User's Manual

CS+ V8.12.00

Integrated Development Environment

User's Manual: Python Console

Target Device

RL78 Family

RX Family

RH850 Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.00 2024.06

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

© 2024 Renesas Electronics Corporation. All rights reserved.

How to Use This Manual

This manual describes the role of the CS+ integrated development environment for developing applications and systems for RH850 family, RX family, and RL78 family, and provides an outline of its features.

CS+ is an integrated development environment (IDE) for RH850 family, RX family, and RL78 family, integrating the necessary tools for the development phase of software (e.g. design, implementation, and debugging) into a single platform.

By providing an integrated environment, it is possible to perform all development using just this product, without the need to use many different tools separately.

Readers	This manual is intended for users who wish to understand the functions of the CS+ and design software and hardware application systems.
Purpose	This manual is intended to give users an understanding of the functions of the CS+ to use for reference in developing the hardware or software of systems using these devices.
Organization	This manual can be broadly divided into the following units. 1. GENERAL 2. FUNCTIONS A. WINDOW REFERENCE B. Python CONSOLE/Python FUNCTIONS
How to Read This Manual	It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.
Conventions	Data significance: Higher digits on the left and lower digits on the right Active low representation: XXX (overscore over pin or signal name) Note: Footnote for item marked with Note in the text Caution: Information requiring particular attention Remarks: Supplementary information Numeric representation: Decimal ... XXXX Hexadecimal ... 0xXXXX

TABLE OF CONTENTS

1. GENERAL 5
1.1 Introduction 5
1.2 Features. 5
2. FUNCTIONS. 6
2.1 Execute Python Functions 6
2.2 Use Sample Script 6
2.3 Control CS+ in the Python 3 Execution Environment 7
A. WINDOW REFERENCE. 8
A.1 Description 8
B. Python CONSOLE/Python FUNCTIONS 11
B.1 Overview 11
B.2 Related File 11
B.3 CS+ Python Function/Class/Property/Event 12
B.3.1 CS+ Python function (for basic operation) 13
B.3.2 CS+ Python function (common) 22
B.3.3 CS+ Python function (for project) 28
B.3.4 CS+ Python function (for build tool) 44
B.3.5 CS+ Python function (for debug tool) 51
B.3.6 CS+ Python class 207
B.3.7 CS+ Python property (common). 263
B.3.8 CS+ Python property (for project). 273
B.3.9 CS+ Python property (for build tool) 280
B.3.10 CS+ Python property (for debug tool). 315
B.3.11 CS+ Python event. 336
B.4 Cautions for Python Console 338
C. External Communications with the Python 3 Execution Environment/csplus Module Functions339
Revision Record . . . . . C - 1

1. GENERAL

CS+ is an integrated development environment for use with microcontrollers. The Python console can control CS+ using IronPython (Python that runs on .NET Framework) which is a script language. The functions, properties, classes, and events to control CS+ are added to the Python console.
CS+ can be controlled by using an external communications facility in a Python 3 execution environment.

This manual describes the usage of the Python console and the functions, properties, classes, and events that have been extended for CS+.

==========
This software includes the work that is distributed in the Apache License 2.0.
http://www.apache.org/licenses/LICENSE-2.0
==========

Caution The above Web site may not be displayed from this document.

1.1 Introduction

This manual covers how to control CS+ (in creating, building, and debugging projects) by using the CS+ control functions, properties, classes, and events which are provided by CS+.

1.2 Features

The features of the Python console are shown below.

- IronPython
  The features of IronPython can be used.
  In the IronPython language usable in the Python console, in addition to the features of the Python language, various class libraries of .NET Framework can be used.
  For the language specifications of IronPython, see the following URL.
  http://ironpython.net/
- Project
  Projects can be created and loaded. The active project can also be changed.
- Build
  Build can be executed in the entire project or in file units.
- Debug
  The debug tool can be connected or disconnected, program execution can be controlled, and memory data or variables can be referred to or set.
- Obtaining sample scripts
  You can obtain sample scripts that are executable in the Python console from the Renesas Web site.
  You can also register script files with projects.
- External communications with the Python 3 execution environment
  CS+ can be controlled in the Python 3 execution environment.
  For details on Python 3, see the following URL.
  https://docs.python.org/3/

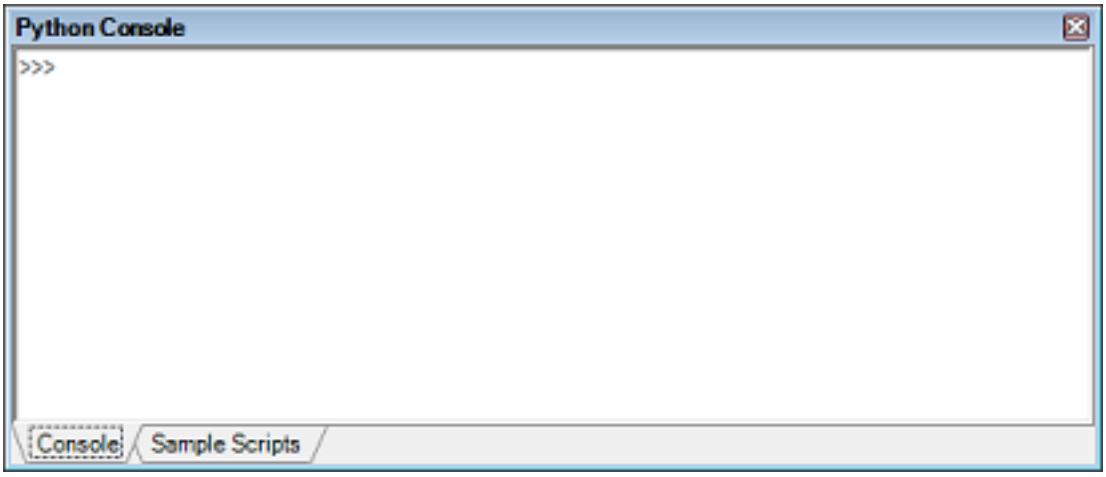
2. FUNCTIONS

This chapter describes how to use the Python console.

2.1 Execute Python Functions

CS+ enables the execution of IronPython functions and control statements, and CS+ Python functions (see "B.3 CS+ Python Function/Class/Property/Event") added for controlling CS+ via command input method.
Select [Python Console] from the [View] menu and select the [Console] tab on the Python Console panel.
You can control CS+ and the debugging tool by executing Python functions and control statements in the panel.

Figure 2.1 Python Console Panel



Caution Do not issue Python commands while building is in progress.

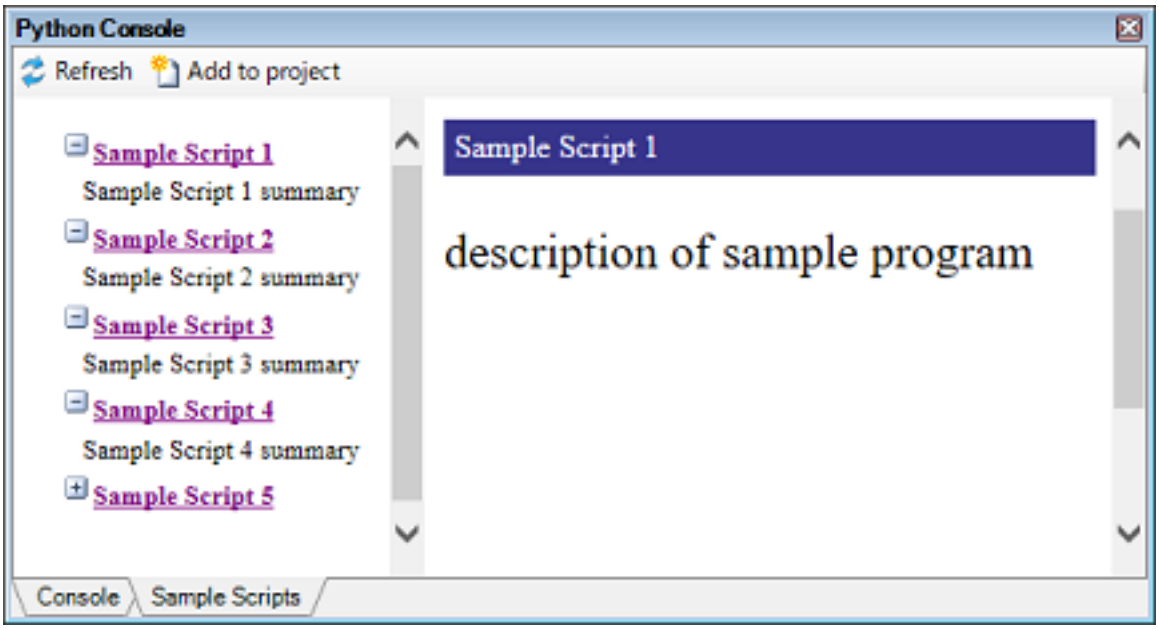
Remark See "B. Python CONSOLE/Python FUNCTIONS" for details about the Python console and Python functions.

2.2 Use Sample Script

You can obtain sample scripts that are executable in the Python console from the Renesas Web site.
You can also register script files with projects.

(1) Selecting [Python Console] from the [View] menu will open the Python Console panel.
Selecting the [Sample Scripts] tab below the panel displays a list of the sample scripts that you have obtained from the Renesas Web site.

Figure 2.2 Python Console Panel



(2) Selecting the title of a sample script will display a description of the script. Clicking on the [Add to project] button will register the script file with the active project.

(3) Double-clicking on the name of a script file in the project tree will open the registered script file in the Editor panel. Modify the script file as required.

(4) Right-click on the name of a script file in the project tree and select [Execute in Python Console]. The [Console] tab will become active and the script file will be executed.

2.3 Control CS+ in the Python 3 Execution Environment

CS+ can be controlled by scripts for Python 3. This section describes how to control CS+ from the command prompt.

(1) Start up the server for the external communications facility in the Python console of CS+.

(2) Run Python from the command prompt.

(3) Add the integration_service folder to the settings for paths.

(4) Import a csplus module by using an import statement.

(5) Call the functions of the csplus module to control CS+. Start by calling the connect() function. To close the server, call the disconnect() function or terminate() function.

```
>>> import sys
>>> sys.path.append("C:\Program Files (x86)\Renesas Electronics\CS+\CC\Plugins\Python-
Console\integration_service")
>>> import csplus
>>> csplus.connect()
>>> session_id = csplus.launch_debug_session("")
>>> csplus.teminate_debug_session(session_id)
>>> csplus.terminate()
>>>
```

For details on the functions of the csplus module, see “C. External Communications with the Python 3 Execution Environment/csplus Module Functions”.

A. WINDOW REFERENCE

This section describes the panel related to the Python console.

A.1 Description

Below is a list of the panel related to the Python console.

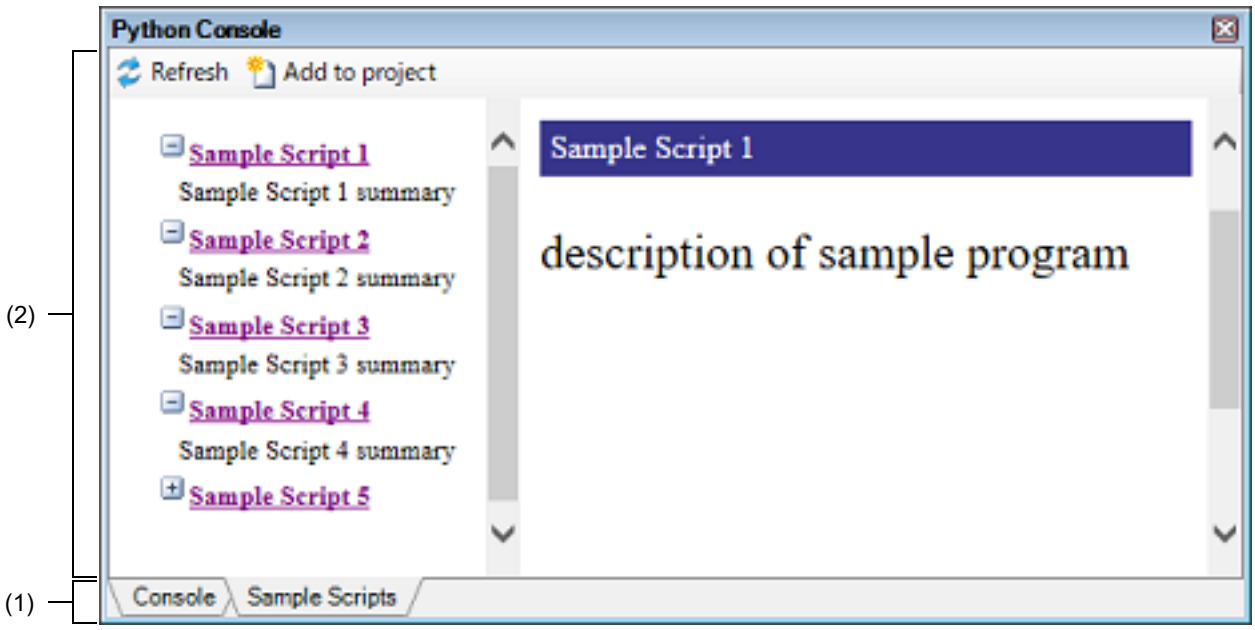
Table A.1 Panel List

Panel Name	Function Description
Python Console panel	You can use IronPython to operate CS+ and the debug tool by command input. You can also register a script with the project by simply displaying the sample script.

Python Console panel

You can use IronPython to operate CS+ and the debug tool by command input. You can also register a script with the project by simply displaying the sample script.

Figure A.1 Python Console Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [Toolbar]
- [[File] menu (Python Console panel-dedicated items)]
- [Context menu]

[How to open]

- From the [View] menu, select [Python Console].

[Description of each area]

(1) Tab selection area
Selecting the tab will switch between the type of information that is displayed in the content area.

Tab Name	Description
Console	Enter and run IronPython functions and control statements, and CS+ Python functions. The results of function execution and errors are also displayed.
Sample Scripts	Displays a sample script that is executable in the Python console and was obtained from the Renesas Web site. The script file is also registered with the project.

(2) Content area

(a) [Console] tab

Enter and run IronPython functions and control statements, and CS+ Python functions.
The results of function execution and errors are also displayed.
Use a print statement to display the result of IronPython functions.

(b) [Sample Scripts] tab
Displays a sample script that is executable in the Python console and was obtained from the Renesas Web site.
The script file is also registered with the project.

[Toolbar]

(1) [Console] tab
None

(2) [Sample Scripts] tab

Button	Function
Refresh	Updates the contents of the sample script that is displayed in the content area.
Add to project	Downloads the script file of a sample script that is being displayed in the content area to the project folder and registers the file in the project tree of the active project.

[[File] menu (Python Console panel-dedicated items)]

(1) [Console] tab
The following items are exclusive for [File] menu in the Python Console panel (other items are common to all the panels).

Save Python Console	Saves the content displayed in the current panel in the last text file (*.txt) to be saved. Note that if this item is selected first after the program starts, then the behavior is the same as selecting [Save Python Console As...].
Save Python Console As...	Opens the Save As dialog box to save the contents currently displayed on this panel in the designated text file (*.txt).

(2) [Sample Scripts] tab
None

[Context menu]

(1) [Console] tab
The following items are exclusive for [File] menu in the Python Console panel (other items are common to all the panels).

Cut	Cuts the selected characters and copies them to the clip board.
Copy	Copies the selected characters to the clip board.
Paste	Inserts the contents of the clipboard into the caret position.
Select All	Selects all characters displayed on this panel.
Abort	Forces the currently running command to stop.
Clear	Clears all output results.
Python Initialize	Initializes Python.
Select Script File...	Opens the Select Script File dialog box to execute the selected Python script file.

(2) [Sample Scripts] tab
None

B. Python CONSOLE/Python FUNCTIONS

This section describes the Python Console and Python functions provided by CS+.

B.1 Overview

The Python Console plug-in is a console tool using the IronPython language.
In addition to the functions and control statements supported by the IronPython language, you can also use CS+ Python functions added in order to control CS+.
The functions provided by CS+ are shown below.

- On the Python Console panel, you can execute IronPython functions and control statements, and CS+ Python functions (see "B.3 CS+ Python Function/Class/Property/Event" and "2.1 Execute Python Functions").
- When you start CS+ from the command line, you can specify and execute a script file (see "CS+ Integrated Development Environment User's Manual: Project Operation").
- When loading a project file, you can run a script you have prepared in advance (see "B.2 Related File").

B.2 Related File

Below is a related file of CS+ Python functions.

- *project-file-name*.py
  If there is a file in the same folder as the project file, and with the same name as the project file but with the "py" extension, then that file is executed automatically when the project file is loaded.
  The active project will be processed.

- *download-file-name*.py
  If there is a file in the same folder as the download file, and with the same name as the download file but with the "py" extension, then that file is executed automatically after downloading.

B.3 CS+ Python Function/Class/Property/Event

This section describes CS+ Python functions, classes, and properties.
Below is a list of CS+ Python functions, classes, and properties.

CS+ Python functions have the following rules.

- If a parameter has a default value, then the [Specification format] parameter is described in the form "*parameter-name=default-value*". You can also specify parameters by value only.

  Example If the [Specification format] is "function(*arg1*, *arg2* = 1, *arg3* = True)", then *arg1* has no default value; *arg2* has a default value of 1; and *arg3* has a default value of "True".
  The parameters can be specified as follows: "function("main", 1, True)".

- Parameters with default values can be omitted.
  This is only possible, however, if the parameter can be determined.

  Example If the [Specification format] is "function(*arg1*, *arg2* = 1, *arg3* = True)"

```
>>>function("main")          : It is assumed that "function("main", 1, True)"
>>>function("main", 2)       : It is assumed that "function("main", 2, True)"
>>>function("main", arg3 = False)    : It is assumed that "function("main",
1, False)"
>>>function("main", False)  : NG because it is assumed that "arg1 = False,
arg2 = "main", arg3 = 3"
```

- You can change the order in which parameters are specified by using the format "*parameter-name=default-value*".

  Example If the [Specification format] is "function(*arg1*, *arg2* = 1, *arg3* = True)"

```
>>>function(arg3 = False, arg1 = "main", arg2 = 3)       ...OK
>>>function(False, "main", 3) : NG because it is assumed that "arg1 = False,
arg2 = "main", arg3 = 3"
```

- You should be careful when you describe a path for a folder or file as parameters.
  IronPython recognizes the backslash character (\) as a control character. For example, if a folder or file name starts with a "t", then the sequence "\t" will be recognized as a tab character. Do the following to avoid this.

  Example 1. In a quoted string ("""), prepend the letter "r" to make IronPython recognize the string as a path.

```
r"C:\test\test.py"
```

  Example 2. Use a forward slash (/) instead of a backslash (\).

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

A slash (/) is used in this document.

B.3.1 CS+ Python function (for basic operation)

Below is a list of CS+ Python functions (for basic operation).

Table B.1 CS+ Python Function (For Basic Operation)

Function Name	Function Description
ClearConsole	This function clears the string displayed on the Python console.
CubeSuiteExit	This function exits from CS+.
Help	This function displays the help for the CS+ Python functions.
Hook	This function registers a hook or callback function.
Save	This function saves all editing files and projects.
Source	This function runs a script file.

ClearConsole

This function clears the string displayed on the Python console.

[Specification format]

```
ClearConsole()
```

[Argument(s)]

None

[Return value]

If the string was cleared successfully: True
If there was an error when clearing the string: False

[Detailed description]

- This function clears the string displayed on the Python console.

[Example of use]

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

CubeSuiteExit

This function exits from CS+.

[Specification format]

```
CubeSuiteExit()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- This function exits from CS+.

Caution The editing file will not be saved, even if the project file has been modified.
Use Save function to save the editing file.

[Example of use]

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

Help

This function displays the help for the CS+ Python functions.

[Specification format]

```
Help()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- This function starts CS+'s integrated help, and displays the help for CS+ Python functions.

[Example of use]

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

Hook

This function registers a hook or callback function.

[Specification format]

```
Hook(scriptFile)
```

[Argument(s)]

Argument	Description
scriptFile	Specify the script file where the hook or callback function is defined.

[Return value]

None

[Detailed description]

- This function loads *scriptFile*, and registers a hook or callback function in the script file.
  There is no problem even if functions other than a hook or callback function are declared.
  The hook or the callback function is registered when the script file is ended.
- If Hook functions are declared, they are called after CS+ events occur.

  Caution Event processing by CS+ is not completed unless execution of the hook function completes or control returns to the calling program.
- The types of hook function are shown below.
  Note that hook functions do not take parameters.

Hook Function	Event
BeforeBuild	Before build
BeforeDownload	Before download
AfterDownload	After download
AfterCpuReset	After CPU reset
BeforeCpuRun	Before execute
AfterCpuStop	After break
AfterActionEvent	After action event (only Printf event)
AfterInterrupt	After acceptance of specified exception cause code (the target is the exception cause code set in debugger.Interrupt.Notification)
AfterTimer	After occurrence of timer interrupt (the target is the timer interrupt set in debugger.Interrupt.SetTimer)

Example Sample script file

```
def BeforeDownload():
  # Processing you want to perform before the download
```

- If callback functions are declared, they are called after CS+ events occur.
- The callback function name is fixed to "pythonConsoleCallback".
  The parameter of the callback function is the callback trigger.

Argument Value	Callback Trigger
10	After event registration
11	After event deletion
12	Before start of execution
13	After break
14	After CPU reset
18	After debug tool properties are changed
19	Before download
20	After memory or register is changed
21	After action event (only Printf event)
30	Before build
50	After occurrence of specified exception cause code (after acceptance of exception cause code specified by debugger.Interrupt.Notification)
63	After period specified by XRunBreak or timer interrupt has elapsed

Caution 1. Hook functions and callback functions are initialized by the following operations.

- When a project file is loaded
- When a new project file is created
- When the active project is changed
- When the debugging tool is switched
- When Python is initialized

Caution 2. Do not include a process that enters an infinite loop in hook functions and callback functions.

Caution 3. Do not use the following functions in the hook functions and callback function.

```
debugger.ActionEvent, debugger.Breakpoint, debugger.Connect,
debugger.Disconnect, debugger.Download, debugger.Erase, debugger.Go,
debugger.Map, debugger.Next, debugger.Reset, debugger.ReturnOut,
debugger.Run, debugger.Step, debugger.Stop
```

Caution 4. It is not possible to call debugger.XRunBreak.Set or debugger.Interrupt.SetTimer with different conditions in the hook function (AfterTimer) and callback function (parameter: 63).

Example 1. Do not make the following specifications in a hook function.

```
def AfterTimer():
  debugger.Interrupt.SetTimer(1, TimeType.Ms, True)
  debugger.XRunBreak.Set(1, TimeType.Ms, True)
```

Example 2. Do not make the following specifications in a callback function.

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

Caution 5. Use the following functions when the hook function is AfterTimer or AfterInterrupt and when the parameter of the callback function is 50 or 63.

```
debugger.Address, debugger.GetIORList, debugger.Interrupt, debugger.Memory,
debugger.Register, debugger.Watch, debugger.XRunBreak
```

Note that debugger.Interrupt.SetTimer and debugger.XRunBreak cannot be used when the hook function is AfterTimer or when the parameter of the callback function is 63.

[Example of use]

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

Save

This function saves all editing files and projects.

[Specification format]

```
Save()
```

[Argument(s)]

None

[Return value]

If all editing files and projects were saved successfully: True
If there was an error when saving all editing files and projects: False

[Detailed description]

- This function saves all editing files and projects.

[Example of use]

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

Source

This function runs a script file.

[Specification format]

```
Source(scriptFile)
```

[Argument(s)]

Argument	Description
scriptFile	Specify the script file to run.

[Return value]

None

[Detailed description]

- This function runs the script file specified by *scriptFile*.
- This function operates the same as "execfile" of IronPython.

[Example of use]

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

B.3.2 CS+ Python function (common)

Below is a list of CS+ Python functions (common).

Table B.2 CS+ Python Function (Common)

Function Name	Function Description
common.GetOutputPanel	This function displays the contents of the Output panel.
common.OutputPanel	This function displays the string on the Output panel.
common.PythonInitialize	This function initializes Python.
server.Start	Start socket communication to connect with Python 3 execution environment.
server.Stop	Stop socket communication to connect with Python 3 execution environment.

common.GetOutputPanel

This function displays the contents of the Output panel.

[Specification format]

```
common.GetOutputPanel()
```

[Argument(s)]

None

[Return value]

String displayed on the Output panel

[Detailed description]

- This function displays the string displayed on the Output panel.

[Example of use]

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

common.OutputPanel

This function displays the string on the Output panel.

[Specification format]

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

[Argument(s)]

Argument	Description	
output	Specify the string displayed on the Output panel.	
messageType	Specify the type of messages to be colored in the Output panel. The colors are in accord with the settings for the [General - Font and Color] category in the Option dialog box.	
	Type	Description
	MessageType.Error	Error
	MessageType.Information	Standard (default).
	MessageType.Warning	Warning

[Return value]

If the string was displayed on the Output panel successfully: True
If there was an error when displaying the string on the Output panel: False

[Detailed description]

- This function displays the string specified by *output* on the Output panel.

[Example of use]

```
>>>common.OutputPanel("An error occurred.", MessageType.Error)
True
>>>
```

common.PythonInitialize

This function initializes Python.

[Specification format]

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

[Argument(s)]

Argument	Description
scriptFile	Specify the script file to run after initializing Python (default: not specified). Specify the absolute path.

[Return value]

None

[Detailed description]

- This function initializes Python.
  Initialization is performed by discarding all defined functions or imported modules. If this function is executed while executing a script, Python is forcibly initialized regardless of the execution state.
- If a script file is specified in *scriptFile*, the specified script file is executed after initialization has finished.
- If *scriptFile* is not specified, Python is merely initialized.

Caution Since Python is forcibly initialized, an error may be displayed depending on the execution state.

[Example of use]

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

server.Start

Start socket communication to connect with Python 3 execution environment.

[Specification format]

```
server.Start()
```

[Argument(s)]

None

[Return value]

None

[Example of use]

```
>>> server.Start()
>>> server.Stop()
>>>
```

server.Stop

Stop socket communication to connect with Python 3 execution environment.

[Specification format]

```
server.Stop()
```

[Argument(s)]

None

[Return value]

None

[Example of use]

```
>>> server.Start()
>>> server.Stop()
>>>
```

B.3.3 CS+ Python function (for project)

Below is a list of CS+ Python functions (for a project).

