeer.ADConvertDataInExecution
False
>>>debugger.ADConvertDataInExecution = True
>>>print debugger.ADConvertDataInExecution
True
>>>
```

debugger.IsMulticore

アクティブ・プロジェクトのマイクロコントローラがマルチコアかどうかを確認します。

[指定形式]

```
debugger.IsMulticore
```

[設定]

なし

[参照]

マルチコアの場合　　：True
マルチコアでない場合　：False

[詳細説明]

- プロジェクトのマイクロコントローラがマルチコアかどうかを確認します。

[使用例]

```
>>>print debugger.IsMulticore
False
>>>
```

debugger.Memory.NoVerify

書き込み時のベリファイ設定を切り替えます。【シミュレータ以外】

[指定形式]

```
debugger.Memory.NoVerify = noverify
```

[設定]

設定	説明
noverify	書き込み時にベリファイするかどうかを設定します。 True ：書き込み時にベリファイします。 False ：書き込み時にベリファイしません。

[参照]

設定されている値

注意 PM+ のワークスペースを CS+ のプロジェクトに変換した場合，メイン・プロジェクトにはデバッグ・ツールがありません。そのため，メイン・プロジェクトがアクティブ・プロジェクトである場合は，“None” を返します。

[詳細説明]

- 書き込み時のベリファイ設定を切り替えます。

[使用例]

```
>>>print debugger.Memory.NoVerify
False
>>>debugger. Memory.NoVerify = True
>>>print debugger. Memory.NoVerify
True
>>>
```

debugger.Option.AccessStopExecution
debugger.Option.AfterTraceMemoryFull
debugger.Opiton.Coverage
debugger.Option.OpenBreak
debugger.Option.ReuseCoverageData
debugger.Option.Timer
debugger.Option.Trace
debugger.Option.UseTraceData

デバッグ・ツールのオプションを設定／参照します。

［指定形式］

```
debugger.Option.AccessStopExecution = afterTrace
debugger.Option.AfterTraceMemoryFull = accessStopExecution
debugger.Opiton.Coverage = coverage
debugger.Option.OpenBreak = openBreak
debugger.Option.ReuseCoverageData = reuseCoverageData
debugger.Option.Timer = timer
debugger.Option.Trace = trace
debugger.Option.UseTraceData = useTraceDataType
```

［設定］

<table>
<tr><th>設定</th><th colspan="2">説明</th></tr>
<tr><td rowspan="5">afterTrace</td><td colspan="2">トレース・メモリを使い切ったあとの動作を設定します。
指定可能な値を以下に示します。</td></tr>
<tr><td>値</td><td>説明</td></tr>
<tr><td>AfterTraceMemoryFull.NoneStop</td><td>トレース・メモリを上書きして実行を続ける</td></tr>
<tr><td>AfterTraceMemoryFull.StopTrace</td><td>トレースを停止する</td></tr>
<tr><td>AfterTraceMemoryFull.Stop</td><td>停止する（プログラムを停止する）</td></tr>
<tr><td>accessStopExecution</td><td colspan="2">実行を一瞬停止してアクセスするかどうかを設定します。
True ：実行を一瞬停止してアクセスします。
False ：実行を一瞬停止してアクセスしません。</td></tr>
<tr><td>coverage</td><td colspan="2">カバレッジ機能を使用するかどうかを設定します。【IECUBE】【IECUBE2】【シミュレータ】
True ：カバレッジ機能を使用します。
False ：カバレッジ機能を使用しません。</td></tr>
<tr><td>openBreak</td><td colspan="2">オープン・ブレーク機能を使用するかどうかを設定します。
True ：オープン・ブレーク機能を使用します。
False ：オープン・ブレーク機能を使用しません。</td></tr>
<tr><td>reuseCoverageData</td><td colspan="2">カバレッジ結果を再利用するかどうかを設定します。
True ：カバレッジ結果を再利用します。
False ：カバレッジ結果を再利用しません。</td></tr>
<tr><td>timer</td><td colspan="2">タイマ機能を使用するかどうかを設定します。
True ：タイマ機能を使用します。
False ：タイマ機能を使用しません。</td></tr>
</table>

<table>
<tr><th>設定</th><th colspan="2">説明</th></tr>
<tr><td>trace</td><td colspan="2">トレース機能を使用するかどうかを設定します。【IECUBE】【IECUBE2】【シミュレータ】
True ：トレース機能を使用します。
False ：トレース機能を使用しません。</td></tr>
<tr><td>useTraceDataType</td><td colspan="2">トレース・データをどの機能で使用するかを設定します。【IECUBE【V850】】【IECUBE2】
指定可能な機能を以下に示します。</td></tr>
<tr><td></td><td>種類</td><td>説明</td></tr>
<tr><td></td><td>UseTraceDataType.RRM</td><td>RRM 機能</td></tr>
<tr><td></td><td>UseTraceDataType.Trace</td><td>トレース機能</td></tr>
<tr><td></td><td>UseTraceDataType.Coverage</td><td>カバレッジ機能</td></tr>
</table>

[参照]

設定されている値

注意 PM+ のワークスペースを CS+ のプロジェクトに変換した場合，メイン・プロジェクトにはデバッグ・ツールがありません。そのため，メイン・プロジェクトがアクティブ・プロジェクトである場合は，“None” を返します。

[詳細説明]

- デバッグ・ツールのオプションを設定／参照します。

[使用例]