Table B.3 CS+ Python Function (For Project)

Function Name	Function Description
project.Change	This function changes the active project.
project.Close	This function closes a project.
project.Create	This function creates a new project.
project.File.Add	This function adds a file to the active project.
project.File.Exists	This function confirms whether the file exists in the active project.
project.File.Information	This function displays the list of the files registered in the active project.
project.File.Remove	This function removes a file from the active project.
project.GetDeviceNameList	This function displays the list of the device names of the microcontroller.
project.GetFunctionList	This function displays the list of the functions of the active project.
project.GetVariableList	This function displays the list of the variables of the active project.
project.Information	This function displays the list of project files.
project.Open	This function opens a project.

project.Change

This function changes the active project.

[Specification format]

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

[Argument(s)]

Argument	Description
projectName	Specify the full path of the project or subproject to be changed.

[Return value]

If the active project was changed successfully: True
If there was an error when changing the active project: False

[Detailed description]

- This function changes the project specified in *projectName* to the active project.
- The project file specified in *projectName* must be included the currently opened project.

[Example of use]

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

project.Close

This function closes a project.

[Specification format]

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

[Argument(s)]

Argument	Description
save	Specify whether to save all files being edited and a project. True: Save all editing files and a project. False: Do not save all editing files and a project (default).

[Return value]

If the project was closed successfully: True
If there was an error when closing the project: False

[Detailed description]

- This function closes a currently opened project.
- If *save* is set to "True", then all files being edited and a project are saved.

[Example of use]

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

project.Create

This function creates a new project.

[Specification format]

```
project.Create(fileName, micomType, deviceName, projectKind = ProjectKind.Auto, com-
piler = Compiler.Auto, subProject = False)
```

[Argument(s)]

<table>
<tr><th>Argument</th><th colspan="2">Description</th></tr>
<tr><td>fileName</td><td colspan="2">Specify the full path of a new project file.
If no file extension is specified, the filename is automatically supplemented.
If the project to be created is a main project (subProject = False) or a subproject (subProject = True), the name is supplemented by ".mtpj" or ".mtsp", respectively.
When the extension is other than that specified, it is replaced by the actual extension.</td></tr>
<tr><td rowspan="7">micomType</td><td colspan="2">Specify the microcontroller type of a new project.
The types that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>MicomType.RH850</td><td>Project for RH850</td></tr>
<tr><td>MicomType.RX</td><td>Project for RX</td></tr>
<tr><td>MicomType.V850</td><td>Project for V850</td></tr>
<tr><td>MicomType.RL78</td><td>Project for RL78</td></tr>
<tr><td>MicomType.K0R</td><td>Project for 78K0R</td></tr>
<tr><td>MicomType.K0</td><td>Project for 78K0</td></tr>
<tr><td>deviceName</td><td colspan="2">Specify the device name of the microcontroller of a new project by a string.</td></tr>
</table>

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>projectKind</td><td>Specify the type of a new project.
The types that can be specified are shown below.
The following is automatically specified if the microcontroller type is RH850 and "ProjectKind.Auto" is specified or projectKind is not specified.
When the microcontroller is single core: ProjectKind.Application
When the microcontroller is multi-core and main project: ProjectKind.MulticoreBootLoader
When the microcontroller is multi-core and subproject: ProjectKind.MulticoreApplication
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>ProjectKind.Application</td><td>Project for application</td></tr>
<tr><td>ProjectKind.Library</td><td>Project for library</td></tr>
<tr><td>ProjectKind.DebugOnly</td><td>Debug-dedicated project</td></tr>
<tr><td>ProjectKind.Empty</td><td>Project for empty application</td></tr>
<tr><td>ProjectKind.CppApplication</td><td>Project for C++ application</td></tr>
<tr><td>ProjectKind.GHSCCProject</td><td>CS+ project using an existing GHS project file</td></tr>
<tr><td>ProjectKind.RI600V4</td><td>Project for RI600V4</td></tr>
<tr><td>ProjectKind.RI600PX</td><td>Project for RI600PX</td></tr>
<tr><td>ProjectKind.RI850V4</td><td>Project for RI850V4</td></tr>
<tr><td>ProjectKind.RI850MP</td><td>Project for RI850MP</td></tr>
<tr><td>ProjectKind.RV850</td><td>Project for RV850</td></tr>
<tr><td>ProjectKind.RI78V4</td><td>Project for RI78V4</td></tr>
<tr><td>ProjectKind.MulticoreBootLoader</td><td>Project for boot loader for multi-core</td></tr>
<tr><td>ProjectKind.MulticoreApplication</td><td>Project for application for multi-core</td></tr>
<tr><td>ProjectKind.Auto</td><td>The type of a project is selected in accord with the specification for micomType, deviceName, and subProject (default).</td></tr>
</table></td></tr>
</table>

Argument	Description	
compiler	Specify the compiler to be used. If the compiler is not specified, it is selected automatically depending on the microcontroller type.	
	Type	Description
	Compiler.Auto	The compiler to be used is selected in accord with the specification for *micomType* (default).
	Compiler.CC_RH	CC-RH If this argument is not specified when *micomType* is set to "MicomType.RH850", CC-RH is selected automatically.
	Compiler.CC_RX	CC-RX If this argument is not specified when *micomType* is set to "MicomType.RX", CC-RX is selected automatically.
	Compiler.CA850	CA850 If this argument is not specified when *micomType* is set to "MicomType.V850" and *deviceName* is set to "V850E" or "V850ES", CA850 is selected automatically.
	Compiler.CX	CX If this argument is not specified when *micomType* is set to "MicomType.V850" and *deviceName* is set to "V850E2", CX is selected automatically.
	Compiler.CC_RL	CC-RL If this argument is not specified when "MicomType.RL78" in CS+ for CC, CC-RL is selected automatically.
	Compiler.CA78K0R	CA78K0R If this argument is not specified when *micomType* is set to "MicomType.K0R" or "MicomType.RL78" in CS+ for CACX, CA78K0R is selected automatically.
	Compiler.CA78K0	CA78K0 If this argument is not specified when *micomType* is set to "MicomType.K0", CA78K0 is selected automatically.
	Compiler.GHSCC	GHSCC The compiler from Green Hills Software.
subProject	Specify whether to create a main project or a subproject. False: Create a main project (default). True: Create a subproject.	

[Return value]

If a new project was created successfully: True
If there was an error when creating a new project: False

[Detailed description]

- This function creates a new project file specified by *fileName*.
  Specify the microcontroller of the project by *micomType* and *deviceName*.
  Specify the kind of the project by *projectKind*.
- If *subProject* is set to "True", then a subproject is created.

[Example of use]

```
>>>project.Create("C:/project/test.mtpj", MicomType.RX, "R5F52105AxFN", Project-
Kind.Application)
True
>>>
```

project.File.Add

This function adds a file to the active project.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify the full path of the file to be added to the active project. When specifying multiple files, specify in the format ["file1", "file2"].
category	Specify the category that the file is added (default: not specified). When specifying multiple levels, specify in the format ["one", "two"].

[Return value]

If a file was added to the active project successfully: True
If there was an error when a file was added to the active project: False
If there was an error when any files were added to the active project when multiple files were specified for *fileName*: False

[Detailed description]

- This function adds the file specified in *fileName* to the active project.
- If *category* is specified, the file is added below that category.
  If the specified category does not exist, it is created newly.

[Example of use]

```
>>>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

This function confirms whether the file exists in the active project.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify the full path of the file whose existence in the active project is to be checked.

[Return value]

If the specified file existed in the active project: True
If the specified file did not exist in the active project: False

[Detailed description]

- This function confirms whether the file specified in *fileName* exists in the active project.

[Example of use]

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

project.File.Information

This function displays the list of the files registered in the active project.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of the files registered in the active project (in a full path)

[Detailed description]

- This function displays the list of the full path of the files registered in the active project.

[Example of use]

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

project.File.Remove

This function removes a file from the active project.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify the full path of the file to be removed from the active project. When specifying multiple files, specify in the format ["file1", "file2"].

[Return value]

If a file was removed from the active project successfully: True
If there was an error when a file was removed from the active project: False

[Detailed description]

- This function removes the file specified in *fileName* from the active project.
- The file is not deleted.

[Example of use]

```
>>>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

This function displays the list of the device names of the microcontroller.

[Specification format]

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

[Argument(s)]

Argument	Description	
micomType	Specify the microcontroller type of a new project. The types that can be specified are shown below.	
	Type	Description
	MicomType.RH850	Project for RH850
	MicomType.RX	Project for RX
	MicomType.V850	Project for V850
	MicomType.RL78	Project for RL78
	MicomType.K0R	Project for 78K0R
	MicomType.K0	Project for 78K0
nickName	Specify the nickname of the microcontroller by a string (default: not specified). Specify a character string displayed in the first layer of the [Using microcontroller] list in the Create Project dialog box that is used to create a new project.	

[Return value]

List of device names

[Detailed description]

- This function displays the list of the device names of the microcontroller specified by *micomType*.
- When *nickName* is specified, only the names of the devices specified by nickName are displayed.

[Example of use]

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

project.GetFunctionList

This function displays the list of the functions of the active project.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify the full path of the file that the list of the functions are displayed (default: not specified).

[Return value]

List of function information (see the FunctionInfo property for detail)

[Detailed description]

- This function displays the list of the functions of the active project shown by the following format.

```
function-name return-value-type start-address end-address file-name
```

- When *fileName* is specified, only the functions included in the specified file are displayed.
- When *fileName* is not specified, then all the functions will be displayed.

Caution This function uses the information displayed in the list of functions for program analysis.

[Example of use]

```
>>>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

This function displays the list of the variables of the active project.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify the full path of the file that the list of the variables are displayed (default: not specified).

[Return value]

List of variable information (see the VariableInfo property for detail)

[Detailed description]

- This function displays the list of the variables of the active project shown by the following format.

```
variable-name attribute type address size file-name
```

- When *fileName* is specified, only the variables included in the specified file are displayed.
- When *fileName* is not specified, then all the variables will be displayed.

Caution This function uses the information displayed in the list of variables for program analysis.

[Example of use]

```
>>>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

This function displays the list of project files.

[Specification format]

```
project.Information()
```

[Argument(s)]

None

[Return value]

List of project file names

[Detailed description]

- This function displays the list of project files of the main project and subprojects included in the loaded project.

[Example of use]

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

project.Open

This function opens a project.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a project file.
save	If another project was opened, specify whether to save any files being edited and the project when you close it. True: Save all editing files and a project. False: Do not save all editing files and a project (default).

[Return value]

If the project was closed successfully: True
If there was an error when closing the project: False

[Detailed description]

- This function opens a project specified by *fileName*.
- If other project is opened, that project is closed.
  If *save* is set to "True", then all files being edited and a project are saved.
- If other project is not opened, the setting of *save* is ignored.

[Example of use]

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

B.3.4 CS+ Python function (for build tool)

Below is a list of CS+ Python functions (for the build tool).

Table B.4 CS+ Python Function (For build Tool)

Function Name	Function Description
build.All	This function runs a build.
build.ChangeBuildMode	This function changes the build mode.
build.Clean	This function runs a clean.
build.File	This function runs a build of a specified file.
build.Stop	This function stops the currently running build.
build.Update	This function updates the dependencies for the build tool.

build.All

This function runs a build.

[Specification format]

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

[Argument(s)]

Argument	Description
rebuild	Specify whether to run a rebuild of a project. True: Run a rebuild of a project. False: Run a build of a project (default).
waitBuild	Specify whether to wait until completing a build. True: Wait until completing a build (default). False: Return a prompt without waiting to complete a build.

[Return value]

- When *waitBuild* is set to "True"
  If a build was completed successfully: True
  If a build failed or was canceled: False

- When *waitBuild* is set to "False"
  If a build successfully started execution: True
  If a build failed to start execution: False

[Detailed description]

- This function runs a build of a project.
  If a subproject is added to the project, a build of the subproject is run.
- If *rebuild* is set to "True", then a rebuild of a project is run.
- If *waitBuild* is set to "False", then a prompt is returned without waiting to complete a build.
- Regardless of whether a build is successful, the build.BuildCompleted event is issued when a build completes.

[Example of use]

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

build.ChangeBuildMode

This function changes the build mode.

[Specification format]

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

[Argument(s)]

Argument	Description
buildmode	Specify the build mode to be changed to with a string.

[Return value]

If the build mode was changed successfully: True
If there was an error when changing the build mode: False

[Detailed description]

- This function changes the build modes of the main project and subprojects to the build mode specified in *buildmode*.
- If *buildmode* does not exist in the project, a new build mode is created based on "DefaultBuild", and then the build mode is changed to that.

[Example of use]

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

build.Clean

This function runs a clean.

[Specification format]

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

[Argument(s)]

Argument	Description
all	Specify whether to clean a project including subprojects. True: Clean all project including subprojects. False: Clean an active project (default).

[Return value]

If a clean was completed successfully: True
If there was an error when running a clean: False

[Detailed description]

- This function runs a clean of a project (removes the files generated by a build).
- If *all* is set to "True", then a clean of the subproject is run.

[Example of use]

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

build.File

This function runs a build of a specified file.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a file to run a build.
rebuild	Specify whether to run a rebuild of a specified file. True: Run a rebuild of a specified file. False: Run a build of a specified file (default).
waitBuild	Specify whether to wait until completing a build. True: Wait until completing a build (default). False: Return a prompt without waiting to complete a build.

[Return value]

- When *waitBuild* is set to "True"
  If a build was completed successfully: True
  If there was an error when running a build: False

- When *waitBuild* is set to "False"
  If a build successfully started execution: True
  If a build failed to start execution: False

[Detailed description]

- This function runs a build of a file specified by *fileName*.
- If *rebuild* is set to "True", then a rebuild of a specified file is run.
- If *waitBuild* is set to "False", then a prompt is returned without waiting to complete a build.
- The build.BuildCompleted event is issued when a build completes.

[Example of use]

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

build.Stop

This function stops the currently running build.

[Specification format]

```
build.Stop()
```

[Argument(s)]

None

[Return value]

If the build was stopped successfully: True
If there was an error when stopping the build: False

[Detailed description]

- This function stops the currently running build.

[Example of use]

```
>>>build.All(True, False)
True
>>>build.Stop()
True
>>>
```

build.Update

This function updates the dependencies for the build tool.

[Specification format]

```
build.Update()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- This function updates the dependencies of the files during build.

[Example of use]

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

B.3.5 CS+ Python function (for debug tool)

Below is a list of CS+ Python functions (for the debug tool).

Table B.5 CS+ Python Function (For Debug Tool)

Function Name	Function Description
debugger.ActionEvent.Delete	This function deletes an action event.
debugger.ActionEvent.Disable	This function disables an action event setting.
debugger.ActionEvent.Enable	This function enables an action event setting.
debugger.ActionEvent.Get	This function references the result of the action event (Printf event).
debugger.ActionEvent.Information	This function displays action event information.
debugger.ActionEvent.Set	This function sets an action event.
debugger.Address	This function evaluates an address expression.
debugger.Assemble.Disassemble	This function performs disassembly.
debugger.Assemble.LineAssemble	This function performs line assembly.
debugger.Breakpoint.Delete	This function deletes a break point.
debugger.Breakpoint.Disable	This function disables a break point setting.
debugger.Breakpoint.Enable	This function enables a break point setting.
debugger.Breakpoint.Information	This function displays break point information.
debugger.Breakpoint.Set	This function configures a break point.
debugger.Connect	This function connects to the debug tool.
debugger.CurrentConsumption.Clear	This function clears current consumption data.
debugger.CurrentConsumption.Disable	This function disables acquiring current consumption data.
debugger.CurrentConsumption.Enable	This function enables acquiring current consumption data.
debugger.CurrentConsumption.Get	This function is used to display the maximum and average values of data on current consumption data that have been acquired.
debugger.CurrentConsumption.Information	This function displays information on acquiring current consumption data.
debugger.DebugTool.Change	This function changes the debug tool.
debugger.DebugTool.GetType	This function displays information about the debug tool.
debugger.DebugTool.RestoreState	This function restores the state of the debug tool to the one saved in the file.
debugger.DebugTool.SaveState	This function saves the state of the debug tool to a file.
debugger.Disconnect	This function disconnects from the debug tool.
debugger.Download.Binary	This function downloads a binary file.
debugger.Download.Binary64Kb	This function downloads a binary file in within-64 KB format.
debugger.Download.BinaryBank	This function downloads a binary file in memory bank format.
debugger.Download.Coverage	This function downloads coverage data.
debugger.Download.Hex	This function downloads a hex file.
debugger.Download.Hex64Kb	This function downloads a hex file in within-64 KB format.

Function Name	Function Description
debugger.Download.HexBank	This function downloads a hex file in memory bank format.
debugger.Download.HexIdTag	This function downloads a hex file with ID tag.
debugger.Download.Information	This function displays download information.
debugger.Download.LoadModule	This function downloads a load module.
debugger.Erase	This function erases the Flash memory.
debugger.GetBreakStatus	This function displays a break condition.
debugger.GetCpuStatus	This function displays the current CPU status.
debugger.GetIeStatus	This function displays the current IE status.
debugger.GetIORList	This function displays a list of the IORs and SFRs.
debugger.GetPC	This function displays the PC value.
debugger.GetProcessorElementNames	This function displays a list of the PE names of multiple cores.
debugger.Go	This function continues program execution.
debugger.Ie.GetValue debugger.Ie.SetValue	This function sets or refers to the IE register or DCU register.
debugger.Interrupt.DeleteTimer	This function deletes the timer interrupt setting.
debugger.Interrupt.Notification	This function sets exception cause codes whose notification is accepted.
debugger.Interrupt.OccurEI	This function generates EI-level interrupts.
debugger.Interrupt.OccurFE	This function generates FE-level interrupts.
debugger.Interrupt.PseudoInterrupt	This function generates pseudo-interrupts.
debugger.Interrupt.ReferTimer	This function displays the timer interrupt setting information.
debugger.Interrupt.RequestEI	This function sends an EI-level interrupt request to the interrupt controller.
debugger.Interrupt.RequestFE	This function sends an FE-level interrupt request to the interrupt controller.
debugger.Interrupt.RequestFENMI	This function sends an NMI request to the interrupt controller.
debugger.Interrupt.SetTimer	This function sets the timer interrupt.
debugger.IsConnected	This function checks the connection status of the debug tool.
debugger.IsRunning	This function checks the execution status of the debug tool.
debugger.Jump.File debugger.Jump.Address	This function displays each panel.
debugger.Map.Clear	This function clears the mapping settings.
debugger.Map.Information	This function displays map information.
debugger.Map.Set	This function configures memory mapping.
debugger.Memory.Copy	This function copies the memory.
debugger.Memory.Fill	This function fills the memory.
debugger.Memory.Read	This function refers to the memory.

Function Name	Function Description
debugger.Memory.ReadRange	This function refers to the specified number of locations in memory.
debugger.Memory.Write	This function writes to the memory.
debugger.Memory.WriteRange	This function writes multiple data to the memory.
debugger.Next	This function performs procedure step execution.
debugger.Performance.Delete	This function deletes a condition of performance measurement.
debugger.Performance.Disable	This function disables performance measurement.
debugger.Performance.Enable	This function enables performance measurement.
debugger.Performance.Get	This function references the result of performance measurement.
debugger.Performance.Information	This function displays performance measurement information.
debugger.Performance.Set	This function sets performance measurement.
debugger.PseudoError.Clear	This function clears the error status of all pseudo-errors.
debugger.PseudoError.Get	This function references ECM error information.
debugger.PseudoError.SetGo	This function sets conditions of a pseudo-error and runs a program.
debugger.PseudoTimer.Delete	This function deletes pseudo-timers.
debugger.PseudoTimer.Information	This function displays the pseudo-timer information.
debugger.PseudoTimer.Set	This function sets pseudo-timers.
debugger.RecoverSWAS	This function recovers the Switch Area Status.
debugger.Register.GetValue	This function refers to register/IO register/SFR.
debugger.Register.SetValue	This function sets the value of a register/IO register/SFR.
debugger.Reset	This function resets the CPU.
debugger.ReturnOut	This function runs until control returns to the program that called the current function.
debugger.Run	This function resets and then run the program.
debugger.SaveRegisterBank.Information	This function displays information on the save register bank.
debugger.SoftwareTrace.Delete	This function deletes a software trace.
debugger.SoftwareTrace.Disable	This function disables a software trace.
debugger.SoftwareTrace.Enable	This function enables a software trace.
debugger.SoftwareTrace.Get	This function refers to the software trace data for the specified number of frames. This function also outputs the acquired software trace data to a file.
debugger.SoftwareTrace.Information	This function displays software trace information.
debugger.SoftwareTrace.Set	This function sets a software trace.
debugger.SoftwareTraceLPD.Delete	This function deletes a software trace (LPD output).
debugger.SoftwareTraceLPD.Disable	This function disables a software trace (LPD output).
debugger.SoftwareTraceLPD.Enable	This function enables a software trace (LPD output).

Function Name	Function Description
debugger.SoftwareTraceLPD.Get	This function refers to the software trace (LPD output) data for the specified number of frames. This function also outputs the acquired software trace (LPD output) data to a file.
debugger.SoftwareTraceLPD.Information	This function displays software trace (LPD output) information.
debugger.SoftwareTraceLPD.Set	This function sets a software trace (LPD output).
debugger.Step	This function performs step execution.
debugger.Stop	This function stops the execution of the debug tool.
debugger.Timer.Clear	This function clears the result measured by a conditional timer.
debugger.Timer.Delete	This function deletes a conditional timer.
debugger.Timer.Detail	This function sets measurement conditions of a conditional timer.
debugger.Timer.Disable	This function disables a conditional timer.
debugger.Timer.Enable	This function enables a conditional timer.
debugger.Timer.Get	This function references the result measured by a conditional timer.
debugger.Timer.Information	This function displays conditional timer information.
debugger.Timer.Set	This function sets a conditional timer.
debugger.Trace.Clear	This function clears the trace memory.
debugger.Trace.Delete	This function deletes a conditional trace.
debugger.Trace.Disable	This function disables a conditional trace.
debugger.Trace.Enable	This function enables a conditional trace.
debugger.Trace.Get	This function dumps the trace data.
debugger.Trace.Information	This function displays conditional trace information.
debugger.Trace.Set	This function sets a conditional trace.
debugger.Upload.Binary	This function saves the memory data in binary format.
debugger.Upload.Coverage	This function saves the coverage data.
debugger.Upload.Intel	This function saves the memory data in Intel format.
debugger.Upload.Motorola	This function saves the memory data in Motorola format.
debugger.Watch.GetValue	This function refers to a variable value.
debugger.Watch.SetValue	This function sets a variable value.
debugger.Where	This function displays a stack backtrace.
debugger.Whereami	This function displays a location.
debugger.XCoverage.Clear	This function clears the coverage memory.
debugger.XCoverage.GetCoverage	This function gets the coverage.
debugger.XRunBreak.Delete	This function deletes XRunBreak setting information.
debugger.XRunBreak.Refer	This function displays XRunBreak setting information.
debugger.XRunBreak.Set	This function configures XRunBreak settings.

Function Name	Function Description
debugger.XTime	This function displays timing information between Go and Break.
debugger.XTrace.Clear	This function clears the trace memory.
debugger.XTrace.Dump	This function dumps the trace data.
TraceInfo.CreateOtherDict	This function converts the value of TraceInfo.Other into the dict type.

debugger.ActionEvent.Delete

This function deletes an action event.

[Specification format]

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

[Argument(s)]

Argument	Description
actionEventNumber	Specify the action event number to delete.

[Return value]

If an action event was deleted successfully: True
If there was an error when deleting an action event: False

[Detailed description]

- This function deletes the action event specified by *actionEventNumber*.
- If *actionEventNumber* is not specified, then events of all action event numbers will be deleted.

[Example of use]

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

debugger.ActionEvent.Disable

This function disables an action event setting.

[Specification format]

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

[Argument(s)]

Argument	Description
actionEventNumber	Specify the action event number to disable.

[Return value]

If an action event setting was disabled successfully: True
If there was an error when disabling an action event setting: False

[Detailed description]

- This function disables the action event specified by *actionEventNumber*.
- If *actionEventNumber* is not specified, then events of all action event numbers will be disabled.

[Example of use]

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

debugger.ActionEvent.Enable

This function enables an action event setting.

[Specification format]

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

[Argument(s)]

Argument	Description
actionEventNumber	Specify the action event number to enable.

[Return value]

If an action event setting was enabled successfully: True
If there was an error when enabling an action event setting: False

[Detailed description]

- This function enables the action event specified by *actionEventNumber*.
- If *actionEventNumber* is not specified, then events of all action event numbers will be enabled.

[Example of use]

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

debugger.ActionEvent.Get

This function references the result of the action event (Printf event).

[Specification format]

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

[Argument(s)]

Argument	Description
output	Specify the string to be attached when the result of an action event is output (default: not specified). Note that this argument should be specified when wishing to acquire only a result matching this argument.

[Return value]

List of result of action event (see the ActionInfo class for detail)

[Detailed description]

- This function holds the result acquired when executing the instruction at the address set as a condition of an action event (Printf event) in the Python console, and all results held up to that moment will be referenced at the timing of this function debugger.ActionEvent.Get being called.
- If *output* is specified, only the result matching *output* is output. Comparison is performed to detect a perfect match.
- If *output* is not specified, the results of all accumulated action events are output.
- To acquire the result at the timing when an action event has occurred, use Hook. For the maximum number of results that can be held in the Python console, see the debugger.ActionEvent.GetLine property.

Caution After a result has been referenced, the result of the action event which was held in the Python console is initialized. Therefore, once a result has been referenced, it cannot be referenced again.

- The result of an action event is displayed in the following format.

```
string-to-be-attached-at-output variable-expression
```

[Example of use]

```
>>>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

This function displays action event information.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of action event information (see the ActionEventInfo class for detail)

[Detailed description]

- This function displays information on the action event that has been set in the following format.
  - For the Printf event

    action-event-number action-event-name state address string-to-be-attached-at-output variable-expression

  - For the interrupt event

    action-event-number action-event-name state address Interrupt vector: *interrupt-vector-number* Priority level: *interrupt-priority*

[Example of use]

```
>>>ai = debugger.ActionEvent.Information()
1 Python Action Event0001 Enable main results: chData
2 Python Action Event0002 Disable sub Interrupt vector: 0x1c Priority level: 7
>>>print ai[0].Number
1
>>>print ai[0].Name
Python Action Event0001
>>>
```

debugger.ActionEvent.Set

This function sets an action event.

[Specification format]

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

[Argument(s)]

Argument	Description
ActionEventCondition	Specify a condition of an action event. See the ActionEventCondition class for creating an action event.

[Return value]

Set action event number (numerical value)

[Detailed description]

- This function sets an action event according to the contents specified with *ActionEventCondition*.
- The specified action event is registered with the following name.

```
Python Action Eventnumerical-value
```

[Example of use]

```
>>>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

This function evaluates an address expression.

[Specification format]

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

[Argument(s)]

Argument	Description
expression	Specify an address expression.

[Return value]

Converted address (numerical value)

[Detailed description]

- This function converts the address expression specified by *expression* into the address.

Caution 1. If a script is specified to execute in the CubeSuite+.exe startup options, then the symbol conversion function will not be available until the debugging tool is connected. In other words, this function cannot be used, so execute it after connection.

Caution 2. When a load module name or file name is specified in an address expression, it needs to be enclosed in double quotation marks (" ") in some cases. See "CS+ Integrated Development Environment User's Manual: Debug Tool" for details.

Example When file name "C:\path\test.c" and function "sub" are specified

```
"\"C:/path/test.c\"#sub"
```

Or

```
"\"C:\\path\\test.c\"#sub"
```

[Example of use]

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

debugger.Assemble.Disassemble

This function performs disassembly.

[Specification format]

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

[Argument(s)]

Argument	Description
address	Specify the address at which to start disassembly.
number	Specify the number of lines to display (default: 1).
code	Specify whether to display instruction codes. True: Display instruction codes (default). False: Do not display instruction codes.

[Return value]

List of result of disassembly (see the DisassembleInfo property for detail)

[Detailed description]

- This function performs disassembly from the address specified by *address*.
- If *number* is specified, the specified number of lines are displayed.
- If *code* is set to "False", then instruction codes are not displayed.
- If "." is specified in *address*, then it is interpreted as the address following the last address disassembled.

[Example of use]

```
>>>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

This function performs line assembly.

[Specification format]

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

[Argument(s)]

Argument	Description
address	Specify the address at which to start assembly.
code	Specify the string to assemble.

[Return value]

If line assembly was performed successfully: True
If there was an error when performing line assembly: False

[Detailed description]

- This function performs assembly of the string specified by *code* from the address specified by *address*.
- If "." is specified in *address*, then it is interpreted as the address following the last address assembled.

[Example of use]

```
>>>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

This function deletes a break point.

[Specification format]

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

[Argument(s)]

Argument	Description
breakNumber	Specify the break event number to delete.

[Return value]

If a break point was deleted successfully: True
If there was an error when deleting a break point: False

[Detailed description]

- This function deletes the break event specified by *breakNumber*.
- If *breakNumber* is not specified, then breaks of all break event numbers will be deleted.

[Example of use]

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

debugger.Breakpoint.Disable

This function disables a break point setting.

[Specification format]

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

[Argument(s)]

Argument	Description
breakNumber	Specify the break event number to disable.

[Return value]

If a break point setting was disabled successfully: True
If there was an error when disabling a break point setting: False

[Detailed description]

- This function disables the break event specified by *breakNumber*.
- If *breakNumber* is not specified, then breaks of all break event numbers will be disabled.

[Example of use]

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

debugger.Breakpoint.Enable

This function enables a break point setting.

[Specification format]

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

[Argument(s)]

Argument	Description
breakNumber	Specify the break event number to enable.

[Return value]

If a break point setting was enabled successfully: True
If there was an error when enabling a break point setting: False

[Detailed description]

- This function enables the break event specified by *breakNumber*.
- If *breakNumber* is not specified, then breaks of all break event numbers will be enabled.

[Example of use]

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

debugger.Breakpoint.Information

This function displays break point information.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of break point information (see the BreakpointInfo property for detail)

[Detailed description]

- This function displays the break point settings in the following format.

```
break-event-number break-name state address-location
```

[Example of use]

```
>>>debugger.Breakpoint.Information()
   1 PythonBreak0001 Enable  0x000002dc
   2 Break0001 Enable  test1.c#_sub1
   3 PythonBreak0002 Enable  0x000002ec
   4 Break0002 Enable  test1.c#_sub1+10
>>>
```

debugger.Breakpoint.Set

This function configures a break point.

[Specification format]

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

[Argument(s)]

Argument	Description
BreakCondition	Specify a break condition. See the BreakCondition property for details about creating break conditions.

[Return value]

Set break event number (numerical value)

[Detailed description]

- This function sets a break point according to the specifications in *BreakCondition*.
- *break-name* is "PythonBreak*xxxx*" (*xxxx*: 4-digit number).

[Example of use]

```
>>>Condition = BreakCondition()
>>>Condition.Address = "main"
>>>breakNumber = debugger.Breakpoint.Set(Condition)
1
>>>print breakNumber
1
>>>debugger.Breakpoint.Information()
   1 PythonBreak0001 Enable  0x000002dc
```

debugger.Connect

This function connects to the debug tool.

[Specification format]

```
debugger.Connect()
```

[Argument(s)]

None

[Return value]

If the debug tool was connected successfully: True
If there was an error when connecting to the debug tool: False

[Detailed description]

- This function connects to the debug tool.

[Example of use]

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

debugger.CurrentConsumption.Clear

This function clears current consumption data. [RL78 (devices with support for peripheral function simulation)] [Simulator]

[Specification format]

```
debugger.CurrentConsumption.Clear()
```

[Argument(s)]

None

[Return value]

If current consumption data was cleared successfully: True
If there was an error when clearing current consumption data: False

[Detailed description]

- This function clears current consumption data.

[Example of use]

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

debugger.CurrentConsumption.Disable

This function disables acquiring current consumption data. [RL78 (devices with support for peripheral function simulation)] [Simulator]

[Specification format]

```
debugger.CurrentConsumption.Disable()
```

[Argument(s)]

None

[Return value]

If acquiring current consumption data was disabled successfully: True
If there was an error when acquiring current consumption data was disabled: False

[Detailed description]

- This function disables acquiring current consumption data.

[Example of use]

```
>>>debugger.CurrentConsumption.Disable()
True
>>>
```

debugger.CurrentConsumption.Enable

This function enables acquiring current consumption data.
If you run a program with this feature enabled, current consumption data will be acquired. [RL78 (devices with support for peripheral function simulation)] [Simulator]

[Specification format]

```
debugger.CurrentConsumption.Enable()
```

[Argument(s)]

None

[Return value]

If acquiring current consumption data was enabled successfully: True
If there was an error when acquiring current consumption data was enabled: False

[Detailed description]

- This function enables acquiring current consumption data.

[Example of use]

```
>>>debugger.CurrentConsumption.Enable()
True
>>>
```

debugger.CurrentConsumption.Get

This function is used to display the maximum and average values of data on current consumption data that have been acquired.

This function also outputs current consumption data to an XML file. [RL78 (devices with support for peripheral function simulation)] [Simulator]

[Specification format]

```
debugger.CurrentConsumption.Get(fileName = "", force = False)
```

[Argument(s)]

Argument	Description
fileName	Specify the full path of the XML file name to which current consumption data will be saved (default: not specified).
force	Specify whether to overwrite the XML file. True: Overwrite the XML file. False: Do not overwrite the XML file (default).

[Return value]

Information of current consumption data (see the CurrentConsumptionInfo class for detail)

[Detailed description]

- The current consumption data is shown by the following format.

```
Max = maximum-current(uA), Average = average-current(uA)
```

- When *fileName* is not specified, then current consumption data will not be saved to an XML file.
- The format of the XML file to be saved is as follows.

Caution There is a possibility that the format will be changed in the future.

```
<?xml version="1.0" encoding="UTF-8"?>
-<Root>
   <FileType>0</FileType>
   <DateTime>YYYY-MM-DD hh:mm:ss</DateTime>  ... Time on which the file was created
  -<Modules>
     <Module no="0" name="Peripheral function name0" />    ... ID number definition
of the peripheral function
     <Module no="1" name="Peripheral function name1" />
       :
     <Module no="n" name="Peripheral function namen" />
   </Modules>
     <!-- Frame n=FrameNo Address;Time(ns);ModuleNo,Current(uA);... -->
   <F n="Frame number0">Execution address;Elapsed time from measurement
start(ns);0,Current consumption value of peripheral function0(uA);1,Current consump-
tion value of peripheral function1(uA);...;</F>
   <F n="Frame number1">Execution address;Elapsed time from measurement
start(ns);0,Current consumption value of peripheral function0(uA);1,Current consump-
tion value of peripheral function1(uA);...;</F>
     :
 </Root>
```

Caution 1. The consumption current value is estimated from typical current value of the actual device. This value is approximated value as MCU alone. Not including the current values of other parts.

Caution 2. The maximum length of the consumption current calculation is 200,000 current changing points. If the changing current points reach maximum length, the user program execution is stopped.

[Example of use]

```
>>>debugger.CurrentConsumption.Get("C:/project/sample.xml")
Max = 1020.30, Average = 300.20
>>>
```

debugger.CurrentConsumption.Information

This function displays information on acquiring current consumption data. [RL78 (devices with support for peripheral function simulation)] [Simulator]

[Specification format]

```
debugger.CurrentConsumption.Information()
```

[Argument(s)]

None

[Return value]

If acquiring current consumption data was enabled: True
If acquiring current consumption data was disabled: False

[Detailed description]

- This function displays information on acquiring current consumption data.

[Example of use]

```
>>>debugger.CurrentConsumption.Information()
True
>>>
```

debugger.DebugTool.Change

This function changes the debug tool.

[Specification format]

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

[Argument(s)]

Argument	Description	
debugTool	Specify the debug tool to change. The debug tools that can be specified are shown below.	
	Type	Description
	DebugTool.Simulator	Simulator
	DebugTool.Minicube	MINICUBE
	DebugTool.Minicube2	MINICUBE2 (Serial connect)
	DebugTool.Minicube2Jtag	MINICUBE2 (JTAG connect)
	DebugTool.Iecube	IECUBE
	DebugTool.Iecube2	IECUBE2
	DebugTool.E1Jtag	E1 (JTAG connect)
	DebugTool.E1Serial	E1 (Serial connect)
	DebugTool.E1Lpd	E1 (LPD connect)
	DebugTool.E2	E2 emulator (abbreviated name: E2)
	DebugTool.E2Lite	E2 emulator Lite (abbreviated name: E2 Lite)
	DebugTool.E20Jtag	E20 (JTAG connect)
	DebugTool.E20Serial	E20 (Serial connect)
	DebugTool.E20Lpd	E20 (LPD connect)
	DebugTool.IE850A	IE850A
	DebugTool.ComPort	COM Port

[Return value]

If the debug tool was changed successfully: True
If there was an error when changing the debug tool: False

[Detailed description]

- This function changes the debug tool to the one specified by *DebugTool*.
  However, the debug tool that can be changed differs depending on the using device. Select [Debug Tool] on the project tree and select [Using Debug Tool] on the context menu. And then confirm the debug tool that can be changed.

Caution It is possible to specify non-selectable emulators. Only specify emulators that can be selected in CS+' s debugging tool.

[Example of use]

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

debugger.DebugTool.GetType

This function displays information about the debug tool.

[Specification format]

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

[Argument(s)]

None

[Return value]

Debug tool type

Type	Description
Simulator	Simulator
Minicube	MINICUBE
Minicube2	MINICUBE2 (Serial connect)
Minicube2Jtag	MINICUBE2 (JTAG connect)
Iecube	IECUBE
Iecube2	IECUBE2
E1Jtag	E1 (JTAG connect)
E1Serial	E1 (Serial connect)
E1Lpd	E1 (LPD connect)
E2	E2 emulator
E2Lite	E2 emulator Lite
E20Jtag	E20 (JTAG connect)
E20Serial	E20 (Serial connect)
E20Lpd	E20 (LPD connect)
IE850A	IE850A
ComPort	COM Port

[Detailed description]

- This function displays information about the debug tool.

[Example of use]

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

debugger.DebugTool.RestoreState

This function restores the state of the debug tool to the one saved in the file.

[Specification format]

```
debugger.DebugTool.RestoreState(fileName)
```

[Argument(s)]

Argument	Description
fileName	Specify the full path of the file to restore the state of the debug tool.

[Return value]

If the file was restored successfully: True
If there was an error when restoring the file: False

[Detailed description]

- This function restores the state of the debug tool to the one saved in the file.
  The state of the debug tool that can be restored is only in the file saved by the debugger.DebugTool.SaveState function.

[Example of use]

```
>>>debugger.DebugTool.SaveState("C:/test/debugtoolstate.log")
True
>>>debugger.DebugTool.RestoreState("C:/test/debugtoolstate.log")
True
>>>
```

debugger.DebugTool.SaveState

This function saves the state of the debug tool to a file.

[Specification format]

```
debugger.DebugTool.SaveState(fileName)
```

[Argument(s)]

Argument	Description
fileName	Specify the full path of the file to save the state of the debug tool.

[Return value]

If the file was saved successfully: True
If there was an error when saving the file: False

[Detailed description]

- This function saves readable/writable memory and register values to a file as the state of the debug tool.

[Example of use]

```
>>>debugger.DebugTool.SaveState("C:/test/debugtoolstate.log")
True
>>>
```

debugger.Disconnect

This function disconnects from the debug tool.

[Specification format]

```
debugger.Disconnect()
```

[Argument(s)]

None

[Return value]

If the debug tool was disconnected successfully: True
If there was an error when disconnecting from the debug tool: False

[Detailed description]

- This function disconnects from the debug tool.

[Example of use]

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

debugger.Download.Binary

This function downloads a binary file.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a download file.
address	Specify a download start address.
append	Specify whether to make an additional download. True: Perform additional download. False: Perform overwrite download (default).
flashErase	Specify whether to initialize a flash memory before download. True: Initialize a flash memory before download. False: Do not initialize a flash memory before download (default).

Caution It is not possible to specify only *fileName* and *address*.
When specifying both *fileName* and *address*, also specify *append* or both *append* and *flashErase*.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- This function downloads data in binary format.

[Example of use]

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

debugger.Download.Binary64Kb

This function downloads a binary file in within-64 KB format.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a download file.
address	Specify a download start address.
append	Specify whether to make an additional download. True: Perform additional download. False: Perform overwrite download (default).
flashErase	Specify whether to initialize a flash memory before download. True: Initialize a flash memory before download. False: Do not initialize a flash memory before download (default).

Caution It is not possible to specify only *fileName* and *address*.
When specifying both *fileName* and *address*, also specify *append* or both *append* and *flashErase*.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- When using the memory bank, this function downloads binary files in within-64 KB format.

[Example of use]

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

debugger.Download.BinaryBank

This function downloads a binary file in memory bank format.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a download file.
address	Specify a download start address.
append	Specify whether to make an additional download. True: Perform additional download. False: Perform overwrite download (default).
flashErase	Specify whether to initialize a flash memory before download. True: Initialize a flash memory before download. False: Do not initialize a flash memory before download (default).

Caution It is not possible to specify only *fileName* and *address*.
When specifying both *fileName* and *address*, also specify *append* or both *append* and *flashErase*.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- When using the memory bank, this function downloads binary files in memory bank format.

[Example of use]

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

debugger.Download.Coverage

This function downloads coverage data. [IECUBE][IECUBE2][Simulator]

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a coverage data file.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- This function downloads coverage data.

[Example of use]

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

debugger.Download.Hex

This function downloads a hex file.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a download file.
offset	Specify an offset (default: 0).
append	Specify whether to make an additional download. True: Perform additional download. False: Perform overwrite download (default).
flashErase	Specify whether to initialize a flash memory before download. True: Initialize a flash memory before download. False: Do not initialize a flash memory before download (default).

Caution It is not possible to specify only *fileName* and *offset*.
When specifying both *fileName* and *offset*, also specify *append* or both *append* and *flashErase*.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- This function downloads data in hex format.

[Example of use]

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

debugger.Download.Hex64Kb

This function downloads a hex file in within-64 KB format.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a download file.
offset	Specify an offset (default: 0).
append	Specify whether to make an additional download. True: Perform additional download. False: Perform overwrite download (default).
flashErase	Specify whether to initialize a flash memory before download. True: Initialize a flash memory before download. False: Do not initialize a flash memory before download (default).

Caution It is not possible to specify only *fileName* and *offset*.
When specifying both *fileName* and *offset*, also specify *append* or both *append* and *flashErase*.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- When using the memory bank, this function downloads hex files in within-64 KB format.

[Example of use]

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

debugger.Download.HexBank

This function downloads a hex file in memory bank format.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a download file.
offset	Specify an offset (default: 0).
append	Specify whether to make an additional download. True: Perform additional download. False: Perform overwrite download (default).
flashErase	Specify whether to initialize a flash memory before download. True: Initialize a flash memory before download. False: Do not initialize a flash memory before download (default).

Caution It is not possible to specify only *fileName* and *offset*.
When specifying both *fileName* and *offset*, also specify *append* or both *append* and *flashErase*.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- When using the memory bank, this function downloads hex files in memory-bank format.

[Example of use]

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

debugger.Download.HexIdTag

This function downloads a hex file with ID tag.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a download file.
offset	Specify an offset (default: 0).
append	Specify whether to make an additional download. True: Perform additional download. False: Perform overwrite download (default).
flashErase	Specify whether to initialize a flash memory before download. True: Initialize a flash memory before download. False: Do not initialize a flash memory before download (default).

Caution It is not possible to specify only *fileName* and *offset*.
When specifying both *fileName* and *offset*, also specify *append* or both *append* and *flashErase*.

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- This function downloads a hex file with ID tag.

[Example of use]

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

debugger.Download.Information

This function displays download information.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of download information (see the DownloadInfo property for detail)

[Detailed description]

- This function displays download information in the following format.

```
download-number: download-file-name
```

[Example of use]

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

debugger.Download.LoadModule

This function downloads a load module.

[Specification format]

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

[Argument(s)]

<table>
<tr><th>Argument</th><th colspan="2">Description</th></tr>
<tr><td>fileName</td><td colspan="2">Specify a download file.</td></tr>
<tr><td rowspan="5">downloadOption</td><td colspan="2">Specify an option.
The options that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>DownloadOption.NoSymbol</td><td>Do not load symbol information.</td></tr>
<tr><td>DownloadOption.SymbolOnly</td><td>Only load symbol information.</td></tr>
<tr><td>DownloadOption.Both</td><td>Load both symbol information and object information (default).</td></tr>
<tr><td>append</td><td colspan="2">Specify whether to make an additional download.
True: Perform additional download.
False: Perform overwrite download (default).</td></tr>
<tr><td>flashErase</td><td colspan="2">Specify whether to initialize a flash memory before download.
True: Initialize a flash memory before download.
False: Do not initialize a flash memory before download (default).</td></tr>
<tr><td rowspan="4">vendorType</td><td colspan="2">Specify the vendor of the compiler.
The types that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>VendorType.Auto</td><td>Automatically specify the vendor of the compiler judging from the output contents of debugging information (default).</td></tr>
<tr><td>VendorType.Ghs</td><td>Make this specification when using a compiler made by Green Hills Software, Inc.</td></tr>
</table>

[Return value]

If a binary file was downloaded successfully: True
If there was an error when downloading a binary file: False

[Detailed description]

- This function downloads a load module.
- If *fileName* is not specified, the file specified on the [Download File Settings] tab in the Property panel of the debugging tool is downloaded.
- If *downloadOption* is specified, the processing is performed in accordance with the specification.

[Example of use]

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

debugger.Erase

This function erases the flash memory.

[Specification format]

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

[Argument(s)]

Argument	Description	
eraseOption	Specify an option. The options that can be specified are shown below.	
	Type	Description
	EraseOption.Code	Erase the code flash memory (default).
	EraseOption.Data	Erase the data flash memory.
	EraseOption.External	Erase the flash memory in external space.

Caution EraseOption.External cannot be specified because CS+ does not support erasing flash memory data in the external space.

[Return value]

If the flash memory was erased successfully: True
If there was an error when erasing the flash memory: False

[Detailed description]

- This function erases the flash memory, specified by *eraseOption*.
- Erasing code-flash and data-flash data will be as shown in the table below.
  The simulator, on the other hand, always pads both code-flash and data-flash memory with 0xff.

Series	Emulator	Description
RL78	E1/E20/E2/E2 Lite	The operation leaves both code-flash and data-flash memory blank.
RL78	IECUBE	Code-flash memory is padded with 0xff and data-flash memory is left blank.
RX	E1/E20/E2/E2 Lite	Both code-flash and data-flash memory are padded with 0xff.
RH850	E1/E20/E2/Full-spec emulator/IE850A	The operation leaves both code-flash and data-flash memory blank.

[Example of use]

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

debugger.GetBreakStatus

This function displays a break condition.

[Specification format]

```
debugger.GetBreakStatus()
```

[Argument(s)]

None

[Return value]

Break-trigger string (See [Detailed description])

Remark 1. Returns the string portion of the "BreakStatus" enum.

Remark 2. Determine conditions by writing in the format "BreakStatus.*string*".

[Detailed description]

- This function displays break-trigger.
  During execution, this will be "None".

Break-trigger String	Description	78K0			RL78,78K0R			V850			
		Iecube	Minicube2 Note 1	Simulator	Iecube	Minicube2 Note 1	Simulator	Iecube	Minicube Note 2	Minicube2 Note 1	Simulator
None	No break	o	o	-	o	o	-	o	o	o	-
Manual	Forced break	o	o	o	o	o	o	o	o	o	o
Event	Break due to event	o	o	o	o	o	o	o	o	o	o
Software	Software break	o	o	-	o	o	-	o	o	o	-
TraceFull	Break due to trace full	o	-	o	o	-	o	o	-	-	o
TraceDelay	Break due to trace delay	o	-	-	o	-	-	-	-	-	-
NonMap	Access to non-mapped area	o	-	o	o	-	o	o	-	-	o
WriteProtect	Write to write-protected area	o	-	o	o	-	o	o	-	-	o
ReadProtect	Read from read-protected area	o	-	-	-	-	-	-	-	-	-
SfrIllegal	Illegal SFR access	o	-	-	-	-	-	-	-	-	-
SfrReadProtect	Read from non-readable SFR	o	-	-	o	-	-	-	-	-	-
SfrWriteProtect	Write to non-writable SFR	o	-	-	o	-	-	-	-	-	-
IorIllegal	Illegal access to peripheral I/O register (with address)	-	-	-	-	-	-	o	-	-	-

Break-trigger String	Description	78K0			RL78,78K0R			V850			
		Iecube	Minicube2[Note 1]	Simulator	Iecube	Minicube2[Note 1]	Simulator	Iecube	Minicube[Note 2]	Minicube2[Note 1]	Simulator
StackOverflow	Break due to stack overflow	o	-	-	o	-	-	-	-	-	-
StackUnderflow	Break due to stack underflow	o	-	-	o	-	-	-	-	-	-
UninitializeStackPointer	Break due to uninitialized stack pointer	o	-	-	o	-	-	-	-	-	-
UninitializeMemoryRead	Read uninitialized memory	o	-	-	o	-	-	-	-	-	-
TimerOver	Execution timeout detected	o	-	-	o	-	-	o	-	-	-
UnspecifiedIllegal	Illegal operation in user program relating to peripheral chip features	o	-	-	o	-	-	-	-	-	-
ImsIxsIllegal	Break due to illegal write to IMS/IXS register	o	-	-	-	-	-	-	-	-	-
BeforeExecution	Pre-execution break	o	-	-	o	-	-	-	-	-	-
SecurityProtect	Accessed security-protected region	-	-	-	-	-	-	-	-	-	-
FlashMacroService	Flash macro service active	-	-	-	-	-	-	-	o	o	-
RetryOver	Number of retries exceeded limit	o	-	-	-	-	-	-	-	-	-
FlashIllegal	Illegal Flash break	o	-	-	o	-	-	-	-	-	-
Peripheral	Break from peripheral	o	-	-	o	-	-	-	-	-	-
WordMissAlignAccess	Word access to odd address	-	-	-	o	-	o	-	-	-	-
Temporary	Temporary break	o	o	o	o	o	o	o	o	o	o
Escape	Escape break	-	-	-	-	-	-	o	o	o	-
Fetch	Fetched from guard area or area where fetches are prohibited	o	-	-	o	-	-	-	-	-	-
IRamWriteProtect	Wrote to IRAM guard area (with address)[Note 3]	-	-	-	-	-	-	o	-	-	-
IllegalOpcodeTrap	Break due to illegal instruction exception	-	-	-	-	-	-	o	Δ Note 6	-	-
Step	Step execution break[Note 4]	o	o	o	o	o	o	-	-	-	o
FetchGuard	Fetch guard break[Note 4]	o	-	-	o	-	-	-	-	-	-
TraceStop	Trace stop[Note 4]	o	-	-	o	-	-	-	-	-	-
ExecutionFails	Execution failed[Note 5]	o	o	-	o	o	-	o	o	o	-
CurrentConsumptionFullBreak	Full of the current consumption buffer	-	-	-	-	-	o	-	-	-	-

Break-trigger String	Description	78K0			RL78,78K0R			V850			
		Iecube	Minicube2[Note 1]	Simulator	Iecube	Minicube2[Note 1]	Simulator	Iecube	Minicube[Note 2]	Minicube2[Note 1]	Simulator
CurrentConsumptionTimeBreak	Current consumption time break	-	-	-	-	-	o	-	-	-	-
ExpansionFunctionAction	E2 expansion function action break	-	-	-	-	o Note 7	-	-	-	-	-
ExpansionFunctionStorageFull	Fully used the storage memory break	-	-	-	-	o Note 7	-	-	-	-	-

Note 1. Applies to all of the following: MINICUBE2, E1Serial, E20Serial, E2, and E2Lite.

Note 2. Applies to all of the following: MINICUBE, E1Jtag, E20Jtag, and MINICUBE2Jtag.

Note 3. Performed a verification check on the IRAM guard area during break, and the value was overwritten (if this affects multiple addresses, only the first address is shown).

Note 4. This is only a break cause during trace.

Note 5. This is only a break cause during a break.

Note 6. Not displayed with V850-MINICUBE on V850E/ME2, etc. (same core) when a post-execution event is used.

Note 7. Only applicable when CS+ for CC and the E2 emulator are in use.

Break-trigger String	Description	RX		V850E2				RH850	
		E1Jtag, E1Serial E20Jtag, E20Serial, E2, E2 Lite	Simulator	Iecube2	Minicube[Note 2]	Minicube2[Note 1]	Simulator	E1/E20/E2/Full-spec emulator/IE850A	SIM
None	No break	o	-	o	o	o	-	-	-
Manual	Forced break	o	o	o	o	o	o	o	o
Event	Break due to event	o	o	o	o	o	o	o	o
Software	Software break	o	-	o	o	o	-	o	-
TraceFull	Break due to trace full	o	o	o	-	-	o	o	o
NonMap	Access to non-mapped area	-	-	-	-	-	o	-	o
WriteProtect	Write to write-protected area	-	-	-	-	-	o	-	o
TimerOver	Execution timeout detected	-	-	o	o	-	-	-	-

Break-trigger String	Description	RX		V850E2				RH850	
		E1Jtag, E1Serial E20Jtag, E20Serial, E2, E2 Lite	Simulator	Iecube2	Minicube[Note 2]	Minicube2[Note 1]	Simulator	E1/E20/E2/Full-spec emulator/IE850A	SIM
FlashMacroService	Flash macro service active	-	-	o	o	o	-	-	-
Temporary	Temporary break	o	o	o	o	o	o	o	o
IllegalOpcodeTrap	Break due to illegal instruction exception	-	-	o	o	-	-	-	-
Step	Step execution break[Note 3]	o	-	-	-	-	o	o	o
ExecutionFails	Execution failed[Note 4]	o	-	o	o	o	-	-	-
WaitInstruction	Break caused by executing WAIT instruction	-	o	-	-	-	-	-	-
UndefinedInstruction-Exception	Break caused by undefined instruction exception	-	o	-	-	-	-	-	-
PrivilegeInstructionException	Break caused by privileged instruction exception	-	o	-	-	-	-	-	-
AccessException	Break caused by access exception	-	o	-	-	-	-	-	-
FloatingPointException	Break caused by floating point exception	-	o	-	-	-	-	-	-
InterruptException	Break caused by interrupt	-	o	-	-	-	-	-	-
IntInstructionException	Break caused by INT instruction exception	-	o	-	-	-	-	-	-
BrkInstructionException	Break caused by BRK instruction exception	-	o	-	-	-	-	-	-
IOFunctionSimulation-Break	Break caused by peripheral function simulation	-	o	-	-	-	-	-	-
IllegalMemoryAccess-Break	Break caused by illegal memory access	-	o	-	-	-	-	-	-
StreamIoError	Break caused by stream I/O error	-	o	-	-	-	-	-	-
CoverageMemoryAllocationFailure	Failed to allocate coverage memory	-	o	-	-	-	-	-	-
TraceMemoryAllocationFailure	Failed to allocate trace memory	-	o	-	-	-	-	-	-
StepCountOver	Step count over	-	-	-	-	-	-	o	o
DebuggingInformation-AcquisitionFailure	Failed to acquire debugging information	-	-	-	-	-	-	o	o

Break-trigger String	Description	RX		V850E2				RH850	
		E1Jtag, E1Serial E20Jtag, E20Serial, E2, E2 Lite	Simulator	Iecube2	Minicube[Note 2]	Minicube2[Note 1]	Simulator	E1/E20/E2/Full-spec emulator/IE850A	SIM
RelayForTrace	An occurrence of Relay Break (only trace)	-	-	-	-	-	-	-	o
ExpansionFunctionAc-tion	E2 expansion function action break	o Note 5	-	-	-	-	-	o Note 5	-
ExpansionFunction-StorageFull	Fully used the storage memory break	o Note 5	-	-	-	-	-	o Note 5	-
SoftwareTraceLpdFull	Fully used the storage memory at LPD output of software trace	-	-	-	-	-	-	o Note 5	-

Note 1. Applies to all of the following: MINICUBE2, E1Serial, and E20Serial.

Note 2. Applies to all of the following: MINICUBE, E1Jtag, E20Jtag, and MINICUBE2Jtag.

Note 3. This is only a break cause during trace.

Note 4. This is only a break cause during a break.

Note 5. Applies to E2.

[Example of use]