```
>>>print debugger.Option.UseTraceData
Trace
>>>debugger.Option.UseTraceData = UseTraceDataType.Coverage
>>>print debugger.Option.Coverage
False
>>>debugger.Option.Coverage = True
>>>print debugger.Option.Coverage
True
>>>
```

debugger.ProcessorElement

マルチコアの PE を設定／参照します。【RH850】

[指定形式]

```
debugger.ProcessorElement = number
```

[設定]

設定	説明
number	PE 番号を数値で設定します。

[参照]

現在の設定値

[詳細説明]

- マルチコアの PE を設定／参照します。

注意 設定する場合は，デバッグ・ツールと接続されている必要があります。

[使用例]

```
>>>print debugger.ProcessorElement
1
>>>debugger.ProcessorElement = 2
>>>print debugger.ProcessorElement
2
>>>
```

debugger.XTrace.Addup
debugger.XTrace.Complement
debugger.XTrace.Mode

トレース・オプションを設定／参照します。【IECUBE】【IECUBE2】【シミュレータ】

[指定形式]

```
debugger.XTrace.Addup = addup【シミュレータ】
debugger.XTrace.Complement = complement【IECUBE【V850】】【IECUBE2【V850】】
debugger.XTrace.Mode = traceMode【シミュレータ】【IECUBE】【IECUBE2】
```

[設定]

<table>
<tr><th>設定</th><th colspan="2">説明</th></tr>
<tr><td>addup</td><td colspan="2">タイム・タグを積算するかどうかを設定します。
True ：タイム・タグを積算します。
False ：タイム・タグを積算しません。</td></tr>
<tr><td>complement</td><td colspan="2">トレースを補完するかどうかを設定します。
True ：トレースを補完します。
False ：トレースを補完しません。</td></tr>
<tr><td rowspan="4">traceMode</td><td colspan="2">トレース制御モードを設定します。
指定可能なトレース制御モードを以下に示します。</td></tr>
<tr><td>種類</td><td>説明</td></tr>
<tr><td>TraceMode.FullBreak</td><td>トレース・メモリを使い切ったら，プログラムの実行とトレース・データの書き込みを停止します。</td></tr>
<tr><td>TraceMode.FullStop</td><td>トレース・メモリを使い切ったら，トレース・データの書き込みを停止します。</td></tr>
<tr><td>TraceMode.NonStop</td><td>トレース・メモリを使い切っても，トレース・データの上書きを続けます。</td></tr>
</table>

[参照]

設定されている値

注意 PM+ のワークスペースを CS+ のプロジェクトに変換した場合，メイン・プロジェクトにはデバッグ・ツールがありません。そのため，メイン・プロジェクトがアクティブ・プロジェクトである場合は，“None” を返します。

[詳細説明]

- トレース・オプションを設定／参照します。

［使用例］