```
>>>debugger.GetBreakStatus()
Temporary
>>>a = debugger.GetBreakStatus()
Temporary
>>>print a
Temporary
>>>if (debugger.GetBreakStatus() == BreakStatus.Temporary):
... print "Temporary break"
...
Temporary
Temporary break
>>>
```

debugger.GetCpuStatus

This function displays the current CPU status.

[Specification format]

```
debugger.GetCpuStatus()
```

[Argument(s)]

None

[Return value]

Current CPU status (string)

CPU Status	Description
Hold	In bus hold
HoldStopIdle	Bus hold/Software STOP/Hardware STOP/IDLE mode
PowOff	Power not supplied to the target
InitialStop	Initial stop
Reset	In reset state
Standby	GTM: Clock is not supplied Other than GTM: In standby mode
Stop	In STOP mode
StopIdle	Software STOP/Hardware STOP/IDLE mode
Wait	In wait state
Halt	In HALT mode
Sleep	In sleep state
DeepStop	In Deep Stop mode
CyclicRun	In Cyclic Run mode
CyclicStop	In Cyclic Stop mode
CyclicDisable	This is the state of the core other than the main core when the main core is in the Cyclic Run or Cyclic Stop mode.
Disable	MCS of GTM is not running.
None	N/A

[Detailed description]

- This function displays the current CPU status.

[Example of use]

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

debugger.GetIeStatus

This function displays the current IE status.

[Specification format]

```
debugger.GetIeStatus()
```

[Argument(s)]

None

[Return value]

Current IE status (string)

IE Status	Description
Break	Break in effect
Coverage	Coverage running
Timer	Timer running
Tracer	Trace running
Step	Step executing
Run	User program running
RunOrStep	User program running or step executing

Caution If a PM+ workspace is converted to a CS+ project, then there will be no debugging tool in the main project. For this reason, "None" will be returned if the main project is the active project.
In addition, "None" will be returned before the debugging tool is connected.

[Detailed description]

- This function displays the current IE status.

[Example of use]

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

debugger.GetIORList

This function displays a list of the IORs and SFRs.

[Specification format]

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

[Argument(s)]

Argument	Description
category	Specify the category in which IORs and SFRs are defined (default: not specified).

[Return value]

List of IOR and SFR information (see the IORInfo class for detail)

[Detailed description]

- This function displays a list of the IORs and SFRs of the active project.
- This function displays a list of the IORs and SFRs defined in *category*.
- If *category* is not specified, a list of all IORs and SFRs.
- This function displays a list of the IORs and SFRs in the following format.

```
IOR-or-SFR-name value type size address
```

[Example of use]

```
>>> 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
>>> debugger.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

This function displays the PC value.

[Specification format]

```
debugger.GetPC()
```

[Argument(s)]

None

[Return value]

PC value (numeric value)

[Detailed description]

- This function displays the PC value.

[Example of use]

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

debugger.GetProcessorElementNames

This function displays a list of the PE names of multiple cores.

[Specification format]

```
debugger.GetProcessorElementNames()
```

[Argument(s)]

None

[Return value]

Array (strings) containing the PE names of multiple cores

[Detailed description]

- This function displays a list of the PE names of multiple cores.
- Display a list of PE names that can be set in debugger.ProcessorElementName.

Caution A debug tool must be connected at the time this function is executed.

[Example of use]

```
>>a = debugger.GetProcessorElementNames()
CPU1
CPU2
>>print a
['CPU1', 'CPU2']
```

debugger.Go

This function continues program execution.

[Specification format]

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

[Argument(s)]

Argument	Description	
goOption	Specify an option. The options that can be specified are shown below.	
	Type	Description
	GoOption.IgnoreBreak	Execute ignoring breakpoints.
	GoOption.WaitBreak	Wait until program stops.
	GoOption.Normal	Breakpoints enabled; do not wait until program stops (default).

[Return value]

None

[Detailed description]

- This function continues program execution.
- If *goOption* is specified, the processing is performed in accordance with the specification.

[Example of use]

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

debugger.Ie.GetValue
debugger.Ie.SetValue

This function sets or refers to the IE register or DCU register.

[Specification format]

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

[Argument(s)]

Argument	Description	
ieType	Specify a register. The registers that can be specified are shown below.	
	Type	Description
	IeType.Reg	IE register [[IECUBE] 78K0] [[IECUBE] RL78] [[IECUBE] 78K0R] [[IECUBE] V850] [[IECUBE2] V850] [[Full-spec emulator] RH850]
	IeType.Dcu	DCU register [[IECUBE] V850] [[IECUBE2] V850] [[Full-spec emulator/E1/E20] RH850]
address	Specify the address to reference/set.	
value	Specify the setting value.	

[Return value]

debugger.Ie.GetValue is the register value (numeric value)
debugger.Ie.SetValue is True if the setting was completed successfully, or False if there was an error when setting the register.

[Detailed description]

- debugger.Ie.GetValue displays the value of the register specified by *address*.
  The register type is specified by *ieType*.
- debugger.Ie.SetValue writes *value* to the register specified by *address*.
  The register type is specified by *ieType*.

Remark When the DCU register is referenced, the register value is reset to 0.

[Example of use]

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

debugger.Interrupt.DeleteTimer

This function deletes the timer interrupt setting. [RH850 Simulator]

Remark This function provides the same function as debugger.XRunBreak.Delete.

[Specification format]

```
debugger.Interrupt.DeleteTimer()
```

[Argument(s)]

None

[Return value]

If the timer interrupt setting was deleted successfully: True
If there was an error when deleting the timer interrupt setting: False

[Detailed description]

- This function deletes the timer interrupt setting.

[Example of use]

```
>>>debugger.Interrupt.ReferTimer()
None
>>>debugger.Interrupt.SetTimer(1, TimeType.S, True)
True
>>>debugger.Interrupt.ReferTimer()
1Second Periodic
>>>debugger.Interrupt.DeleteTimer()
True
>>>debugger.Interrupt.ReferTimer()
None
```

debugger.Interrupt.Notification

This function sets exception cause codes whose notification is accepted. [RH850 Simulator]

[Specification format]

```
debugger.Interrupt.Notification(notificationMode = NotificationMode.Deny, code)
```

[Argument(s)]

Argument	Description	
notificationMode	Specify the mode for accepting notification of exception cause codes. The modes that can be specified are shown below.	
	Type	Description
	NotificationMode.Deny	Deny notification of all exception cause codes (default).
	NotificationMode.Allow	Allow notification of all exception cause codes.
code	Specify the list of exception cause codes whose notification is accepted (numerical value).	

[Return value]

If exception cause codes were set successfully: True
If there was an error when setting exception cause codes: False

[Detailed description]

- This function sets exception cause codes whose notification is accepted.
- To accept notification of only specific exception cause codes, specify NotificationMode.Deny in *notificationMode* and specify the exception cause codes whose notification is accepted in *code*.
  To deny notification of only specific exception cause codes, specify NotificationMode.Allow in *notificationMode* and specify the exception cause codes whose notification is denied in *code*.
  When this function is used, all exception cause codes that have been previously set are discarded.
- Define the processing to be performed after accepting a specified exception cause code in a hook function or callback function. See “Hook“ for detail.

[Example of use]

```
>>>expcode = [0x00000020, 0x00000030, 0x00000050]
>>>debugger.Interrupt.Notification(NotificationMode.Deny, expcode)
True
>>>
```

debugger.Interrupt.OccurEI

This function generates EI-level interrupts.[RH850 Simulator] [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH, RH850G4MH (versions earlier than 2.0)]

[Specification format]

```
debugger.Interrupt.OccurEI(channel, priority, eiVectorType = EIVectorType.Standard)
```

[Argument(s)]

Argument	Description	
channel	Specify an interrupt name (string) or vector address (numerical value).	
priority	Specify the interrupt priority as a numerical value (0 to 15).	
eiVectorType	Specify the interrupt vector mode. The modes that can be specified are shown below.	
	Type	Description
	EIVectorType.Standard	Standard mode (default)
	EIVectorType.Expanded	Expanded mode

[Return value]

If interrupts were generated successfully: True
If there was an error when generating interrupts: False

[Detailed description]

- This function generates EI-level interrupts.
- Specify the interrupt name to be generated in *channel* and the priority in *priority*.
  Specify *eiVectorType* according to the interrupt vector mode in use.

[Example of use]

```
>>>debugger.Interrupt.OccurEI(0x20, 1, EIVectorType.Standard)
True
>>>
```

debugger.Interrupt.OccurFE

This function generates FE-level interrupts. [RH850 Simulator] [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH, RH850G4MH (versions earlier than 2.0)]

This function generates FE-level interrupts caused by a SYSERR interrupt. [RH850 Simulator] [RH850G4MH (version 2.0 or later)]

[Specification format]

```
debugger.Interrupt.OccurFE(feVectorType, isGuestMode = false, gpid = None)
```

[Argument(s)]

Argument	Description	
feVectorType	Specify the type of an interrupt. The types that can be specified are shown below.	
	Type	Description
	FEVectorType.FENMI	NMI interrupt
	FEVectorType.FEINT	INT interrupt [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH]
	FEVectorType.FEINT0	INT interrupt 0 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT1	INT interrupt 1 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT2	INT interrupt 2 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT3	INT interrupt 3 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT4	INT interrupt 4 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT5	INT interrupt 5 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT6	INT interrupt 6 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT7	INT interrupt 7 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT8	INT interrupt 8 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT9	INT interrupt 9 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT10	INT interrupt 10 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT11	INT interrupt 11 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT12	INT interrupt 12 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT13	INT interrupt 13 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT14	INT interrupt 14 [RH850G4MH (versions earlier than 2.0)]
	FEVectorType.FEINT15	INT interrupt 15 [RH850G4MH (versions earlier than 2.0)]

Argument	Description	
	FEVectorType.SyserrCause10	SyserrCause10 interrupt
	FEVectorType.SyserrCause11	SyserrCause11 interrupt
	FEVectorType.SyserrCause12	SyserrCause12 interrupt
	FEVectorType.SyserrCause13	SyserrCause13 interrupt
	FEVectorType.SyserrCause14	SyserrCause14 interrupt
	FEVectorType.SyserrCause15	SyserrCause15 interrupt
	FEVectorType.SyserrCause16	SyserrCause16 interrupt
	FEVectorType.SyserrCause17	SyserrCause17 interrupt
	FEVectorType.SyserrCause18	SyserrCause18 interrupt
	FEVectorType.SyserrCause19	SyserrCause19 interrupt
	FEVectorType.SyserrCause1a	SyserrCause1a interrupt
	FEVectorType.SyserrCause1b	SyserrCause1b interrupt
	FEVectorType.SyserrCause1c	SyserrCause1c interrupt
	FEVectorType.SyserrCause1d	SyserrCause1d interrupt
	FEVectorType.SyserrCause1e	SyserrCause1e interrupt
	FEVectorType.SyserrCause1f	SyserrCause1f interrupt
isGuestMode	Specifies whether the device context that raises the exception is in guest mode. True : Guest mode False : Host mode or conventional mode (default)	
gpid	Specify the GPID of the device context that causes the exception. None: : Do not specify GPID (default) (numeric value) : GPID of the device context that raises the exception.	

[Return value]

If interrupts were generated successfully: True
If there was an error when generating interrupts: False

[Detailed description]

- This function generates FE-level interrupts.
- Specify the interrupt name to be generated in *feVectorType*.

[Example of use]

```
>>>debugger.Interrupt.OccurFE(FEVectorType.FENMI)
True
>>>
```

debugger.Interrupt.PseudoInterrupt

This function generates pseudo-interrupts. [RL78 instruction simulator]

[Specification format]

```
debugger.Interrupt.PseudoInterrupt(vectorAddress, priority)
```

[Argument(s)]

Argument	Description
vectorAddress	Specify the interrupt name (string) or vector address (numerical value: 0x4 to 0x7c).
priority	Specify the interrupt priority (numerical value: 0 to 3).

[Return value]

If interrupts were generated successfully : True
If there was an error when generating interrupts : False

[Detailed description]

- An interrupt is generated in the interrupt enable (EI) state.
- An interrupt is reserved in the interrupt disable (DI) state and generated when the state next becomes interrupt enable (EI).
- Setting registers for the interrupt functions is not required. Among such registers, only the PSW is changed by an interrupt.
- If a reset is generated while an interrupt is held pending, this interrupt will be deleted.

[Example of use]

```
>>>debugger.Interrupt.PseudoInterrupt(8, 0)
True
>>>debugger.Interrupt.PseudoInterrupt("INTWDTI", 1)
True
>>>
```

debugger.Interrupt.ReferTimer

This function displays the timer interrupt setting information. [RH850 Simulator]

Remark This function provides the same function as debugger.XRunBreak.Refer.

[Specification format]

```
debugger.Interrupt.ReferTimer()
```

[Argument(s)]

None

[Return value]

List of period time value and period information (TimeType) (see the XRunBreakInfo property for detail)

[Detailed description]

- This function displays the periodic information (periodic time [Periodic]) of the timer interrupt that is set.
- If there is no timer interrupt setting, "None" is displayed.

[Example of use]

```
>>>debugger.Interrupt.ReferTimer()
None
>>>debugger.Interrupt.SetTimer(1, TimeType.S, True)
True
>>>debugger.Interrupt.ReferTimer()
1Second Periodic
```

debugger.Interrupt.RequestEI

This function sends an EI-level interrupt request to the interrupt controller. [RH850 Simulator (G4MH)]

[Specification format]

```
debugger.Interrupt.RequestEI(channel)
```

[Argument(s)]

Argument	Description
channel	Specify an interrupt name (string) or vector address (numeric value).

[Return value]

If the interrupt request was successful : True
If the interrupt request failed : False

[Detailed description]

- This function sends an EI-level interrupt request to the interrupt controller.
- Generation of the interrupt depends on the settings and state of the interrupt controller.

[Example of use]

```
>>>debugger.Interrupt.RequestEI(1)
True
>>>
```

debugger.Interrupt.RequestFE

This function sends an FE-level interrupt request to the interrupt controller. [RH850 Simulator (G4MH)]

[Specification format]

```
debugger.Interrupt.RequestFE(channelNumber)
```

[Argument(s)]

Argument	Description
channelNumber	Specify the channel number of the FE-level interrupt.

[Return value]

If the interrupt request was successful : True
If the interrupt request failed : False

[Detailed description]

- This function sends an FE-level interrupt request to the interrupt controller.
- Generation of the interrupt depends on the settings and state of the interrupt controller.

[Example of use]

```
>>>debugger.Interrupt.RequestFE(0)
True
>>>
```

debugger.Interrupt.RequestFENMI

This function sends an NMI request to the interrupt controller. [RH850 Simulator (G4MH)]

[Specification format]

```
debugger.Interrupt.RequestFENMI()
```

[Argument(s)]

None

[Return value]

If the interrupt request was successful : True
If the interrupt request failed : False

[Detailed description]

- This function sends an NMI request to the interrupt controller.
- Generation of the interrupt depends on the settings and state of the interrupt controller.

[Example of use]

```
>>>debugger.Interrupt.RequestFENMI()
True
>>>
```

debugger.Interrupt.SetTimer

This function sets the timer interrupt. [RH850 Simulator]

Remark This function provides the same function as debugger.XRunBreak.Set.

[Specification format]

```
debugger.Interrupt.SetTimer(time, timeType = TimeType.Ms, periodic = False)
```

[Argument(s)]

Argument	Description	
time	Specify the break time.	
timeType	Specify the break time unit. The units that can be specified are shown below.	
	Type	Description
	TimeType.Min	Minute unit
	TimeType.S	Second unit
	TimeType.Ms	Millisecond unit (default)
	TimeType.Us	Microsecond unit
	TimeType.Ns	Nanosecond unit
periodic	Specify whether to call the callback every time the specified time elapses. True: Call at every specified time interval. False: Call one time only (default).	

[Return value]

If the timer interrupt was set successfully: True
If there was an error when setting the timer interrupt: False

[Detailed description]

- This function sets the timer interrupt.
- The calling interval of a timer interrupt depends on the simulator.
- Register the Python function that is processed after the specified time passes. See “Hook“ for detail.

[Example of use]

```
>>>debugger.Interrupt.ReferTimer()
None
>>>debugger.Interrupt.SetTimer(1, TimeType.S, True)
True
>>>debugger.Interrupt.ReferTimer()
1Second Periodic
```

debugger.IsConnected

This function checks the connection status of the debug tool.

[Specification format]

```
debugger.IsConnected()
```

[Argument(s)]

None

[Return value]

If the debug tool is connected: True
If the debug tool is not connected: False

[Detailed description]

- This function checks the connection status of the debug tool.

[Example of use]

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

debugger.IsRunning

This function checks the execution status of the user program.

[Specification format]

```
debugger.IsRunning()
```

[Argument(s)]

None

[Return value]

If the user program is running: True
If the user program is not running: False

[Detailed description]

- This function checks the execution status of the user program.

[Example of use]

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

debugger.Jump.File
debugger.Jump.Address

This function displays each panel.

[Specification format]

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

[Argument(s)]

Argument	Description	
fileName	Specify the name of the file to display.	
lineNumber	Specify the line to display (default: 1).	
jumpType	Specify the type of panel to display. The panel types that can be specified are shown below.	
	Type	Description
	JumpType.Source	Editor panel
	JumpType.Assemble	Disassemble panel
	JumpType.Memory	Memory panel
address	Specify the address to display (default: 0).	

[Return value]

None

[Detailed description]

- debugger.Jump.File displays the file specified by *fileName* in the Editor panel.
  If *lineNumber* is specified, then the line specified by *lineNumber* in the file specified by *fileName* is displayed.
- debugger.Jump.Address displays the panel specified by *jumpType*.
  If *address* is specified, then the area corresponding to the specified address is displayed.

[Example of use]

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

debugger.Map.Clear

This function clears the mapping settings.

[Specification format]

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

[Argument(s)]

None

[Return value]

If the memory map was cleared successfully: True
If there was an error when clearing the memory map: False

[Detailed description]

- This function clears the mapping settings.

[Example of use]

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

debugger.Map.Information

This function displays map information.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of map information (see the MapInfo class for detail)

[Detailed description]

- This function displays map information.

```
number: start-address end-address access-size memory-type
```

[Example of use]

```
>>>debugger.Map.Information()
1: 0x00000000 0x0005FFFF 32 (Internal ROM area)
2: 0x00060000 0x03FF6FFF 8 (Non map area)
3: 0x03FF7000 0x03FFEFFF 32 (Internal RAM area)
4: 0x03FFF000 0x03FFFFFF 8 (SFR)
>>>
```

debugger.Map.Set

This function configures memory mapping.

[Specification format]

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

[Argument(s)]

<table>
<tr><th>Argument</th><th colspan="2">Description</th></tr>
<tr><td>mapType</td><td colspan="2">Specify a memory type.
The memory types that can be specified are shown below.</td></tr>
<tr><td></td><td>Type</td><td>Description</td></tr>
<tr><td></td><td>MapType.EmulationRom</td><td>Emulation ROM area</td></tr>
<tr><td></td><td>MapType.EmulationRam</td><td>Emulation RAM area</td></tr>
<tr><td></td><td>MapType.Target</td><td>Target memory area</td></tr>
<tr><td></td><td>MapType.TargetRom</td><td>Target ROM area</td></tr>
<tr><td></td><td>MapType.Stack</td><td>Stack area</td></tr>
<tr><td></td><td>MapType.Protect</td><td>I/O protect area</td></tr>
<tr><td>address1</td><td colspan="2">Specify a map start address.</td></tr>
<tr><td>address2</td><td colspan="2">Specify a map end address.</td></tr>
<tr><td>accessSize</td><td colspan="2">Specify an access size (bit) (default: 8).
For V850, specify either 8, 16, or 32.
For 78K0R [IECUBE], specify either 8 or 16.</td></tr>
<tr><td>cs</td><td colspan="2">Specify the chip select (default: not specified).
When mapping emulation memory (alternative ROM/RAM) in the IECUBE [V850E1], specify the one of the following chip selects as a string: cs0, cs1, cs2, cs3, cs4, cs5, cs6, or cs7.
For models in the V850ES series, however, the chip select allocation is fixed, or the chip select will not function, so this can be omitted.
If chip select is specified, then accessSize cannot be omitted.</td></tr>
</table>

[Return value]

If memory mapping was configured successfully: True
If there was an error when configuring memory mapping: False

[Detailed description]

- This function configures memory mapping with the memory type specified by *mapType*.

[Example of use]

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

debugger.Memory.Copy

This function copies the memory.

[Specification format]

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

[Argument(s)]

Argument	Description
address1	Specify the start address to copy from.
address2	Specify the end address to copy from.
address3	Specify the address to copy to.

[Return value]

If the memory was copied successfully: True
If there was an error when copying the memory: False

[Detailed description]

- This function copies the memory from *address1* to *address2* into *address3*.

[Example of use]

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

debugger.Memory.Fill

This function fills the memory.

[Specification format]

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

[Argument(s)]

Argument	Description	
address1	Specify the start address to fill.	
address2	Specify the end address to fill to.	
value	Specify the fill value.	
memoryOption	Specify the fill unit. The units that can be specified are shown below.	
	Type	Description
	MemoryOption.Byte	Byte unit (8 bits) (default)
	MemoryOption.HalfWord	Half-word unit (16 bits) [RH850,RX,V850]
	MemoryOption.Word	Word unit (RL78,78K: 16 bits, RH850,RX,V850: 32 bits)

[Return value]

If the memory was filled successfully: True
If there was an error when filling the memory: False

[Detailed description]

- This function fills from *address1* to *address2* with *value*.
- If *memoryOption* is specified, fill according to that specification.

[Example of use]

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

debugger.Memory.Read

This function refers to the memory.

[Specification format]

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

[Argument(s)]

Argument	Description	
address	Specify the address to reference.	
memoryOption	Specify the display unit. The units that can be specified are shown below.	
	Type	Description
	MemoryOption.Byte	Byte unit (8 bits) (default)
	MemoryOption.HalfWord	Half-word unit (16 bits) [RH850,RX,V850]
	MemoryOption.Word	Word unit (RL78,78K: 16 bits, RH850,RX,V850: 32 bits)

[Return value]

Referenced memory value (numeric value)

[Detailed description]

- This function displays the address specified by *address*, according to *memoryOption* in hexadecimal format.
- When multiple values are to be read from consecutive addresses, using debugger.Memory.ReadRange reduces the overhead of processing for reading.

[Example of use]

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

debugger.Memory.ReadRange

This function refers to the specified number of locations in memory.

[Specification format]

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

[Argument(s)]

Argument	Description	
address	Specify the start address to reference.	
count	Specify the number of locations in memory for reference.	
memoryOption	Specify the display unit. The units that can be specified are shown below.	
	Type	Description
	MemoryOption.Byte	Byte unit (8 bits) (default)
	MemoryOption.HalfWord	Half-word unit (16 bits) [RH850,RX,V850]
	MemoryOption.Word	Word unit (RL78,78K: 16 bits, RH850,RX,V850: 32 bits)

[Return value]

List of referenced memory value (numeric value)

[Detailed description]

- This function displays, in hexadecimal notation, the number of values specified by *count* with the width in memory specified by *memoryOption* in the range from the address specified by *address*.
- In case of failure to acquire a value from memory, "?" is displayed (0x??, 0x????, and 0x???????? in the 8-, 16-, and 32-bit cases, respectively).

[Example of use]

```
>>>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[0]
35
```

debugger.Memory.Write

This function writes to the memory.

[Specification format]

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

[Argument(s)]

Argument	Description	
address	Specify the address to set.	
value	Specify the value to set.	
memoryOption	Specify the unit to set. The units that can be specified are shown below.	
	Type	Description
	MemoryOption.Byte	Byte unit (8 bits) (default)
	MemoryOption.HalfWord	Half-word unit (16 bits) [RH850,RX,V850]
	MemoryOption.Word	Word unit (RL78,78K: 16 bits, RH850,RX,V850: 32 bits)

[Return value]

If the memory was written to successfully: True
If there was an error when writing to the memory: False

[Detailed description]

- This function sets the value at the address specified by *address*, according to *memoryOption*.
- When multiple values are to be written to consecutive addresses, using debugger.Memory.WriteRange reduces the overhead of processing for writing.

[Example of use]

```
>>>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

This function writes multiple data to the memory.

[Specification format]

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

[Argument(s)]

Argument	Description	
address	Specify the start address to write.	
valuelist	Specify the list of the value to set.	
memoryOption	Specify the unit to set. The units that can be specified are shown below.	
	Type	Description
	MemoryOption.Byte	Byte unit (8 bits) (default)
	MemoryOption.HalfWord	Half-word unit (16 bits) [RH850,RX,V850]
	MemoryOption.Word	Word unit (RL78,78K: 16 bits, RH850,RX,V850: 32 bits)

[Return value]

If the memory was written to successfully: True
If there was an error when writing to the memory: False

[Detailed description]

- This function writes, in accord with the setting of *memoryOption*, the list of values specified by *valuelist* to the address range starting at the address specified by *address*.

[Example of use]

```
>>> 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

This function performs procedure step execution.

[Specification format]

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

[Argument(s)]

Argument	Description	
nextOption	Specify the execution unit. The units that can be specified are shown below.	
	Type	Description
	NextOption.Source	Source-line unit (default)
	NextOption.Instruction	Instruction unit

[Return value]

None

[Detailed description]

- This function performs procedure step execution.
  If a function call is being performed, then stop after the function executes.

[Example of use]

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

debugger.Performance.Delete

This function deletes a condition of performance measurement. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.Performance.Delete(performanceNumber = "")
```

[Argument(s)]

Argument	Description
performanceNumber	Specify the performance measurement event number to delete.

[Return value]

If the condition of performance measurement event was deleted successfully: True
If there was an error when deleting the condition of performance measurement event: False

[Detailed description]

- This function deletes the condition of the performance measurement event number specified by *performanceNumber*.
- If *performanceNumber* is not specified, then conditions of all performance measurement event numbers will be deleted.

[Example of use]

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

debugger.Performance.Disable

This function disables performance measurement. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.Performance.Disable(performanceNumber = "")
```

[Argument(s)]

Argument	Description
performanceNumber	Specify the performance measurement event number to disable.

[Return value]

If performance measurement was disabled successfully: True
If there was an error when disabling performance measurement: False

[Detailed description]

- This function disables performance measurement of a performance measurement event number specified by *performanceNumber*.
- If *performanceNumber* is not specified, then performance measurement of all performance measurement event numbers will be disabled.

[Example of use]

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

debugger.Performance.Enable

This function enables performance measurement. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.Performance.Enable(performanceNumber = "")
```

[Argument(s)]

Argument	Description
performanceNumber	Specify the performance measurement event number to enable.

[Return value]

If performance measurement was enabled successfully: True
If there was an error when enabling performance measurement: False

[Detailed description]

- This function enables performance measurement of a performance measurement event number specified by *performanceNumber*.
- If *performanceNumber* is not specified, then performance measurement of all performance measurement event numbers will be enabled.

[Example of use]

```
>>>debugger.Performance.Enable()
True
>>>
```

debugger.Performance.Get

This function references the result of performance measurement. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.Performance.Get()
```

[Argument(s)]

None

[Return value]

List of performance measurement information (see the PerformanceInfo class for detail)

[Detailed description]

- Performance measurement information is shown by the following format.

```
[performance-measurement-event-number] [count] [performance-measurement-mode] [per-
formance-measurement-item]
```

[Example of use]

```
>>>pf = debugger.Performance.Get()
1 2030 MaxCount AllFetchCall
2 3000 MinCount AllFetchBranch
>>>print pf[0].Count
2030
>>>print pf[0].Mode
PerformanceMode.MaxCount
>>>
```

debugger.Performance.Information

This function displays performance measurement information. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.Performance.Information()
```

[Argument(s)]

None

[Return value]

List of performance measurement information (see the PerformanceEventInfo class for detail)

[Detailed description]

- This function displays software trace information displays in the following format.

```
[performance-measurement-event-number] [performance-measurement-name] [state]
[start-address] - [end-address]
```

[Example of use]

```
>>>pi = debugger.Performance.Information()
1 PythonPerformanceMeasurement001 Enable 0x00000200 - 0x00000300
>>>print pi.Enable
True
>>>print pi.StartAddress
0x00000200
>>>
```

debugger.Performance.Set

This function sets performance measurement. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.Performance.Set(PerformanceCondition)
```

[Argument(s)]

Argument	Description
PerformanceCondition	Specify a condition of performance measurement. See the PerformanceCondition class for a condition of performance measurement.

[Return value]

If performance measurement was set successfully: Performance measurement event number
If there was an error when setting performance measurement: None

[Detailed description]

- This function sets performance measurement according to the contents specified with *PerformanceCondition*.

[Example of use]

```
>>>pf = PerformanceCondition()
>>>pf.StartAddress = 0x1000
>>>pf.EndAddress = 0xffe000
>>>pf.EndData = 0x10
>>>pf.EndPerformanceType = PerformanceType.Read
>>>pf.PerformanceMode = PerformanceMode.MaxCount
>>>pf.PerformanceItem = PerformanceItem.AllFetchBranch
>>>ps = debugger.Performance.Set(pf)
1
>>>print ps
1
>>>
```

debugger.PseudoError.Clear

This function clears the error status of all pseudo-errors. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.PseudoError.Clear()
```

[Argument(s)]

None

[Return value]

If the error status of all pseudo-errors was cleared successfully: True
If there was an error when clearing the error status of all pseudo-errors: False

[Detailed description]

- This function clears the error status of all pseudo-errors.

[Example of use]

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

debugger.PseudoError.Get

This function references ECM error information. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.PseudoError.Get(nameList = [])
```

[Argument(s)]

Argument	Description
nameList	Specify the list of error names (abbreviated form) to be acquired.

[Return value]

List of ECM error information (see the PseudoErrorInfo class for detail)

[Detailed description]

- ECM error information is shown by the following format.

```
[number] [error-name (abbreviated-form) ] [bit-IOR-name] [error-value]
```

[Example of use]

```
>>>rl = ["ECC_DED", "ECC_CodeFlash_AddressOverflow"]
>>>ei = debugger.PseudoError.Get(rl)
28 ECC_DED ECMPE028 False
35 ECC_CodeFlash_AddressOverflow ECMPE103 False
>>>print ei[0].Name
ECC_DED
>>>print ei[0].BitName
ECMPE028
>>>
```

debugger.PseudoError.SetGo

This function sets conditions of a pseudo-error and runs a program. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.PseudoError.SetGo(PseudoErrorCondition[], runOption = RunOption.Normal)
```

[Argument(s)]

Argument	Description	
PseudoErrorCondition[]	Specify conditions of a pseudo-error as a list. See the PseudoErrorCondition class for details.	
runOption	Specify whether to wait until the program stops.	
	Type	Description
	RunOption.WaitBreak	Wait until the program stops.
	RunOption.Normal	Do not wait until the program stops (default).

[Return value]

If pseudo-error conditions were set and the program was executed successfully: True
If there was an error when setting pseudo-error conditions and executing the program: False

[Detailed description]

- This function sets conditions of a pseudo-error and runs a program according to the contents specified with *PseudoErrorCondition[]*.

[Example of use]

```
>>>pe = PseudoErrorCondition()
>>>pe.Name = "ECC_DTS_2Bit"
>>>pe1 = PseudoErrorCondition()
>>>pe1.BitName = "ECMPE023"
>>>pe1.BreakAddress = [0x2000, "main]
>>>debugger.PseudoError.SetGo([pe, pe1])
True
>>>
```

Caution If both Name and BitName are set as conditions of a pseudo-error, Name is given priority and BitName is ignored.

注意 疑似エラー条件に Name と BitName の両方を指定した場合は，Name が優先され，BitName は無視されます。

debugger.PseudoTimer.Delete

This function deletes pseudo-timers. [RL78 instruction simulator]

[Specification format]

```
debugger.PseudoTimer.Delete(pseudoTimerNumber = 0)
```

[Argument(s)]

Argument	Description
pseudoTimerNumber	Specify the pseudo-timer number to be deleted (numerical value: 1 to 64). Delete all pseudo-timers (numerical value: 0).

[Return value]

If pseudo-timers were deleted successfully: True
If there was an error when deleting pseudo-timers : False

[Detailed description]

- This function stops and deletes pseudo-timers specified with pseudoTimerNumber.
- If pseudoTimerNumber is not specified or is specified as 0, pseudo-timers with all pseudo-timer numbers will be stopped and deleted.

[Example of use]

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

debugger.PseudoTimer.Information

This function displays the pseudo-timer information. [RL78 instruction simulator]

[Specification format]

```
debugger.PseudoTimer.Information()
```

[Argument(s)]

None

[Return value]

List of pseudo-timer information (See the PseudoTimerInfo class for details.)

[Detailed description]

- The specified pseudo-timer information is shown in the format below.

```
pseudo-timer-number vector-address Priority: interrupt-priority Interval Time:
interval-time{Min|S|Ms|Us|Clock} {Periodic|Once}
```

[Example of use]

```
>>>pi = debugger.PseudoTimer.Information()
1 0x40 Priority: 3 Interval Time: 1Ms Periodic
3 0x7c Priority: 0 Interval Time: 10Clock Once
>>>print pi[0].Number
1
>>> print pi[1].Number
3
>>> print pi[0].Periodic
True
```

debugger.PseudoTimer.Set

This function sets pseudo-timers. [RL78 instruction simulator]

[Specification format]

```
debugger.PseudoTimer.Set(pseudoTimerNumber, PseudoTimerCondition)
```

[Argument(s)]

Argument	**Caution 1.** Description
pseudoTimerNumber	Specify the pseudo-timer number to be set (numerical value: 1 to 61).
PseudoTimerCondition	Specify the pseudo-timer condition. See the PseudoTimerCondition class for creating pseudo-timer conditions.

[Return value]

If pseudo-timers were set successfully: True
If there was an error when setting pseudo-timers: False

[Detailed description]

- A pseudo-timer with the specified pseudo-timer number is set and counting is started according to the description specified by PseudoTimerCondition.
- When counting reaches the interval time specified in PseudoTimerCondition, an interrupt is generated with the specified priority in the specified interrupt vector address.
- If "Reset" or 0x0 is specified for the interrupt vector address, a reset will be generated.
- An interrupt is generated in the interrupt enable (EI) state. It is reserved in the interrupt disable (DI) state and generated when the state next becomes interrupt enable (EI).
- Setting registers for the interrupt functions is not required. Among such registers, only the PSW is changed by an interrupt.
- If a reset is generated, counting by the pseudo-timer will be stopped and all pseudo-timers which have been set will be deleted.
- The pseudo-timer continues counting in the halt and stop modes.
- If the pseudo-timer number which has already been set is specified, the given pseudo-timer will be enabled and setting of the new pseudo-timer will fail.

[Example of use]

```
>>>ptc1 = PseudoTimerCondition()
>>>ptc1.VectorAddr = 0x40
>>>ptc1.Priority = 3
>>>ptc1.IntervalTime = 1
>>>ptc1.IntervalTimeUnit = IntervalTimeUnit.Ms
>>>ptc1.Periodic = False
>>>debugger.PseudoTimer.Set(1, ptc1)
True
>>>
```

debugger.RecoverSWAS

This function recovers the Switch Area Status. [RH850 G4MH] [E2/IE850A]

[Specification format]

```
debugger.Register.RecoverSWAS()
```

[Argument(s)]

None

[Return value]

If recovery of the Switch Area Status was successful: True
If recovery of the Switch Area Status failed : False

[Detailed description]

- This function recovers the Switch Area Status.

[Example of use]

```
>>>debugger.RecoverSWAS()
True
>>>
```

debugger.Register.GetValue

This function refers register/IO register/SFR.

[Specification format]

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

[Argument(s)]

Argument	Description
regName	Specify the register name to reference.

[Return value]

Register value (numeric value)

[Detailed description]

- This function displays the value of the register specified by "regName".

[Example of use]

```
>>>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

This function sets the value of a register, IO register, and SFR.

[Specification format]

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

[Argument(s)]

Argument	Description
regName	Specify the register name to set.
value	Specify the value to set.

[Return value]

If the value was set successfully: True
If there was an error when setting the value: False

[Detailed description]

- This function sets the value specified by *value* in the register specified by *regName*.

[Example of use]

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

debugger.Reset

This function resets the CPU.

[Specification format]

```
debugger.Reset()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- This function resets the CPU.

Caution This function will not be executed after the CPU has been reset regardless of the setting of the [Execute to the specified symbol after CPU Reset] property.

[Example of use]

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

debugger.ReturnOut

This function runs until control returns to the program that called the current function.

[Specification format]

```
debugger.ReturnOut()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- This function runs until control returns to the program that called the current function.

[Example of use]

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

debugger.Run

This function resets and then run the program.

[Specification format]

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

[Argument(s)]

Argument	Description	
runOption	Specify an option. The options that can be specified are shown below.	
	Type	Description
	RunOption.WaitBreak	Wait until the program stops.
	RunOption.Normal	Breakpoints enabled; do not wait until the program stops (default).

[Return value]

None

[Detailed description]

- This function resets and then run the program.
  If "RunOption.WaitBreak" is specified in *runOption*, then it will wait until the program stops.

[Example of use]

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

debugger.SaveRegisterBank.Information

This function displays information on the save register bank. [RX]

[Specification format]

```
debugger.SaveRegisterBank.Information(bankNumberList = [])
```

[Argument(s)]

Argument	Description
bankNumberList	Specify the bank number of the banked register to display information (default: not specified). When specifying multiple numbers, specify them by delimiting with a comma.

[Return value]

List of banked register information (see the BankedRegisterInfo property for detail)

[Detailed description]

- This function displays information on the save register bank in the following format.

```
bank-number
register-name value
```

- When *bankNumberList* is specified, information on the specified bank number.
- When *bankNumberList* is not specified, then information on all banks will be displayed.

[Example of use]

```
>>> srb = debugger.SaveRegisterBank.Information([1, 3])

Save register bank 1
R1 0x00000000
R2 0x00000000
...
ACC0 0x000000000000000000
ACC1 0x000000000000000000
Save register bank 3
R1 0x00000000
R2 0x00000000
...
ACC0 0x000000000000000000
ACC1 0x000000000000000000
--------------
>>> print srb[0].BankNumber
1
>>> print srb[0].RegisterName
R1
>>> print srb[0].Value
0
```

debugger.SoftwareTrace.Delete

This function deletes a software trace. [RH850]

[Specification format]

```
debugger.SoftwareTrace.Delete()
```

[Argument(s)]

None

[Return value]

If a software trace was deleted successfully: True
If there was an error when deleting a software trace: False

[Detailed description]

- This function deletes conditions of the software trace specified by debugger.SoftwareTrace.Set.

[Example of use]

```
>>>debugger.SoftwareTrace.Delete()
True
>>>
```

debugger.SoftwareTrace.Disable

This function disables a software trace. [RH850]

[Specification format]

```
debugger.SoftwareTrace.Disable()
```

[Argument(s)]

None

[Return value]

If a software trace was disabled successfully: True
If there was an error when disabling a software trace: False

[Detailed description]

- This function disables a software trace.

[Example of use]

```
>>>debugger.SoftwareTrace.Disable()
True
>>>
```

debugger.SoftwareTrace.Enable

This function enables a software trace. [RH850]

[Specification format]

```
debugger.SoftwareTrace.Enable()
```

[Argument(s)]

None

[Return value]

If a software trace was enabled successfully: True
If there was an error when enabling a software trace: False

[Detailed description]

- This function enables a software trace.

[Example of use]

```
>>>debugger.SoftwareTrace.Enable()
True
>>>
```

debugger.SoftwareTrace.Get

This function refers to the software trace data for the specified number of frames.
This function also outputs the acquired software trace data to a file. [RH850]

[Specification format]

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

[Argument(s)]

Argument	Description
frameCount	Specify the number of frames for which software trace data is acquired.
fileName	Specify the full path of the file name to which data will be output (default: not specified).
append	Specify whether to append software trace data to the file. True: Append software trace data to the file. False: Do not append software trace data to the file (default).

[Return value]

Software trace data (see the SoftwareTraceInfo class for detail)
If there is no data, None is set.

[Detailed description]

- The software trace data is shown by the following format.

When the microcontroller is single core:

- DBCP

```
number-of-frames timestamp PC DBCP
```

- DBTAG (with PC)

```
number-of-frames timestamp PC category data DBTAG
```

- DBTAG (without PC)

```
number-of-frames timestamp category data DBTAG
```

- DBPUSH (with PC)

```
number-of-frames timestamp PC register-ID register-data DBPUSH
```

- DBPUSH (without PC)

```
number-of-frames timestamp register-ID register-data DBPUSH
```

When the microcontroller is multi-core:

- DBCP

```
number-of-frames PE-number timestamp PC DBCP
```

- DBTAG (with PC)

```
number-of-frames PE-number timestamp PC category data DBTAG
```

- DBTAG (without PC)

```
number-of-frames PE-number timestamp category data DBTAG
```

- DBPUSH (with PC)

```
number-of-frames PE-number timestamp PC register-ID register-data DBPUSH
```

- DBPUSH (without PC)

```
number-of-frames PE-number timestamp register-ID register-data DBPUSH
```

[Example of use]

```
>>>trace = debugger.SoftwareTrace.Get(100)
 99  00h00min00s003ms702us000ns 0x00001028 0x03 0x20 DBTAG
 99  00h00min00s003ms702us000ns 0x00001030 0x03 0x0020 DBPUSH
100  00h00min00s003ms702us000ns 0x00001032 DBCP
>>>
```

debugger.SoftwareTrace.Information

This function displays software trace information. [RH850]

[Specification format]

```
debugger.SoftwareTrace.Information()
```

[Argument(s)]

None

[Return value]

List of software trace information (see the SoftwareTraceEventInfo class for detail)

[Detailed description]

- This function displays software trace information displays in the following format.

```
[state] DBCP=[DBCP-information] DBTAG=[DBTAG-information] DBPUSH=[DBPUSH-informa-
tion] PC=[PC-information]
```

[Example of use]

```
>>>si = debugger.SoftwareTrace.Information()
Enable DBCP=False DBTAG=True DBPUSH=False PC=False
>>>print si.DBCP
False
>>>print si.DBTAG
True
>>>print si.PC
False
>>>
```

debugger.SoftwareTrace.Set

This function sets a software trace. [RH850]

Caution [Except simulator]
When software trace is set, trace data cannot be acquired. Either delete the software trace setting or disable software trace.
Use debugger.SoftwareTrace.Delete to delete the software trace setting and use debugger.SoftwareTrace.Disable to disable software trace.

[Specification format]

```
debugger.SoftwareTrace.Set(DBCP, DBTAG, DBPUSH, PC = True)
```

[Argument(s)]

Argument	Description
DBCP	Specify whether to acquire the results of DBCP. True: Acquire the results of DBCP. False: Do not acquire the results of DBCP.
DBTAG	Specify whether to acquire the results of DBTAG. True: Acquire the results of DBTAG. False: Do not acquire the results of DBTAG.
DBPUSH	Specify whether to acquire the results of DBPUSH. True: Acquire the results of DBPUSH. False: Do not acquire the results of DBPUSH.
PC	Specify whether to include information of the PC address in the results of DBTAG and DBPUSH. True: Include information of the PC address (default). False: Do not include information of the PC address.

[Return value]

If a software trace was set successfully: True
If there was an error when setting a software trace: False

[Detailed description]

- This function sets a software trace.

[Example of use]

```
>>>debugger.SoftwareTrace.Set(True,True,False,False)
True
>>>
```

debugger.SoftwareTraceLPD.Delete

This function deletes a software trace (LPD output). [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.Delete()
```

[Argument(s)]

None

[Return value]

If a software trace (LPD output) was deleted successfully: True
If there was an error when deleting a software trace (LPD output): False

[Detailed description]

- This function deletes conditions of the software trace (LPD output) specified by debugger.SoftwareTraceLPD.Set.

[Example of use]

```
>>>debugger.SoftwareTraceLPD.Delete()
True
>>>
```

debugger.SoftwareTraceLPD.Disable

This function disables a software trace (LPD output). [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.Disable()
```

[Argument(s)]

None

[Return value]

If a software trace (LPD output) was disabled successfully: True
If there was an error when disabling a software trace (LPD output): False

[Detailed description]

- This function disables a software trace (LPD output).

[Example of use]

```
>>>debugger.SoftwareTraceLPD.Disable()
True
>>>
```

debugger.SoftwareTraceLPD.Enable

This function enables a software trace (LPD output). [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.Enable()
```

[Argument(s)]

None

[Return value]

If a software trace (LPD output) was enabled successfully: True
If there was an error when enabling a software trace (LPD output): False

[Detailed description]

- This function enables a software trace (LPD output).

[Example of use]

```
>>>debugger.SoftwareTraceLPD.Enable()
True
>>>
```

debugger.SoftwareTraceLPD.Get

This function refers to the software trace (LPD output) data for the specified number of frames.
This function also outputs the acquired software trace (LPD output) data to a file. [RH850][E2]

[Specification format]

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

[Argument(s)]

Argument	Description
frameCount	Specify the number of frames for which software trace (LPD output) data is acquired.
fileName	Specify the full path of the file name to which data will be output (default: not specified).
append	Specify whether to append software trace (LPD output) data to the file. True: Append software trace data to the file. False: Do not append software trace data to the file (default).

[Return value]

Software trace data (LPD output) (see the SoftwareTraceInfo class for detail)
If there is no data, None is set.

[Detailed description]

- The software trace (LPD output) data is shown by the following format.

  When the microcontroller is single core:

  - DBCP

    ```
    number-of-frames timestamp PC DBCP
    ```

  - DBTAG (with PC)

    ```
    number-of-frames timestamp PC category data DBTAG
    ```

  - DBTAG (without PC)

    ```
    number-of-frames timestamp category data DBTAG
    ```

  - DBPUSH (with PC)

    ```
    number-of-frames timestamp PC register-ID register-data DBPUSH
    ```

  - DBPUSH (without PC)

    ```
    number-of-frames timestamp register-ID register-data DBPUSH
    ```

When the microcontroller is multi-core:

- DBCP

```
number-of-frames PE-number timestamp PC DBCP
```

- DBTAG (with PC)

```
number-of-frames PE-number timestamp PC category data DBTAG
```

- DBTAG (without PC)

```
number-of-frames PE-number timestamp category data DBTAG
```

- DBPUSH (with PC)

```
number-of-frames PE-number timestamp PC register-ID register-data DBPUSH
```

- DBPUSH (without PC)

```
number-of-frames PE-number timestamp register-ID register-data DBPUSH
```

[Example of use]

```
>>>trace = debugger.SoftwareTraceLPD.Get(100)
 99  00h00min00s003ms702us000ns 0x00001028 0x03 0x20 DBTAG
 99  00h00min00s003ms702us000ns 0x00001030 0x03 0x0020 DBPUSH
100  00h00min00s003ms702us000ns 0x00001032 DBCP
>>>
```

debugger.SoftwareTraceLPD.Information

This function displays software trace (LPD output) information. [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.Information()
```

[Argument(s)]

None

[Return value]

List of software trace (LPD output) information (see the SoftwareTraceLPDEventInfo class for detail)

[Detailed description]

- This function displays software trace (LPD output) information displays in the following format.

```
[state] DBCP=[DBCP-information] DBTAG=[DBTAG-information] DBPUSH=[DBPUSH-informa-
tion] PC=[PC-information] PE=[PE-number]
```

[Example of use]

```
>>>si = debugger.SoftwareTraceLPD.Information()
Enable DBCP=False DBTAG=True DBPUSH=False PC=False PE=1
>>>print si.DBCP
False
>>>print si.DBTAG
True
>>>print si.PC
False
>>>
```

debugger.SoftwareTraceLPD.Set

This function sets a software trace (LPD output). [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.Set(DBCP, DBTAG, DBPUSH, PC = True, PE)
```

[Argument(s)]

Argument	Description
DBCP	Specify whether to acquire the results of DBCP. True: Acquire the results of DBCP. False: Do not acquire the results of DBCP.
DBTAG	Specify whether to acquire the results of DBTAG. True: Acquire the results of DBTAG. False: Do not acquire the results of DBTAG.
DBPUSH	Specify whether to acquire the results of DBPUSH. True: Acquire the results of DBPUSH. False: Do not acquire the results of DBPUSH.
PC	Specify whether to include information of the PC address in the results of DBTAG and DBPUSH. True: Include information of the PC address (default). False: Do not include information of the PC address.
PE	If the device is multi-core, specify the number of the core from which you wish to acquire trace information. Specification of this argument is ignored if the device in use only has one core.

[Return value]

If a software trace (LPD output) was set successfully: True
If there was an error when setting a software trace (LPD output): False

[Detailed description]

- This function sets a software trace (LPD output).

[Example of use]

```
>>>debugger.SoftwareTraceLPD.Set(True,True,False,False,1)
True
>>>
```

debugger.Step

This function performs step execution.

[Specification format]

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

[Argument(s)]

Argument	Description	
stepOption	Specify the execution unit. The units that can be specified are shown below.	
	Type	Description
	StepOption.Source	Source-line unit (default)
	StepOption.Instruction	Instruction unit

[Return value]

None

[Detailed description]

- This function performs step execution.
  If a function call is being performed, then stop at the top of the function.

[Example of use]

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

debugger.Stop

This function stops the execution of the debug tool.

[Specification format]

```
debugger.Stop()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- This function stops the execution of the debug tool.
  Forcibly halt the program.