```
>>>print debugger.XTrace.Addup
False
>>>debugger.XTrace.Addup = True
>>>print debugger.XTrace.Addup
True
>>>
```

B.3.11 CS+ Python イベント

以下に，CS+ Python イベントの一覧を示します。

表 B.11 CS+ Python イベント

イベント名	機能概要
build.BuildCompleted	ビルド実行が完了したことを通知します。

build.BuildCompleted

ビルド実行が完了したことを通知します。

[ハンドラ形式]

```
build.BuildCompleted(sender, e)
```

[ハンドラ引数]

引数	説明
sender	ビルド・イベントの発行元が渡されます。
e	ビルド実行完了時のパラメータが渡されます。

[戻り値]

なし

[詳細説明]

- ビルド実行が完了したことを通知します。

[使用例]

```
>>>def buildCompleted(sender, e):
... print "Error = {0}".format(e.Error)
... print "BuildError = " + e.HasBuildError.ToString()
... print "BuildWarning = " + e.HasBuildWarning.ToString()
... print "BuildCancelled = " + e.Cancelled.ToString()
...
>>>build.BuildCompleted += buildCompleted       … イベントの接続
>>>build.All(True)
Error = None
BuildError = False
BuildWarning = False
BuildCancelled = False
True
>>>
>>>build.File("C:/sample/src/test1.c")
Error = None
BuildError = False
BuildWarning = False
BuildCancelled = False
True
>>>
>>>
>>>build.Clean()
Error = None
BuildError = False
BuildWarning = False
BuildCancelled = False
True
>>>
```

B.4 Python コンソールの注意事項

(1) 日本語入力に関する注意事項
Python コンソールでは日本語入力機能を有効にすることができません。日本語を入力する場合は，外部テキスト・エディタなどで作成し，コピーして貼り付けてください。

(2) プロンプト表示に関する注意事項
Python コンソールのプロンプトが >>> であるところが >>>>>> というように複数表示される場合や，>>> の後に結果が表示され，キャレットの前に >>> がない場合があります。このような状態でも継続して関数を入力することが可能です。

(3) ロード・モジュールがないプロジェクトのスクリプト実行に関する注意事項
ロード・モジュール・ファイルがないプロジェクトを使用して起動オプションでスクリプト指定した場合，または *プロジェクト・ファイル名* .py をプロジェクト・ファイルと同じフォルダに置いている場合，通常プロジェクト読み込み後に自動的にスクリプトを実行しますが，ロード・モジュール・ファイルがない場合は実行しません。

(4) 強制終了に関する注意事項
無限ループしているようなスクリプトを実行中に以下の操作を行うと，強制的に関数の実行を終了させるため，関数の実行結果がエラーになる場合があります。

- Python コンソールのコンテキスト・メニューの［強制終了］や Ctrl ＋ D で強制終了
- 複数のプロジェクトを持つプロジェクトでアクティブ・プロジェクトを変更

改訂記録

Rev.	発行日	改定内容	
		ページ	ポイント
1.00	2014.08.01	-	初版発行

CS+ V3.00.00 ユーザーズマニュアル
Pythonコンソール編

発行年月日　2014年 8月 1日　Rev.1.00
発行　ルネサス エレクトロニクス株式会社
〒211-8668 神奈川県川崎市中原区下沼部 1753

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒100-0004 千代田区大手町2-6-2（日本ビル）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

Colophon 2.0

CS+ V3.00.00

RENESAS

ルネサス エレクトロニクス株式会社

R20UT3097JJ0100