[Example of use]

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

debugger.Timer.Clear

This function clears the result measured by a conditional timer.

[Specification format]

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

[Argument(s)]

None

[Return value]

If the result measured by a conditional timer was cleared successfully: True
If there was an error when clearing the result measured by a conditional timer: False

[Detailed description]

- This function clears the result measured by a conditional timer.

[Example of use]

```
>>>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

This function deletes a conditional timer.

[Specification format]

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

[Argument(s)]

Argument	Description
timerNumber	Specify the timer event number to delete.

[Return value]

If a timer was deleted successfully: True
If there was an error when deleting a timer: False

[Detailed description]

- This function deletes the timer of the timer event number specified by *timerNumber*.
- If *timerNumber* is not specified, then timers of all timer event numbers will be deleted.

[Example of use]

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

debugger.Timer.Detail

This function sets measurement conditions of a conditional timer. [RH850][E1/E20/Full-spec emulator/IE850A]

[Specification format]

```
debugger.Timer.Detail(timerNumber = "", timerOption)
```

[Argument(s)]

Argument	Description		
timerNumber	Specify the timer event number for which you wish to set measurement conditions.		
timerOption	Set measurement conditions of a conditional timer.		
	Member	Target Device	Description
	TimerOption.PassCount	RH850	Pass count
	TimerOption.MinCount	RH850	Minimum count
	TimerOption.MaxCount	RH850	Maximum count
	TimerOption.AddCount	RH850	Add count

[Return value]

If setting measurement conditions of a conditional timer was disabled successfully: True
If there was an error when setting measurement conditions of a conditional timer: False

[Detailed description]

- This function sets measurement conditions of the timer event number specified by *timerNumber*.
- If *timerNumber* is not specified, then measurement conditions of all timer events will be set.

[Example of use]

```
>>>debugger.Timer.Information()
1 Timer Result1 Enable 0x00001000 - 0x00002000
2 Timer Result2 Enable 0x00003000 - 0x00004000
>>>debugger.Timer.Detail(1, TimerOption.PassCount)    …Change the timer measurement
condition to pass count
True
>>>
```

```
>>>debugger.Timer.Detail(TimerOption.MaxCount)    …Change the timer measurement condi-
tion of all timer events to maximum execution time
True
>>>
```

debugger.Timer.Disable

This function disables a conditional timer.

[Specification format]

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

[Argument(s)]

Argument	Description
timerNumber	Specify the timer event number to disable.

[Return value]

If a timer setting was disabled successfully: True
If there was an error when disabling a timer setting: False

[Detailed description]

- This function disables the timer of the timer event specified by *timerNumber*.
- If *timerNumber* is not specified, then timers of all timer event numbers will be disabled.

[Example of use]

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

debugger.Timer.Enable

This function enables a conditional timer.

[Specification format]

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

[Argument(s)]

Argument	Description
timerNumber	Specify the timer event number to enable.

[Return value]

If a timer setting was enabled successfully: True
If there was an error when enabling a timer setting: False

[Detailed description]

- This function enables the timer of the timer event specified by *timerNumber*.
- If *timerNumber* is not specified, then timers of all timer event numbers will be enabled.

[Example of use]

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

debugger.Timer.Get

This function references the result measured by a conditional timer.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of conditional timer information (see the TimerInfo class for detail)

[Detailed description]

- The result measured by a conditional timer is shown by the following format.

```
timer-event-number Total: total-execution-time ns, Pass Count: pass-count , Average:
average-execution-time ns, Max: maximum-execution-time ns, Min: minimum-execution-
time ns
```

[Example of use]

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

debugger.Timer.Information

This function displays conditional timer information.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of conditional timer event information (see the TimerEventInfo class for detail)

[Detailed description]

- This function displays conditional timer information displays in the following format.

```
timer-event-number timer-name state start-address - end-address
```

[Example of use]

```
>>>ti = debugger.Timer.Information()
1 PythonTimer0001 Enable main - sub
>>>print ti[0].Number
1
>>>print ti[0].Name
PythonTimer0001
>>>
```

debugger.Timer.Set

This function sets a conditional timer.

[Specification format]

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

[Argument(s)]

Argument	Description
TimerCondition	Specify a condition of a conditional timer. See the TimerCondition class for creating a conditional timer.

[Return value]

Set timer event number (numerical value)

[Detailed description]

- This function sets a conditional timer according to the contents specified with *TimerCondition*.
- The specified conditional timer is registered with the following name.
  number is a four-digit decimal.

```
Python Timer number
```

[Example of use]

```
>>>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

This function clears the trace memory.

Remark This function provides the same function as debugger.XTrace.Clear.

[Specification format]

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

[Argument(s)]

None

[Return value]

If the trace memory was cleared successfully: True
If there was an error when clearing the trace memory: False

[Detailed description]

- This function clears the trace memory.

[Example of use]

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

debugger.Trace.Delete

This function deletes a conditional trace.

[Specification format]

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

[Argument(s)]

Argument	Description
timerNumber	Specify the trace event number to delete.

[Return value]

If a trace was deleted successfully: True
If there was an error when deleting a trace: False

[Detailed description]

- This function deletes the trace of the trace event number specified by *tracenumber*.
- If *traceNumber* is not specified, then traces of all trace event numbers will be deleted.

[Example of use]

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

debugger.Trace.Disable

This function disables a conditional trace.

[Specification format]

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

[Argument(s)]

Argument	Description
traceNumber	Specify the trace event number to disable.

[Return value]

If a trace setting was disabled successfully: True
If there was an error when disabling a trace setting: False

[Detailed description]

- This function disables the timer of the trace event specified by *traceNumber*.
- If *traceNumber* is not specified, then traces of all trace event numbers will be disabled.

[Example of use]

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

debugger.Trace.Enable

This function enables a conditional trace.

[Specification format]

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

[Argument(s)]

Argument	Description
traceNumber	Specify the trace event number to enable.

[Return value]

If a trace setting was enabled successfully: True
If there was an error when enabling a trace setting: False

[Detailed description]

- This function enables the timer of the trace event specified by *traceNumber*.
- If *traceNumber* is not specified, then traces of all trace event numbers will be enabled.

[Example of use]

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

debugger.Trace.Get

This function dumps the trace data.

Remark This function provides the same function as debugger.XTrace.Dump.

[Specification format]

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

[Argument(s)]

Argument	Description
frameCount	Specify the number of dumps.
fileName	Specify the name of the file to dump to (default: not specified).
append	Specify whether to append trace data to the file. True: Append trace data to the file. False: Do not append trace data to the file (default).

[Return value]

List of trace information (see the TraceInfo property for detail)

[Detailed description]

- This function dumps trace data for the number of frames specified by *frameCount*.
- If *fileName* is specified, then the trace data is written to the file.
- If *append* is set to "True", then the trace data is appended to the file.
- This function displays the trace data in the following format. Information which is not included in the trace data is displayed as blank space.

  For a single-core microcontroller

  - Execution of instructions

    ```
    number-of-frames timestamp fetch-address mnemonic
    ```

  - Read access

    ```
    number-of-frames timestamp read-address R read-data
    ```

  - Write access

    ```
    number-of-frames timestamp write-address W write-data
    ```

  - Vector-read access

    ```
    number-of-frames timestamp vector-read-address V vector-read-data
    ```

  - DMA

    ```
    number-of-frames timestamp DMA
    ```

For a multi-core microcontroller

- Execution of instructions

```
number-of-frames PE-number timestamp fetch-address mnemonic
```

- Read access

```
number-of-frames timestamp read-address R read-data
```

- Write access

```
number-of-frames timestamp write-address W write-data
```

- Vector-read access

```
number-of-frames timestamp vector-read-address V vector-read-data
```

- DMA

```
number-of-frames timestamp DMA
```

[Example of use]

```
>>>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

This function displays conditional trace information.

[Specification format]

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

[Argument(s)]

None

[Return value]

List of conditional trace information (see the TraceEventInfo class for detail)

[Detailed description]

- This function displays conditional trace information is shown by the following format.

```
trace-event-number Trace state start-address - end-address
```

[Example of use]

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

debugger.Trace.Set

This function sets a conditional trace.

[Specification format]

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

[Argument(s)]

Argument	Description
TraceCondition	Specify a condition of a conditional trace. See the TraceCondition class for creating a conditional trace.

[Return value]

Set trace event number (numerical value)

[Detailed description]

- This function sets a conditional trace according to the contents specified with *TraceCondition*.
- The specified conditional trace is registered with the following name.

```
Trace
```

[Example of use]

```
>>>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

This function saves the memory data in binary format.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a file name.
address1	Specify an upload start address.
address2	Specify an upload end address.
force	Specify whether to overwrite. True: Overwrite False: Do not overwrite (default).

[Return value]

If the memory data was uploaded successfully: True
If there was an error when uploading the memory data: False

[Detailed description]

- This function saves the memory data from *address1* to *address2* in binary format.

[Example of use]

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

debugger.Upload.Coverage

This function saves the coverage data. [Simulator]

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a file name.
force	Specify whether to overwrite. True: Overwrite False: Do not overwrite (default).

[Return value]

If the memory data was uploaded successfully: True
If there was an error when uploading the memory data: False

[Detailed description]

- This function saves the coverage data to a file.

[Example of use]

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

debugger.Upload.Intel

This function saves the memory data in Intel format.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a file name.
address1	Specify an upload start address.
address2	Specify an upload end address.
force	Specify whether to overwrite. True: Overwrite False: Do not overwrite (default).

[Return value]

If the memory data was uploaded successfully: True
If there was an error when uploading the memory data: False

[Detailed description]

- This function saves the memory data from *address1* to *address2* in Intel format.

[Example of use]

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

debugger.Upload.Motorola

This function saves the memory data in Motorola format.

[Specification format]

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

[Argument(s)]

Argument	Description
fileName	Specify a file name.
address1	Specify an upload start address.
address2	Specify an upload end address.
force	Specify whether to overwrite. True: Overwrite False: Do not overwrite (default).

[Return value]

If the memory data was uploaded successfully: True
If there was an error when uploading the memory data: False

[Detailed description]

- This function saves the memory data from *address1* to *address2* in Motorola format.

[Example of use]

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

debugger.Watch.GetValue

This function refers to a variable value.

[Specification format]

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

[Argument(s)]

<table>
<tr><th>Argument</th><th colspan="2">Description</th></tr>
<tr><td>variableName</td><td colspan="2">Specify the variable name, register name, or I/O register name/SFR register name to reference.</td></tr>
<tr><td>encode</td><td colspan="2">Specify the encoding to use when displaying strings.
By default, the system encoding is used.
The encoding name conforms to the .NET specifications.
Examples: Encoding.utf-8, Encoding.euc-jp</td></tr>
<tr><td rowspan="13">watchOption</td><td colspan="2">Specify an option.
The options that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>WatchOption.Auto</td><td>Automatically detect when displaying (default).</td></tr>
<tr><td>WatchOption.Binary</td><td>Display in binary format.</td></tr>
<tr><td>WatchOption.Octal</td><td>Display in octal format.</td></tr>
<tr><td>WatchOption.Decimal</td><td>Display in decimal format.</td></tr>
<tr><td>WatchOption.SignedDecimal</td><td>Display in signed decimal format.</td></tr>
<tr><td>WatchOption.UnsignedDecimal</td><td>Display in unsigned decimal format.</td></tr>
<tr><td>WatchOption.Hexdecimal</td><td>Display in hexadecimal format.</td></tr>
<tr><td>WatchOption.String</td><td>Display as a string.</td></tr>
<tr><td>WatchOption.Sizeof</td><td>Display the variable size in decimal format.</td></tr>
<tr><td>WatchOption.Float</td><td>Display in float type.</td></tr>
<tr><td>WatchOption.Double</td><td>Display in double type.</td></tr>
</table>

[Return value]

The displayed value is returned in the format specified by *watchOption*.
When *watchOption* is specified as "WatchOption.Auto", the format is returned to match the variable value.
However, if the return value is a double type, it is returned as a string (when *watchOption* is specified as "*WatchOption*.Double", or *watchOption* is specified as "WatchOption.Auto" and the return value is a double type).

[Detailed description]

- This function displays the value of the variable specified by *variableName*.
- If *encode* is specified, then perform encoding using *encode*.
- If *watchOption* is specified, display according to *watchOption*.

Caution When a load module name or file name is specified as a variable (*variableName*), it needs to be enclosed in double quotation marks (" ") in some cases. See "CS+ Integrated Development Environment User's Manual: Debug Tool" for details.

Example When file name "C:\path\test.c" and variable "var" are specified

```
"\"C:/path/test.c\"#var"
```

Or

```
"\"C:\\path\\test.c\"#var"
```

[Example of use]

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

debugger.Watch.SetValue

This function sets a variable value.

[Specification format]

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

[Argument(s)]

Argument	Description
variableName	Specify the variable name, register name, and I/O register name or SFR register name to set.
value	Specify the value to set.

[Return value]

If a variable value was set successfully: True
If there was an error when setting a variable value: False

[Detailed description]

- This function sets the value specified by *value* in the variable, register, and I/O register or SFR register specified by *variableName*.

Caution When a load module name or file name is specified as a variable (*variableName*), it needs to be enclosed in double quotation marks (" ") in some cases. See "CS+ Integrated Development Environment User's Manual: Debug Tool" for details.

Example When file name "C:\path\test.c" and variable "var" are specified

```
"\"C:/path/test.c\"#var"
```

Or

```
"\"C:\\path\\test.c\"#var"
```

[Example of use]

```
>>>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

This function displays a stack backtrace.

[Specification format]

```
debugger.Where()
```

[Argument(s)]

None

[Return value]

List of a backtrace (see the StackInfo property for detail)

[Detailed description]

- This function displays a stack backtrace.

Caution If "--- Information below might be inaccurate." is displayed, then the information displayed below may not be reliable. [RL78][78K0R]

[Example of use]

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

debugger.Whereami

This function displays a location.

[Specification format]

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

[Argument(s)]

Argument	Description
address	Specify the address of the location to display.

[Return value]

Strings of the location

[Detailed description]

- This function displays the location at the address specified by *address*.
- The location is normally displayed in the following format.

```
file-name#function-name at file-name#line-number
```

However, if the function or line number at that address is not found, then the location is displayed in the following format.

```
at symbol-name+offset-value
```

If the symbol is not found, then the location is displayed in the following format.

```
at address-value
```

If *address* is omitted, then the location of the pc value is displayed.

[Example of use]

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

debugger.XCoverage.Clear

This function clears the coverage memory. [IECUBE][IECUBE2][Simulator]

[Specification format]

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

[Argument(s)]

None

[Return value]

If the coverage memory was cleared successfully: True
If there was an error when clearing the coverage memory: False

[Detailed description]

- This function clears the coverage memory.

[Example of use]

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

debugger.XCoverage.GetCoverage

This function gets the coverage. [IECUBE][IECUBE2][Simulator]

[Specification format]

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

[Argument(s)]

Argument	Description
funcName	Specify the function name to retrieve coverage for.
progName	Specify the name of the load module containing the function. If there is only one load module, then this can be omitted (default).
fileName	Specify the name of the file containing the function. If it is a global function, then this can be omitted (default).

Caution If two or more parameters are specified, then three parameters must be specified.

[Return value]

Value without "%" (numeric value)

Remark The results of function execution are displayed with a "%" sign added.

[Detailed description]

- This function gets coverage for the function specified by *funcName*.
- If there are multiple load modules, specify *progName*.
- In the case of a static function, specify *fileName*.

Caution When a load module name (*progName*) or file name (*fileName*) is specified, it needs to be enclosed in double quotation marks (" ") in some cases. See "CS+ Integrated Development Environment User's Manual: Debug Tool" for details.

Example When file name "C:\path\test.c" is specified

```
"\"C:/path/test.c\""
```

Or

```
"\"C:\\path\\test.c\""
```

[Example of use]

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

debugger.XRunBreak.Delete

This function deletes XRunBreak information. [V850 Simulator][RH850 Simulator]

[Specification format]

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

[Argument(s)]

None

[Return value]

If XRunBreak information was deleted successfully: True
If there was an error when deleting XRunBreak information: False

[Detailed description]

- This function deletes XRunBreak information.

[Example of use]

```
>>>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

This function displays XRunBreak setting information. [V850 Simulator][RH850 Simulator]

[Specification format]

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

[Argument(s)]

None

[Return value]

List of period time value and period information (TimeType) (see the XRunBreakInfo property for detail)

[Detailed description]

- This function displays the period information (period time [Periodic]) of the set XRunBreak.
- If there is no XRunBreak setting, "None" is displayed.

[Example of use]

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

debugger.XRunBreak.Set

This function configures XRunBreak information. [V850 Simulator][RH850 Simulator]

[Specification format]

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

[Argument(s)]

Argument	Description	
time	Specify the break time.	
timeType	Specify the break time unit. The units that can be specified are shown below.	
	Type	Description
	TimeType.Min	Minute unit
	TimeType.S	Second unit
	TimeType.Ms	Millisecond unit (default)
	TimeType.Us	Microsecond unit
	TimeType.Ns	Nanosecond unit
periodic	Specify whether to call the callback every time the specified time elapses. True: Call at every specified time interval. False: Call one time only (default).	

[Return value]

If XRunBreak information was configured successfully: True
If there was an error when configuring XRunBreak information: False

[Detailed description]

- This function configures XRunBreak information.
- The XRunBreak calling interval depends on the simulator.
- Register the Python function that is processed after the specified time passes. See “Hook“ for detail.

Caution If you use the following operations while program is running after the XRunBreak information is set, please use these operations after program is stopped.
- Resets the CPU
- Resets the CPU and then executes the program from the reset address
- Set/Remove Breakpoints

[Example of use]

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

debugger.XTime

This function displays timing information between Go and Break.

[Specification format]

```
debugger.XTime()
```

[Argument(s)]

None

[Return value]

List of timing information (see the XTimeInfo property for detail)

[Detailed description]

- This function displays timing information between Go and Break in nanoseconds.

[Example of use]

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

debugger.XTrace.Clear

This function clears the trace memory. [IECUBE][IECUBE2][Simulator]

[Specification format]

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

[Argument(s)]

None

[Return value]

If the trace memory was cleared successfully: True
If there was an error when clearing the trace memory: False

[Detailed description]

- This function clears the trace memory.

[Example of use]

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

debugger.XTrace.Dump

This function dumps the trace data. [IECUBE][IECUBE2][Simulator]

[Specification format]

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

[Argument(s)]

Argument	Description
frameCount	Specify the number of dumps.
fileName	Specify the name of the file to dump to (default: not specified).
append	Specify whether to append trace data to the file. True: Append trace data to the file. False: Do not append trace data to the file (default).

[Return value]

List of trace information (see the TraceInfo property for detail)

[Detailed description]

- This function dumps trace data for the number of frames specified by *frameCount*.
- If *fileName* is specified, then the trace data is written to the file.
- If *append* is set to "True", then the trace data is appended to the file.

[Example of use]

```
>>>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")
>>>
```

TraceInfo.CreateOtherDict

This function converts the value of TraceInfo.Other into the dict type. [IECUBE] [IECUBE2] [Simulator]

[Specification format]

```
traceInfo.CreateOtherDict()
```

[Argument(s)]

None

[Return value]

Object produced by converting the value of TraceInfo.Other into the dict type (for details of TraceInfo.Other, see the description of the TraceInfoTraceInfo class.)

[Detailed description]

This function converts the value of TraceInfo.Other into the dict type.

[Example of use]

```
>>>info = debugger.Trace.Get(1)
     1853 00h00min00s003ms702µs000ns 0x000003c2 jarl _errfunc, lp
>>>print info[0].Other
Guest,GPID=0,SPID=2
>>>print info[0].CreateOtherDict()
{'SPID': '2', 'GPID': '0', 'Guest': ''}
```

B.3.6 CS+ Python class

Below is a list of CS+ Python classes.

Table B.6 CS+ Python Class

Class Name	Function Description
ActionEventCondition	This class creates an action event condition.
ActionEventInfo	This class holds action event information.
ActionInfo	This class holds result information of the action event.
BankedRegisterInfo	This class holds information on the save register bank.
BreakCondition	This class creates a break condition.
BreakpointInfo	This class holds break point information.
BuildCompletedEventArgs	This class holds the parameters when a build completes.
CurrentConsumptionInfo	This class holds information of current consumption data.
DisassembleInfo	This class holds disassembly information.
DownloadCondition	This class creates conditions of the download file.
DownloadInfo	This class holds download information.
FunctionInfo	This class holds function information.
IORInfo	This class holds IOR and SFR information.
MapInfo	This class holds map information.
PerformanceCondition	This class creates conditions of performance measurement.
PerformanceEventInfo	This class holds performance measurement event information.
PerformanceInfo	This class holds performance measurement information.
PseudoErrorCondition	This class creates a pseudo-error condition.
PseudoErrorInfo	This class holds ECM error information.
PseudoTimerCondition	This class creates conditions of pseudo-timer.
PseudoTimerInfo	This class holds pseudo-timer information.
SoftwareTraceEventInfo	This class holds software trace event information.
SoftwareTraceInfo	This class holds software trace information or software trace (LPD output) information.
SoftwareTraceLPDEventInfo	This class holds software trace (LPD output) event information.
StackInfo	This class holds stack information.
TimerCondition	This class creates conditions of a conditional timer.
TimerEventInfo	This class holds conditional timer event information.
TimerInfo	This class holds conditional timer information.
TraceCondition	This class creates conditions of a conditional trace.
TraceEventInfo	This class holds conditional trace event information.
TraceInfo	This class holds trace information.
VariableInfo	This class holds variable information.

Class Name	Function Description
XRunBreakInfo	This class holds XRunBreak information.
XTimeInfo	This class holds timer information.

ActionEventCondition

This class creates an action event condition.

[Type]

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

[Variable]

Variable	ActionEventType Specification	Description
`Address`	ActionEventType.Printf	Specify an address of an action event. Must be specified.
	ActionEventType.Interrupt	Specify an address of an action event. Must be specified.
`Output`	ActionEventType.Printf	Specify a string to be attached at output.
	ActionEventType.Interrupt	Ignored.
`Expression`	ActionEventType.Printf	Specify a variable expression. Up to ten can be specified by delimiting them with a comma.
	ActionEventType.Interrupt	Ignored.
`Vector`	ActionEventType.Printf	Ignored.
	ActionEventType.Interrupt	Specify the interrupt vector number. [RX Simulator] Specify a value between the range from 0 to 255.
`Priority`	ActionEventType.Printf	Ignored.
	ActionEventType.Interrupt	Specify the interrupt priority. [RX Simulator] Specify a value between the range from 0 to 255. The specifiable range differs for each series. See "CS+ Integrated Development Environment User's Manual: RX Debug Tool" for details.
`ActionEventType`	Specify the action event type. The break types that can be specified are shown below.	
	Type	Description
	ActionEventType.Printf	Printf event (default)
	ActionEventType.Interrupt	Interrupt event

[Detailed description]

- "ActionEventCondition" is in class format, and the action event condition is set in the variable.
  In order to create an action event condition, create an instance, and set conditions for that instance.

[Example of use]

```
>>>ae = ActionEventCondition()                  ...Printf event
>>>ae.Address = 0x3000
>>>ae.Output = "chData = "
>>>ae.Expression = "chData"
>>>ae.ActionEventType = ActionEventType.Printf
>>>debugger.ActionEvent.Set(ae)
1
>>>
>>>ae = ActionEventCondition()                  ...Interrupt event
>>>ae.Address = 0x4000
>>>ae.Vector = 10
>>>ae.Priority = 2
>>>ae.ActionEventType = ActionEventType.Interrupt
>>>debugger.ActionEvent.Set(ae)
2
>>>
```

ActionEventInfo

This class holds action event information (return value of the debugger.ActionEvent.Information function).

[Type]

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

[Variable]

Variable	Description
`Number`	This holds the action event number.
`Name`	This holds the name of the action event.
`Enable`	This holds whether the action event is enabled or not. True: Enabled False: Disabled
`Address`	This holds the address of the action event.
`Output`	This holds the string to be attached at output. **Caution** This should be referenced only when ActionEventType is ActionEventType.Printf.
`Expression`	This holds the variable expression (string). **Caution** This should be referenced only when ActionEventType is ActionEventType.Printf.
`Vector`	This holds the interrupt vector number (numerical value). **Caution** This should be referenced only when ActionEventType is ActionEventType.Interrupt.
`Priority`	This holds the interrupt priority (numerical value). **Caution** This should be referenced only when ActionEventType is ActionEventType.Interrupt.
`ActionEventType`	This holds the type of the action event. Type: Description ActionEventType.Printf: Printf event ActionEventType.Interrupt: Interrupt event

[Detailed description]

- ActionEventInfo is a class, and it is passed as the return value when the debugger.ActionEvent.Information function is executed.

[Example of use]

```
>>>info = debugger.ActionEvent.Information()
1 Python Action Event0001 Enable main - sub
>>>print info[0].Number
1
>>>print info[0].Name
Python Action Event0001
>>>print info[0].Enable
True
>>>
```

ActionInfo

This class holds result information of the action event (return value of the debugger.ActionEvent.Get function).

[Type]

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

[Variable]

Variable	Description	
Number	This holds the action event number (numerical value).	
Name	This holds the name of the action event (string).	
Address	This holds the address of the action event.	
Output	This holds the string to be attached at output.	
Expression	This holds the variable expression (string).	
ActionEventType	This holds the type of the action event.	
	Type	Description
	ActionEventType.Printf	Printf event
HostDate	This holds the time in the host PC when an action event occurred. Take account of the time being that in the host PC.	

[Detailed description]

- ActionInfo is a class, and it is passed as the return value when the debugger.ActionEvent.Get function is executed.

[Example of use]

```
>>>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
```

BankedRegisterInfo

This class holds information on the save register bank (return value of the debugger.SaveRegisterBank.Information function). [RX]

[Type]

```
class BankedRegisterInfo:
        BankNumber = ""
        RegisterName = ""
        Value = ""
```

[Variable]

Variable	Description
BankNumber	This holds the bank number.
RegisterName	This holds the name of the register.
Value	This holds the value.

[Detailed description]

- BankedRegisterInfo is a class, and it is passed as the return value when the debugger.SaveRegisterBank.Information function is executed.

[Example of use]

```
>>> srb = debugger.SaveRegisterBank.Information([1, 3])

Save register bank 1
R1 0x00000000
R2 0x00000000
...
ACC0 0x000000000000000000
ACC1 0x000000000000000000
Save register bank 3
R1 0x00000000
R2 0x00000000
...
ACC0 0x000000000000000000
ACC1 0x000000000000000000
--------------
>>> print srb[0].BankNumber
1
>>> print srb[0].RegisterName
R1
>>> print srb[0].Value
0
```

BreakCondition

This class creates a break condition.

[Type]

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

[Variable]

Variable	Description
`Address`	Specify the address at which to set a break. Must be specified.
`Data`	Specify the number to set as a break condition for the data. If "None" is specified, then the data condition is ignored.
`AccessSize`	Specify the access size (8, 16, 32, or 64). If "None" is specified, then all access sizes will be specified.
`BreakType`	Specify the break type. The break types that can be specified are shown below.

Type	Description
BreakType.Software	Software break (except a simulator)
BreakType.Hardware	Hardware break (default)
BreakType.Read	Data read break
BreakType.Write	Data write break
BreakType.Access	Data access break

[Detailed description]

- "BreakCondition" is in class format, and the break condition is set in the variable.
  In order to create a break condition, create an instance, and set conditions for that instance.

[Example of use]

```
>>>executeBreak = BreakCondition()          ... Create instance
>>>executeBreak.Address = "main"
>>>executeBreak.BreakType = BreakType.Software
>>>debugger.Breakpoint.Set(executeBreak)    ... Specify function in which to set the
break point in parameter
>>>
>>>dataBreak = BreakCondition()             ... Create instance
>>>dataBreak.Address = "chData"
>>>dataBreak.Data = 0x10
>>>dataBreak.BreakType = BreakType.Access
>>>debugger.Breakpoint.Set(dataBreak)       ... Specify function in which to set the
break point in parameter
>>>
>>>executeBreak.Address = "sub + 0x10"      ... Reuse break condition
>>>debugger.Breakpoint.Set(executeBreak)    ... Specify function in which to set the
break point in parameter
>>>
```

BreakpointInfo

This class holds break point information (return value of the debugger.Breakpoint.Information function).

[Type]

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

[Variable]

Variable	Description	
`Number`	This holds the event number.	
`Name`	This holds the name of the break point.	
`Enable`	This holds whether the break point is enabled or not. True: Enabled False: Disabled	
`BreakType`	This holds the break type.	
	Type	Description
	BreakType.Software	Software break (except a simulator)
	BreakType.Hardware	Hardware break
	BreakType.Read	Data read break
	BreakType.Write	Data write break
`Address1`	This holds address information 1 as a string.	
`Address2`	This holds address information 2 as a string (Only for combined breaks).	
`Address3`	This holds address information 3 as a string (Only for combined breaks).	
`Address4`	This holds address information 4 as a string (Only for combined breaks).	

[Detailed description]

- BreakpointInfo is a class, and it is passed as the return value when the debugger.Breakpoint.Information function is executed.

[Example of use]

```
>>>info = debugger.Breakpoint.Information()
   1 Break0001 Enable  test1.c#_main+2
   2 Break0002 Disable test2.c#_sub4+10
>>>print info[0].Number
1
>>>print info[0].Name
Break0001
>>>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
Break0002
>>>print info[1].BreakType
Hardware
>>>print info[1].Enable
False
>>>print info[1].Address1
test2.c#_sub4+10
>>>print info[1].Address2
None
>>>
```

BuildCompletedEventArgs

This class holds the parameters when a build completes.

[Type]

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

[Variable]

Variable	Description
`Error`	When an exception occurs in the build, this holds the error contents (System.Exception).
`Cancelled`	This holds whether the build execution was canceled or not.
`HasBuildError`	This holds whether an error occurred in the build or not.
`HasBuildWarning`	This holds whether a warning occurred in the build or not.

[Detailed description]

- BreakCompletedEventArgs is a class, and it is passed as the argument only when the build.BuildCompleted event is issued.
  It is not therefore possible to generate an instance of this class.

[Example of use]

```
>>>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   ... Event connection
>>>build.All(True)
Error = None
BuildError = False
BuildWarning = False
BuildCancelled = False
True
>>>                              ... When an exception occurs, displayed as follows
>>>build.All(True)
Error = System.Exception:An error occurred during build.(E0203001)
BuildError = False
BuildWarning = False
BuildCancelled = False
False
>>>
>>>                              ... When a build error occurs, displayed as follows
>>>build.All(True)
Error = None
```

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

CurrentConsumptionInfo

This class holds information of current consumption data (return value of the debugger.CurrentConsumption.Get function). [RL78 (devices with support for peripheral function simulation)] [Simulator]

[Type]

```
class CurrentConsumptionInfo:
        Max = 0
        Average = 0
        Count = 0
        ModuleNames = []
```

[Variable]

Variable	Description
Max	This holds the maximum current (uA).
Average	This holds the average current (uA).
Count	This holds the count of current consumption data.
ModuleNames	This holds a list of the peripheral modules for which measurement was performed.

[Detailed description]

- CurrentConsumptionInfo is a class, and it is passed as the return value when the debugger.CurrentConsumption.Get function is executed.

[Example of use]

```
>>>ci = debugger.CurrentConsumption.Get()
Max = 120.20, Average = 30.20
>>>print ci.Max
120.20
>>>print ci.Count
3020
>>>
```

DisassembleInfo

This class holds disassembly information (return value of the debugger.Assemble.Disassemble function).

[Type]

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

[Variable]

Variable	Description
Address	This holds the address.
Code	This holds code information as a collection of bytes.
Mnemonic	This holds mnemonic information.

[Detailed description]

- DisassembleInfo is a class, and it is the structure of the return value from the debugger.Assemble.Disassemble function.

[Example of use]

```
>>>info = debugger.Assemble.Disassemble("main", 4)      ...Disassemble command
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]
>>>
```

DownloadCondition

This class creates conditions of the download file (parameters of the debugger.Download.Property property).

[Type]

```
class DownloadCondition:
        FileName = ""
        DownloadFileType = DownloadFileType.LoadModule
        DownloadObject = True
        DownloadSymbol = True
        VendorType = VendorType.Auto
        OutputInputCorrection = True
```

[Variable]

<table>
<tr><th>Variable</th><th colspan="2">Description</th></tr>
<tr><td>FileName</td><td colspan="2">Specify the full path of the download file.</td></tr>
<tr><td rowspan="6">DownloadFileType</td><td colspan="2">Specify the type of the download file.
The types that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>DownloadFileType.LoadModule</td><td>Load module file (default)</td></tr>
<tr><td>DownloadFileType.Hex</td><td>Intel HEX file</td></tr>
<tr><td>DownloadFileType.SRecord</td><td>Motorola S-record file</td></tr>
<tr><td>DownloadFileType.Binary</td><td>Binary file</td></tr>
<tr><td>DownloadObject</td><td colspan="2">Specify whether to download object information.
True: Download object information.
False: Do not download object information.</td></tr>
<tr><td>DownloadSymbol</td><td colspan="2">This holds whether to download symbol information.
True: Download symbol information.
False: Do not download symbol information.</td></tr>
<tr><td rowspan="4">VendorType</td><td colspan="2">Specify the vendor of the compiler.
The types that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>VendorType.Auto</td><td>Automatically specify the vendor of the compiler judging from the output contents of debugging information (default).</td></tr>
<tr><td>VendorType.Ghs</td><td>Make this specification when using a compiler made by Green Hills Software, Inc.</td></tr>
<tr><td>OutputInputCorrection</td><td colspan="2">Specify whether to generate information for the input correction feature.
True: Generate information for the input correction feature.
False: Do not generate information for the input correction feature.</td></tr>
</table>

[Detailed description]

- "DownloadCondition" is in class format, and it is the structure of the parameter of the debugger.Download.Property property.

[Example of use]

```
>>>di = debugger.Download.Property
>>>print di[0].FileName
C:\project\test.abs
>>>print di[0].DownloadFileType
LoadModule
>>>dc = DownloadCondition()
>>>dc.FileName = "C:/project/test2.abs"
>>>dc.DownloadFileType = DownloadFileType.LoadModule
>>>di.Add(dc)
>>>debugger.Download.Property = di
>>>
```

DownloadInfo

This class holds download information (return value of the debugger.Download.Information function).

[Type]

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

[Variable]

Variable	Description
`Number`	This holds the download number.
`Name`	This holds the file name.
`ObjectDownload`	This holds whether object information has been downloaded or not. True: Object information has been downloaded. False: Object information has not been downloaded.
`SymbolDownload`	This holds whether symbol information has been downloaded or not. True: Symbol information has been downloaded. False: Symbol information has not been downloaded.

[Detailed description]

- DownloadInfo is a class, and it is the structure of the return value from the debugger.Download.Information function.

[Example of use]

```
>>>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

This class holds function information (return value of the project.GetFunctionList function).

[Type]

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

[Variable]

Variable	Description
FunctionName	This holds the function name.
FileName	This holds the full path of the file that the function is defined.
ReturnType	This holds the type of the return value.
StartAddress	This holds the start address of the function.
EndAddress	This holds the end address of the function.

[Detailed description]

- FunctionInfo is a class, and it is the structure of the return value from the project.GetFunctionList function.

[Example of use]

```
>>>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

This class holds IOR and SFR information (return value of the debugger.GetIORList function).

[Type]

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

[Variable]

Variable	Description
IORName	This holds the name of IOR or SFR.
Value	This holds the value.
Type	This holds the type.
Size	This holds the size. The number of bytes is held when the unit of the size is bytes and the number of bits (bits) is held when the unit of the size is bits.
Address	This holds the address
Category	This holds the category.

[Detailed description]

- IORInfo is a class, and it is passed as the return value when the debugger.GetIORList function is executed.

[Example of use]

```
>>> 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
```

MapInfo

This class holds map information (return value of the debugger.Map.Information function).

[Type]

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

[Variable]

Variable	Description
Number	This holds the number.
StartAddress	This holds the start address of the map area.
EndAddress	This holds the end address of the map area.
AccessSize	This holds the access size of the map area.
MapTypeName	This holds the type name of the map area.

[Detailed description]

- MapInfo is a class, and it is the structure of the return value from the debugger.Map.Information function.

[Example of use]

```
>>>info = debugger.Map.Information()    ...Execute Map.Information function
   1: 0x00000000 0x0003FFFF 32 (Internal ROM area)
   2: 0x00040000 0x00048FFF  8 (Non map area)
   3: 0x00049000 0x001003FF  8 (Emulation ROM area)
   4: 0x00100400 0x03FF8FFF  8 (Non map area)
   5: 0x03FF9000 0x03FFEFFF 32 (Internal RAM area)
   6: 0x03FFF000 0x03FFFFFF  8 (I/O register area)
>>>print info[0].StartAddress
0
>>>print info[0].EndAddress
262143
>>>print info[0].AccessSize
32
>>>print info[0].MapTypeName
Internal ROM area
>>>print info[5].StartAddress
67104768
>>>print info[5].EndAddress
67108863
>>>print info[5].AccessSize
8
>>>print info[5].MapTypeName
I/O register area
>>>
```

PerformanceCondition

This class creates conditions of performance measurement. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Type]

```
class PerformanceCondition:
        StartAddress = ""
        StartData = ""
        StartPerformanceType = PerformanceType.Execution
        EndAddress = ""
        EndData = ""
        EndPerformanceType = PerformanceType.Execution
        PerformanceMode = PerformanceMode.MaxCount
        PerformanceItem = PerformanceItem.AllFetchCall
```

[Variable]

Variable	Description	
`StartAddress`	Specify an address starting performance measurement.	
`StartData`	Specify a data condition (number) of an address starting performance measurement. This is valid only when the condition of performance measurement is data access.	
`StartPerformanceType`	Specify the type which start performance measurement. The types that can be specified are shown below.	
	Type	Description
	PerformanceType.Execution	Start/end performance measurement at execution (default)
	PerformanceType.Read	Start/end performance measurement at data read
	PerformanceType.Write	Start/end performance measurement at data write
	PerformanceType.Access	Start/end performance measurement at data access
`EndAddress`	Specify the type which end performance measurement.	
`EndData`	Specify a data condition (number) of an address ending performance measurement. This is valid only when the condition of performance measurement is data access.	
`EndPerformanceType`	Specify the type which end performance measurement. The types that can be specified are shown below.	
	Type	Description
	PerformanceType.Execution	Start/end performance measurement at execution (default)
	PerformanceType.Read	Start/end performance measurement at data read
	PerformanceType.Write	Start/end performance measurement at data write
	PerformanceType.Access	Start/end performance measurement at data access

Variable	Description	
`PerformanceMode`	Specify the mode for performance measurement. The modes that can be specified are shown below.	
	Mode	Description
	PerformanceMode.PassCount	Pass count
	PerformanceMode.NewCount	New count
	PerformanceMode.MinCount	Minium count
	PerformanceMode.MaxCount	Maximum count
	PerformanceMode.AddCount	Total count

Variable	Description	
`PerformanceItem`	Specify performance measurement items. The items that can be specified are shown below.	
	Item	Description
	PerformanceItem.FlashRomDataRequest	Flash ROM data request count [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH]
	PerformanceItem.CpuFetchRequestHit	Response count for CPU issued instruction fetch request
	PerformanceItem.CpuFetchRequest	CPU issued instruction fetch request count
	PerformanceItem.DisableInterruptCycle	Interrupt disable cycle of DI/EI
	PerformanceItem.NoInterruptCycle	Non-interrupt cycle
	PerformanceItem.ClockCycle	Clock cycle
	PerformanceItem.StallCycle	Stall cycle of instructions issued to the instruction execution unit [RH850G4MH]
	PerformanceItem.ALLInstructionSyncException	All instruction sync exception count
	PerformanceItem.AllInstructionAsyncException	All instruction async exception count
	PerformanceItem.FetchFELevelInterrupt	FE level interrupt count
	PerformanceItem.FetchEILevelInterrupt	EI level interrupt count
	PerformanceItem.BranchPredictionMiss	Number of errors in branch prediction for conditional branch instructions (for Bcond and Loop instructions) [RH850G4MH]
	PerformanceItem.FetchBcondLoop	Number of executed conditional branch instructions (for Bcond and Loop instructions) [RH850G4MH]
	PerformanceItem.FetchBranch	Number of executed branch instructions (except for Bcond, Loop, and exception instructions for which the conditions were not matched) [RH850G4MH]
	PerformanceItem.AllFetchBranch	Branch instruction count [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH]
	PerformanceItem.AllFetchCall	All instruction count
	PerformanceItem.BackgroundInterrupt	Background interrupt count
	PerformanceItem.BackgroundEILevelInterrupt	Background EI level interrupt count
	PerformanceItem.BackgroundFELevelInterrupt	Background FE level interrupt count
	PerformanceItem.BackgroundInstructionAsyncException	Background instruction async exception count

[Detailed description]

- "PerformanceCondition" is in class format, and the condition of performance measurement is set in the variable.
  In order to create a condition of performance measurement, create an instance, and set conditions for that instance.

PerformanceEventInfo

This class holds performance measurement event information (return value of the debugger.Performance.Information function). [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Type]

```
class PerformanceEventInfo:
        Number = 0
        Name = ""
        Enable = False
        StartAddress = ""
        StartData = ""
        StartPerformanceType = PerformanceType.Execution
        EndAddress = ""
        EndData = ""
        EndPerformanceType = PerformanceType.Execution
        PerformanceMode = PerformanceMode.MaxCount
        PerformanceItem = PerformanceItem.AllFetchCall
```

[Variable]

Variable	Description	
`Number`	This holds the performance measurement event number.	
`Name`	This holds the name of the performance measurement.	
`Enable`	This holds whether performance measurement is enabled or not. True: Enabled False: Disabled	
`StartAddress`	This holds an address starting performance measurement.	
`StartData`	This holds a data condition (number) of an address starting performance measurement.	
`StartPerformanceType`	This holds the type which start performance measurement.	
	Type	Description
	PerformanceType.Execution	Start/end performance measurement at execution
	PerformanceType.Read	Start/end performance measurement at data read
	PerformanceType.Write	Start/end performance measurement at data write
	PerformanceType.Access	Start/end performance measurement at data access
`EndAddress`	This holds the type which end performance measurement.	
`EndData`	This holds a data condition (number) of an address ending performance measurement.	

Variable	Description	
`EndPerformanceType`	This holds the type which end performance measurement.	
	Type	Description
	PerformanceType.Execution	Start/end performance measurement at execution
	PerformanceType.Read	Start/end performance measurement at data read
	PerformanceType.Write	Start/end performance measurement at data write
	PerformanceType.Access	Start/end performance measurement at data access
`PerformanceMode`	This holds the mode for performance measurement.	
	Mode	Description
	PerformanceMode.PassCount	Pass count
	PerformanceMode.NewCount	New count
	PerformanceMode.MinCount	Minium count
	PerformanceMode.MaxCount	Maximum count
	PerformanceMode.AddCount	Total count

Variable	Description	
`PerformanceItem`	This holds performance measurement items.	
	Item	Description
	PerformanceItem.FlashRomDataRequest	Flash ROM data request count [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH]
	PerformanceItem.CpuFetchRequestHit	Response count for CPU issued instruction fetch request
	PerformanceItem.CpuFetchRequest	CPU issued instruction fetch request count
	PerformanceItem.DisableInterruptCycle	Interrupt disable cycle of DI/EI
	PerformanceItem.NoInterruptCycle	Non-interrupt cycle
	PerformanceItem.ClockCycle	Clock cycle
	PerformanceItem.StallCycle	Stall cycle of instructions issued to the instruction execution unit [RH850G4MH]
	PerformanceItem.ALLInstructionSyncException	All instruction sync exception count
	PerformanceItem.AllInstructionAsyncException	All instruction async exception count
	PerformanceItem.FetchFELevelInterrupt	FE level interrupt count
	PerformanceItem.FetchEILevelInterrupt	EI level interrupt count
	PerformanceItem.BranchPredictionMiss	Number of errors in branch prediction for conditional branch instructions (for Bcond and Loop instructions) [RH850G4MH]
	PerformanceItem.FetchBcondLoop	Number of executed conditional branch instructions (for Bcond and Loop instructions) [RH850G4MH]
	PerformanceItem.FetchBranch	Number of executed branch instructions (except for Bcond, Loop, and exception instructions for which the conditions were not matched) [RH850G4MH]
	PerformanceItem.AllFetchBranch	Branch instruction count [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH]
	PerformanceItem.AllFetchCall	All instruction count
	PerformanceItem.BackgroundInterrupt	Background interrupt count
	PerformanceItem.BackgroundEILevelInterrupt	Background EI level interrupt count
	PerformanceItem.BackgroundFELevelInterrupt	Background FE level interrupt count
	PerformanceItem.BackgroundInstructionAsyncException	Background instruction async exception count

[Detailed description]

- TimerEventInfo is a class, and it is passed as the return value when the debugger.Performance.Information function is executed.

PerformanceInfo

This class holds performance measurement information (return value of the debugger.Performance.Get function).
[RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Type]

```
class PerformanceInfo:
        Number = 0
        Count = 0
        Mode = PerformanceMode.MaxCount
        Item = PerformanceItem.AllFetchCall
        IsOverflow = False
```

[Variable]

Variable	Description	
`Number`	This holds the performance measurement event number.	
`Count`	This holds the number of times it was counted.	
`PerformanceMode`	This holds the mode for performance measurement.	
	Mode	Description
	PerformanceMode.PassCount	Pass count
	PerformanceMode.NewCount	New count
	PerformanceMode.MinCount	Min count
	PerformanceMode.MaxCount	Max count
	PerformanceMode.AddCount	Total count

Variable	Description	
`PerformanceItem`	This holds performance measurement items.	
	Item	Description
	PerformanceItem.FlashRomDataRequest	Flash ROM data request count [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH]
	PerformanceItem.CpuFetchRequestHit	Response count for CPU issued instruction fetch request
	PerformanceItem.CpuFetchRequest	CPU issued instruction fetch request count
	PerformanceItem.DisableInterruptCycle	Interrupt disable cycle of DI/EI
	PerformanceItem.NoInterruptCycle	Non-interrupt cycle
	PerformanceItem.ClockCycle	Clock cycle
	PerformanceItem.StallCycle	Stall cycle of instructions issued to the instruction execution unit [RH850G4MH]
	PerformanceItem.ALLInstructionSyncException	All instruction sync exception count
	PerformanceItem.AllInstructionAsyncException	All instruction async exception count
	PerformanceItem.FetchFELevelInterrupt	FE level interrupt count
	PerformanceItem.FetchEILevelInterrupt	EI level interrupt count
	PerformanceItem.BranchPredictionMiss	Number of errors in branch prediction for conditional branch instructions (for Bcond and Loop instructions) [RH850G4MH]
	PerformanceItem.FetchBcondLoop	Number of executed conditional branch instructions (for Bcond and Loop instructions) [RH850G4MH]
	PerformanceItem.FetchBranch	Number of executed branch instructions (except for Bcond, Loop, and exception instructions for which the conditions were not matched) [RH850G4MH]
	PerformanceItem.AllFetchBranch	Branch instruction count [RH850G3M, RH850G3K, RH850G3MH, RH850G3KH]
	PerformanceItem.AllFetchCall	All instruction count
	PerformanceItem.BackgroundInterrupt	Background interrupt count
	PerformanceItem.BackgroundEILevelInterrupt	Background EI level interrupt count
	PerformanceItem.BackgroundFELevelInterrupt	Background FE level interrupt count
	PerformanceItem.BackgroundInstructionAsyncException	Background instruction async exception count

[Detailed description]

- PerformanceInfo is a class, and it is passed as the return value when the debugger.Performance.Get function is executed.

PseudoErrorCondition

This class creates a pseudo-error condition. [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Type]

```
class PseudoErrorCondition:
        Name = ""
        BitName = ""
        BreakAddress = []
```

[Variable]

Variable	Description
`Name`	Specify the name (abbreviated form) of the error to be generated as a pseudo-error. See the debugger.PseudoError.Get function for the name (abbreviated form) of the error that can be specified.
`BitName`	Specify the name of the IOR bit that will generate a pseudo-error. See the debugger.PseudoError.Get function for the name of the IOR bit that can be specified.
`BreakAddress`	Specify the address to be stopped the program after generating a pseudo-error as a list.

[Detailed description]

- "PseudoErrorCondition" is in class format, and the pseudo-error condition is set in the variable.
  In order to create a pseudo-error condition, create an instance, and set conditions for that instance.

PseudoErrorInfo

This class holds ECM error information (return value of the debugger.PseudoError.Get function). [RH850][E1/E2/E20/Full-spec emulator/IE850A]

[Type]

```
class PseudoErrorInfo:
        Number = ""
        Name = ""
        BitName = ""
        Category = ""
        Error = False
```

[Variable]

Variable	Description
`Number`	This holds the error number.
`Name`	This holds the name (abbreviated form) of the ECM error.
`BitName`	This holds the name of the IOR bit.
`Category`	This holds the category name.
`Error`	This holds information on whether an error was generated. True: An error was generated. False: An error was not generated.

[Detailed description]

Example 1. "PseudoErrorInfo" is a class, and it is passed as the return value when the debugger.PseudoError.Get function is executed.

PseudoTimerCondition

This class creates conditions for pseudo-timer (parameter of debugger.PseudoTimer.Set function). [RL78 instruction simulator]

[Type]

```
class PseudoTimerCondition:
    VectorAddr = None
    Priority = 3
    IntervalTime = 1
    IntervalTimeUnit = IntervalTimeUnit.Ms
    Periodic = False
```

[Variable]

Variable	Description
`VectorAddr`	Specify the vector address (numerical value: 0, 0x4 to 0x7c, string: the name of a maskable interrupt or “Reset”) of the interrupt which is to be generated when the interval time of the pseudo-timer matches the condition. Must be specified.
`Priority`	Specify the interrupt priority (numerical value: 0 to 3).
`IntervalTime`	Specify the interval time for the pseudo-timer (numerical value: 0x1 to 0xFFFFFFFF).
`IntervalTimeUnit`	Specify the unit of the interval time. The following lists the specifiable units.

Type	Description
IntervalTimeUnit.Min	Minutes
IntervalTimeUnit.S	Seconds
IntervalTimeUnit.Ms	Milliseconds
IntervalTimeUnit.Us	Microseconds
IntervalTimeUnit.Ns	Nanoseconds
IntervalTimeUnit.Clock	CPU clock cycles

`Periodic`	Specify whether an interrupt is to be generated each time the specified time elapses. True: An interrupt is generated each time the specified interval elapses. False: An interrupt is only generated once.

[Detailed description]

- "PseudoTimerCondition" is in class format, and the condition of a pseudo timer is set in the variable.
  In order to create a condition of a pseudo timer, create an instance, and set conditions for that instance.

PseudoTimerInfo

This class holds pseudo-timer information (return value of debugger.PseudoTimer.Information function). [RL78 instruction simulator]

[Type]

```
class PseudoTimerInfo:
    Number = None
    VectorAddr = None
    Priority = None
    IntervalTime = None
    IntervalTimeUnit = None
    Periodic = None
```

[Variable]

Variable	Description
`Number`	This holds the pseudo-timer number.
`VectorAddr`	This holds the vector address of the interrupt to be generated when the interval time of the pseudo-timer matches the condition.
`Priority`	This holds the interrupt priority.
`IntervalTime`	This holds the interval time of the pseudo-timer.
`IntervalTimeUnit`	This holds the unit of the interval time. The following lists the specifiable units. Type / Description IntervalTimeUnit.Min / Minutes IntervalTimeUnit.S / Seconds IntervalTimeUnit.Ms / Milliseconds IntervalTimeUnit.Us / Microseconds IntervalTimeUnit.Ns / Nanoseconds IntervalTimeUnit.Clock / CPU clock cycles
`Periodic`	This holds whether an interrupt is to be generated each time the specified time elapses. True: An interrupt is generated each time the specified interval elapses. False: An interrupt is only generated once.

[Detailed description]

- PseudoTimerInfo is a class, and it is passed as the return value when the debugger.PseudoTimer.Information function is executed.

SoftwareTraceEventInfo

This class holds software trace event information (return value of the debugger.SoftwareTrace.Information function). [RH850]

[Type]

```
class SoftwareTraceEventInfo:
        Enable = False
        DBCP = False
        DBTAG = False
        DBPUSH = False
        PC = False
```

[Variable]

Variable	Description
`Enable`	This holds whether the software trace is enabled or not. True: Enabled False: Disabled
`DBCP`	This holds whether to acquire the results of DBCP. True: The results of DBCP is acquired. False: The results of DBCP is not acquired.
`DBTAG`	This holds whether to acquire the results of DBTAG. True: The results of DBTAG is acquired. False: The results of DBTAG is not acquired.
`DBPUSH`	This holds whether to acquire the results of DBPUSH. True: The results of DBPUSH is acquired. False: The results of DBPUSH is not acquired.
`PC`	This holds whether to acquire the program counter. True: The program counter is acquired. False: The program counter is not acquired.

[Detailed description]

- SoftwareTraceEventInfo is a class, and it is passed as the return value when the debugger.SoftwareTrace.Information function is executed.

SoftwareTraceInfo

This class holds software trace information (return value of the debugger.SoftwareTrace.Get function) or software trace (LPD output) information (return value of the debugger.SoftwareTraceLPD.Get function). [RH850]

[Type]

```
class SoftwareTraceInfo:
        FrameNumber = None
        Timestamp = None
        DataType = None
        ProgramCounter = None
        RegisterID = None
        RegisterData = None
        Data = None
        Category = None
        RealData = None
        ProcessorElement = None
        ClockCount = None
```

[Variable]

Variable	Description
FrameNumber	This holds the frame number.
Timestamp	This holds the timestamp. When the microcontroller is multi-core, there is the following difference between emulators and a simulator. [E1/E20/Full-spec emulator] The differential time from the time of the previous data that has the same PE number is held. [Simulator] The accumulated time or differential time is held depending on the setting of debugger.XTrace.Addup. When the differential time is held, it is the difference from the time of the previous data regardless the PE number.
DataType	This holds the type of data. Type / Description SoftwareTraceDataType.DBCP / Check point SoftwareTraceDataType.DBTAG / Tag SoftwareTraceDataType.DBPUSH / Push SoftwareTraceDataType.Lost / Lost data
ProgramCounter	This holds the program counter.
RegisterID	This holds the register ID. (DBPUSH)
RegisterData	This holds the register data. (DBPUSH)
Category	This holds the category. (DBTAG)
Data	This holds the data. (DBTAG)
RealData	This holds the composite data of category and data. (DBTAG)

Variable	Description
ProcessorElement	When the microcontroller is multi-core, this holds the PE number.
ClockCount	This holds the value counted by the clock.

[Detailed description]

- SoftwareTraceInfo is a class, and it is passed as the return value when the debugger.SoftwareTrace.Get or debugger.SoftwareTraceLPD.Get function is executed.

SoftwareTraceLPDEventInfo

This class holds software trace (LPD output) event information (return value of the debugger.SoftwareTraceLPD.Information function). [RH850]

[Type]

```
class SoftwareTraceLPDEventInfo:
        Enable = False
        DBCP = False
        DBTAG = False
        DBPUSH = False
        PC = False
        PE = None
```

[Variable]

Variable	Description
`Enable`	This holds whether the software trace (LPD output) is enabled or not. True: Enabled False: Disabled
`DBCP`	This holds whether to acquire the results of DBCP. True: The results of DBCP is acquired. False: The results of DBCP is not acquired.
`DBTAG`	This holds whether to acquire the results of DBTAG. True: The results of DBTAG is acquired. False: The results of DBTAG is not acquired.
`DBPUSH`	This holds whether to acquire the results of DBPUSH. True: The results of DBPUSH is acquired. False: The results of DBPUSH is not acquired.
`PC`	This holds whether to acquire the program counter. True: The program counter is acquired. False: The program counter is not acquired.
`PE`	When the microcontroller is multi-core, this holds the PE number. When the microcontroller is single-core, this holds "None".

[Detailed description]

- SoftwareTraceLPDEventInfo is a class, and it is passed as the return value when the debugger.SoftwareTraceLPD.Information function is executed.

StackInfo

This class holds stack information (return value of the debugger.Where function).

[Type]

```
class StackInfo:
        Number = 0
        AddressInfoText = None
```

[Variable]

Variable	Description
Number	This holds the stack number.
AddressInfoText	This holds the stack address information as a string.

[Detailed description]

- StackInfo is a class, and it is the structure of the return value from the debugger.Where function.

[Example of use]

```
>>>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

This class creates conditions of a conditional timer.

[Type]

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

[Variable]

<table>
<tr><th>Variable</th><th colspan="2">Description</th></tr>
<tr><td>StartAddress</td><td colspan="2">Specify an address starting timer measurement.
Must be specified.</td></tr>
<tr><td>StartData</td><td colspan="2">Specify a data condition (number) of an address starting timer measurement.
This specification is ignored if “TimerType.Execution” is specified for Start-TimerType.</td></tr>
<tr><td rowspan="5">StartTimerType</td><td colspan="2">Specify the type of timers which start timer measurement.
The types that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>TimerType.Execution</td><td>Start a timer at execution (default)</td></tr>
<tr><td>TimerType.Read</td><td>Start a timer at data read</td></tr>
<tr><td>TimerType.Write</td><td>Start a timer at data write</td></tr>
<tr><td>TimerType.Access</td><td>Start a timer at data access</td></tr>
<tr><td>EndAddress</td><td colspan="2">Specify the type of timers which end timer measurement.
Must be specified.</td></tr>
<tr><td>EndData</td><td colspan="2">Specify a data condition (number) of an address ending timer measurement.
This specification is ignored if “TimerType.Execution” is specified for EndTimer-Type.</td></tr>
<tr><td rowspan="5">EndTimerType</td><td colspan="2">Specify the type of timers which end timer measurement.
The types that can be specified are shown below.</td></tr>
<tr><td>Type</td><td>Description</td></tr>
<tr><td>TimerType.Execution</td><td>End a timer at execution (default)</td></tr>
<tr><td>TimerType.Read</td><td>End a timer at data read</td></tr>
<tr><td>TimerType.Write</td><td>End a timer at data write</td></tr>
<tr><td>TimerType.Access</td><td>End a timer at data access</td></tr>
</table>

[Detailed description]

- "TimerCondition" is in class format, and the condition of a conditional timer is set in the variable.
  In order to create a condition of a conditional timer, create an instance, and set conditions for that instance.

[Example of use]

```
>>>execute_timer = TimerCondition()     ... Create instance
>>>execute_timer.StartAddress = "main"
>>>execute_timer.StartTimerType = TimerType.Execution
>>>execute_timer.EndAddress = "sub"
>>>execute_timer.EndTimerType = TimerType.Execution
>>>debugger.Timer.Set(execute_timer)    ... Specify function in which to set the con-
ditional timer in parameter
1
>>>
```

TimerEventInfo

This class holds conditional timer event information (return value of the debugger.Timer.Information function).

[Type]

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

[Variable]

Variable	Description	
`Number`	This holds the timer event number.	
`Name`	This holds the name of the timer.	
`Enable`	This holds whether the timer is enabled or not. True: Enabled False: Disabled	
`StartAddress`	This holds the address starting timer measurement.	
`StartData`	This holds the data condition (number) of an address starting timer measurement.	
`StartTimerType`	This holds the type of timers which start timer measurement.	
	Type	Description
	TimerType.Execution	Start a timer at execution
	TimerType.Read	Start a timer at data read
	TimerType.Write	Start a timer at data write
	TimerType.Access	Start a timer at data access
`EndAddress`	This holds the address ending timer measurement.	
`EndData`	This holds the data condition (number) of an address ending timer measurement.	
`EndTimerType`	This holds the type of timers which end timer measurement.	
	Type	Description
	TimerType.Execution	End a timer at execution
	TimerType.Read	End a timer at data read
	TimerType.Write	End a timer at data write
	TimerType.Access	End a timer at data access

[Detailed description]

- TimerEventInfo is a class, and it is passed as the return value when the debugger.Timer.Information function is executed.

[Example of use]

```
>>>info = debugger.Timer.Information()
1 PythonTimer0001 Enable main - sub
>>>print info[0].Number
1
>>>print info[0].Name
PythonTimer0001
>>>print info[0].Enable
True
>>>
```

TimerInfo

This class holds conditional timer information (return value of the debugger.Timer.Get function).

[Type]

```
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
```

[Variable]

Variable	Description
`Number`	This holds the timer event number.
`MaxTime`	This holds the maximum execution time.
`MaxClockCount`	This holds the maximum number of clocks to be executed.
`IsMaxOverflow`	This holds whether the maximum execution time or number of clocks was overflowed. True: The maximum execution time or number of clocks was overflowed. False: The maximum execution time or number of clocks was not overflowed.
`MinTime`	This holds the minimum execution time.
`MinClockCount`	This holds the minimum number of clocks to be executed.
`IsMinOverflow`	This holds whether the minimum execution time or number of clocks was overflowed. True: The minimum execution time or number of clocks was overflowed. False: The minimum execution time or number of clocks was not overflowed.
`AverageTime`	This holds the average execution time.
`AverageClockCount`	This holds the average execution number of clocks.
`IsAverageOverflow`	This holds whether the average execution time or number of clocks was overflowed. True: The average execution time or number of clocks was overflowed. False: The average execution time or number of clocks was not overflowed.
`TotalTime`	This holds the total execution time.
`TotalClockCount`	This holds the total execution number of clocks.

Variable	Description
`IsTotalOverflow`	This holds whether the total execution time or number of clocks was overflowed. True: The total execution time or number of clocks was overflowed. False: The total execution time or number of clocks was not overflowed.
`PassCount`	This holds the pass count.
`IsPassCountOverflow`	This holds whether the pass count was overflowed. True: The pass count was overflowed. False: The pass count was not overflowed.

[Detailed description]

- TimerInfo is a class, and it is passed as the return value when the debugger.Timer.Get function is executed.

[Example of use]

```
>>>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

This class creates conditions of a conditional trace.

[Type]

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

[Variable]

Variable	Description	
`StartAddress`	Specify an address starting a trace. Must be specified.	
`StartData`	Specify a data condition (number) of an address starting a trace. This specification is ignored if “TraceType.Execution” is specified for `StartTraceType`.	
`StartTraceType`	Specify the type of timers which start a trace. The types that can be specified are shown below.	
	Type	Description
	TraceType.Execution	Start a trace at execution (default)
	TraceType.Read	Start a trace at data read
	TraceType.Write	Start a trace at data write
	TraceType.Access	Start a trace at data access
`EndAddress`	Specify the type of timers which end a trace. Must be specified.	
`EndData`	Specify a data condition (number) of an address ending a trace. This specification is ignored if “TraceType.Execution” is specified for `EndTraceType`.	
`EndTraceType`	Specify the type of timers which end a trace. The types that can be specified are shown below.	
	Type	Description
	TraceType.Execution	Start a trace at execution (default)
	TraceType.Read	Start a trace at data read
	TraceType.Write	Start a trace at data write
	TraceType.Access	Start a trace at data access

[Detailed description]

- "TraceCondition" is in class format, and the condition of a conditional trace is set in the variable.
  In order to create a condition of a conditional trace, create an instance, and set conditions for that instance.

[Example of use]

```
>>>execute_trace = TraceCondition()     ... Create instance
>>>execute_trace.StartAddress = "main"
>>>execute_trace.StartTraceType = TraceType.Execution
>>>execute_trace.EndAddress = "sub"
>>>execute_trace.EndTraceType = TraceType.Execution
>>>debugger.Trace.Set(execute_trace)    ... Specify function in which to set the con-
ditional trace in parameter
1
>>>
```

TraceEventInfo

This class holds conditional trace event information (return value of the debugger.Trace.Information function).

[Type]

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

[Variable]

Variable	Description	
`Number`	This holds the trace event number.	
`Name`	This holds the name of the trace.	
`Enable`	This holds whether the trace is enabled or not. True: Enabled False: Disabled	
`StartAddress`	This holds an address starting a trace.	
`StartData`	This holds a data condition (number) of an address starting a trace.	
`StartTraceType`	This holds the type of timers which start a trace.	
	Type	Description
	TraceType.Execution	Start a trace at execution
	TraceType.Read	Start a trace at data read
	TraceType.Write	Start a trace at data write
	TraceType.Access	Start a trace at data access
`EndAddress`	This holds an address ending a trace.	
`EndData`	This holds a data condition (number) of an address ending a trace.	
`EndTraceType`	This holds the type of timers which end a trace.	
	Type	Description
	TraceType.Execution	Start a trace at execution
	TraceType.Read	Start a trace at data read
	TraceType.Write	Start a trace at data write
	TraceType.Access	Start a trace at data access

[Detailed description]

- TraceEventInfo is a class, and it is passed as the return value when the debugger.Trace.Information function is executed.

[Example of use]

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

TraceInfo

This class holds trace information (return value of the debugger.XTrace.Dump function).

[Type]

```
class TraceInfo:
        FrameNumber = None
        Timestamp = None
        FetchAddress = None
        Mnemonic = None
        ReadAddress = None
        ReadData = None
        WriteAddress = None
        WriteData = None
        VectorAddress = None
        VectorData = None
        IsDma = True
        ProcessorElement = None
        AccessArea = None
        AccessFactor = None
        AccessID = None
        ClockCount = None
        Other = None
```

[Variable]

Variable	Description
FrameNumber	This holds frame number information.
Timestamp	This holds time stamp information.
FetchAddress	This holds fetch address information.
Mnemonic	This holds mnemonic information.
ReadAddress	This holds read address information.
ReadData	This holds read data information.
WriteAddress	This holds write address information.
WriteData	This holds write data information.
VectorAddress	This holds vector address information.
VectorData	This holds the vector data.
IsDma	This holds whether the data is DMA or not. True: The data is DMA. False: The data is other than DMA.
ProcessorElement	For a multi-core microcontroller, this holds the PE number.
AccessArea	This holds access area information.
AccessFactor	This holds access factor information.
AccessID	This holds access ID information.
ClockCount	This holds the value counted by the clock.

Variable	Description
Other	This holds trace information other than that covered by the variables above.

[Function]

Function	Description
TraceInfo.CreateOtherDict	Converts the value of TraceInfo.Other into the dict type.

[Detailed description]

- TraceInfo is a class, and it is the structure of the return value from the debugger.XTrace.Dump function.

[Example of use]

```
>>>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

This class holds variable information (return value of the project.GetVariableList function).

[Type]

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

[Variable]

Variable	Description
`VariableName`	This holds the variable name.
`FileName`	This holds the full path of the file that the variable is defined.
`Attribute`	This holds the attribute.
`Type`	This holds the type.
`Address`	This holds the address.
`Size`	This holds the size.

[Detailed description]

- VariableInfo is a class, and it is the structure of the return value from the project.GetVariableList function.

[Example of use]

```
>>>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

This class holds XRunBreak information (return value of the debugger.XRunBreak.Refer and debugger.Interrupt.ReferTimer functions).

[Type]

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

[Variable]

Variable	Description	
`Value`	This holds the event interval value.	
`TimeType`	This holds the unit of the interval value.	
	Type	Description
	TimeType.Min	Minute unit
	TimeType.S	Second unit
	TimeType.Ms	Millisecond unit
	TimeType.Us	Microsecond unit
	TimeType.Ns	Nanosecond unit
`IsPeriodic`	This holds whether the callback is used periodically.	

[Detailed description]

- XRunBreakInfo is a class, and it is passed as the return value when the debugger.XRunBreak.Refer or debugger.Interrupt.ReferTimer function is executed.

[Example of use]

```
>>>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

This class holds timer information (return value of the debugger.XTime function).

[Type]

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

[Variable]

Variable	Description
`Value`	This holds the timer measurement.
`IsCpuClock`	This holds whether this is a CPU clock measurement or not. True: This is a CPU clock measurement. False: Otherwise.
`IsOverFlow`	This holds whether an overflow has occurred or not. True: An overflow has occurred. False: An overflow has not occurred.

[Detailed description]

- XTimeInfo is a class, and it is the structure of the return value from the debugger.XTime function.

[Example of use]

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

B.3.7 CS+ Python property (common)

Below is a list of CS+ Python properties (common).

Table B.7 CS+ Python Property (Common)

Property Name	Function Description
common.ExecutePath	This property refers to the absolute path of the folder containing the exe file of the currently running CS+.
common.ConsoleClear	This property sets or refers to whether to clear the display of the Python console when changing the active project.
common.EnableRemotingStartup	This property sets and displays the setting for enabling or disabling the function for linking to an external tool at CS+ startup.
common.Output	This property refers to the return value or the contents of an error of the CS+ Python function.
common.ThrowExcept	This property sets or refers to whether to throw an exception during the Python function is executed.
common.UseRemoting	This property sets and displays the setting for enabling or disabling the function for linking to an external tool at CS+ startup.
common.Version	This property refers to the version of CS+.
common.ViewLine	This property sets or refers to the number of screen lines for the Python console.
common.ViewOutput	This property sets and displays the setting for whether or not to display results of Python functions for CS+ and error messages in the Python console.

common.ExecutePath

This property refers to the absolute path of the folder containing the exe file of the currently running CS+.

[Specification format]

```
common.ExecutePath
```

[Setting(s)]

None

[Reference]

Absolute path of the folder containing the exe file of the currently running CS+

[Detailed description]

- This property refers to the absolute path of the folder containing the exe file (CubeSuiteW+.exe or CubeSuite+.exe) of the currently running CS+.

[Example of use]

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

common.ConsoleClear

This property sets or refers to whether to clear the display of the Python console when changing the active project.

[Specification format]

```
common.ConsoleClear = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to verify during writes. True: Verify during writes. False: Do not verify during writes.

[Reference]

Current set value

[Detailed description]

- This property sets or refers to whether to clear the display of the Python console when changing the active project.

[Example of use]

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

common.EnableRemotingStartup

This property sets and displays the setting for enabling or disabling the function for linking to an external tool at CS+ startup.

[Specification format]

```
common.EnableRemotingStartup = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to enable or disable the function for linking to an external tool at CS+ startup. True: Enable the function for linking to an external tool (default). False: Disable the function for linking to an external tool. Use the common.UseRemoting property to enable or disable linking to an external tool while running.

[Reference]

Current set value

[Detailed description]

- This property sets and displays the setting for enabling or disabling the function for linking to an external tool at CS+ startup.

[Example of use]

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

common.Output

This property refers to the execution result or the contents of an error of the CS+ Python function.

[Specification format]

```
common.Output
```

[Setting(s)]

None

[Reference]

Execution result or an error message of the CS+ Python function (strings)

Caution Error messages can only be referred to when the common.ThrowExcept property is set not to throw an exception (False).

Remark The reference content is retained until the next CS+ Python function call.

[Detailed description]

- This property refers to the execution result or the contents of an error.

[Example of use]

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

common.ThrowExcept

This property sets or refers to whether to throw an exception during the Python function is executed.

[Specification format]

```
common.ThrowExcept = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to throw an exception during the Python function is executed. True: Throw an exception. False: Do not throw an exception (default).

[Reference]

Current set value

[Detailed description]

- This property sets or refers to whether to throw an exception during the Python function is executed.
- To use the try-except statement, set *bool* to "True".

[Example of use]

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

common.UseRemoting

This property sets and displays the setting for enabling or disabling the function for linking to an external tool at CS+ startup.

[Specification format]

```
common.UseRemoting = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to enable or disable the function for linking to an external tool at CS+ startup. True: Enable the function for linking to an external tool (default). False: Disable the function for linking to an external tool. This will be True if the common.EnableRemotingStartup property is set to True on startup, and False otherwise.

[Reference]

Current set value

[Detailed description]

- This property sets and displays the setting for enabling or disabling the function for linking to an external tool at CS+ startup.

[Example of use]

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

common.Version

This property refers to the version of CS+.

[Specification format]

```
common.Version
```

[Setting(s)]

None

[Reference]

Version of CS+

[Detailed description]

- This property refers to the version of CS+.

[Example of use]

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

common.ViewLine

This property sets or refers to the number of screen lines for the Python console.

[Specification format]

```
common.ViewLine = number
```

[Setting(s)]

Setting	Description
number	Set the number of screen lines for the Python console (default: 10000).

[Reference]

Current set value

[Detailed description]

- This property sets or refers to the number of screen lines for the Python console.

[Example of use]

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

common.ViewOutput

This property sets and displays the setting for whether or not to display results of Python functions for CS+ and error messages in the Python console.

[Specification format]

```
common.ViewOutput = bool
```

[Setting(s)]

Setting	Description
bool	Set whether or not to display results of Python functions for CS+ and error messages in the Python console. True: Display in the Python console (default). False: Do not display in the Python console.

[Reference]

Current set value

[Detailed description]

- This property sets and displays the setting for whether or not to display results of Python functions for CS+ and error messages in the Python console.

[Example of use]

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

B.3.8 CS+ Python property (for project)

Below is a list of CS+ Python properties (for a project).

Table B.8 CS+ Python Property (For Project)

Property Name	Function Description
project.Device	This property refers to the microcontroller of the active project.
project.IsOpen	This property confirms whether the project has been opened.
project.Kind	This property refers to the kind of the active project.
project.Name	This property refers to the active project file name (without path).
project.Nickname	This property refers to the nickname of the microcontroller of the active project.
project.Path	This property refers to the active project file name (with path).

project.Device

This property refers to the microcontroller of the active project.

[Specification format]

```
project.Device
```

[Setting(s)]

None

[Reference]

Microcontroller of the active project

[Detailed description]

- This property refers to the microcontroller of the active project.

[Example of use]

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

project.IsOpen

This property confirms whether the project has been opened.

[Specification format]

```
project.IsOpen
```

[Setting(s)]

None

[Reference]

If the project has been opened: True
If the project has not been opened: False

[Detailed description]

- This property confirms whether the project has been opened.

[Example of use]

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

project.Kind

This property refers to the kind of the active project.

[Specification format]

```
project.Kind
```

[Setting(s)]

None

[Reference]

Kind of active project

Type	Description
Application	Project for application
Library	Project for library
DebugOnly	Debug-dedicated project
Empty	Project for empty application
CppApplication	Project for C++ application
RI600V4	Project for RI600V4
RI600PX	Project for RI600PX
RI850V4	Project for RI850V4
RI850MP	Project for RI850MP
RI78V4	Project for RI78V4
MulticoreBootLoader	Project for boot loader for multi-core
MulticoreApplication	Project for application for multi-core

[Detailed description]

- This property refers to the kind of the active project.

[Example of use]

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

project.Name

This property refers to the active project file name (without path).

[Specification format]

```
project.Name
```

[Setting(s)]

None

[Reference]

Active project file name (without path)

[Detailed description]

- This property refers to the active project file name (without path).

[Example of use]

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

project.Nickname

This property refers to the nickname of the microcontroller of the active project.

[Specification format]

```
project.Nickname
```

[Setting(s)]

None

[Reference]

Nickname of the microcontroller of the active project

[Detailed description]

- This property refers to the nickname of the microcontroller of the active project.

[Example of use]

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

project.Path

This property refers to the active project file name (with path).

[Specification format]

```
project.Path
```

[Setting(s)]

None

[Reference]

Active project file name (with path)

[Detailed description]

- This property refers to the active project file name (with path).

[Example of use]

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

B.3.9 CS+ Python property (for build tool)

Below is a list of CS+ Python properties (for the build tool).

Table B.9 CS+ Python Property (For Build Tool)

Property Name	Function Description
build.Assemble.AssembleListFile-OutputFolder	This property is for setting or referring to the folder for the output of assembly listing files. It is an assembly option for the active project.
build.Assemble.OutputAssembleList-File	This property is for setting or referring to whether to output assembly listing files. It is an assembly option for the active project.
build.Common.DataEndian	This property is for setting or referring to the endian of data. It is a common option for the build tool for the active project.
build.Common.IntermediateFileOut-putFolder	This property is for setting or referring to the folder for the output of intermediate files. It is a common option for the build tool for the active project.
build.Common.MergedErrorMes-sageFileOutputFolder	This property is for setting or referring to the folder for the output of error message merge files. It is a common option for the build tool for the active project.
build.Common.MergeErrorMessage-File	This property is for setting or referring to whether to merge error message files. It is a common option for the build tool for the active project.
build.Common.PrecisionOfDouble-Type	This property is for setting or referring to the precision used with double and long double types. It is a common option for the build tool for the active proj-ect.
build.Common.UseDPFPU	This property is for setting or referring to whether to use double-precision floating-point operation instructions. It is an assembly option for the active project.
build.Compile.AdditionalOptions	This property sets or refers to the compile options for the active project regarding other additional options.
build.Compile.AssemblySourceFile-OutputFolder	This property is for setting or referring to the folder for the output of assembly source files. It is a compile option for the active project.
build.Compile.FloatType	This property is for setting or referring to the method of floating-point calcula-tion. It is a compile option for the active project.
build.Compile.IncludePath	This property sets or refers to the compile options for the active project regarding additional include paths.
build.Compile.ListFileOutputFolder	This property is for setting or referring to the folder for the output of assembly listing files. It is a compile option for the active project.
build.Compile.Macro	This property sets or refers to the compile options for the active project regarding defined macros.
build.Compile.OutputAssembly-SourceFile	This property is for setting or referring to whether to output assembly source files. It is a compile option for the active project.
build.Compile.OutputListFile	This property is for setting or referring to whether to output assembly listing files or source listing files. It is a compile option for the active project.
build.Compile.PrecisionOfDouble-Type	This property is for setting or referring to the precision of double type and long double type. It is a compile option for the active project.
build.Compile.PreprocessedSource-FileOutputFolder	This property is for setting or referring to the folder for the output of preprocessed source files. It is a compile option for the active project.
build.HexOutput.OutputFolder	This property is for setting or referring to the folder for hexadecimal output. It is a hexadecimal output option for the active project.

Property Name	Function Description
build.IsBuilding	This property confirms whether a build is running.
build.Link.AdditionalOptions	This property sets or refers to the link options for the active project regarding other additional options.
build.Library.EnableMathfH	This property is for setting or referring to whether or not mathf.h (C89/C99) is enabled. It is a library generate option for the active project.
build.Library.EnableMathH	This property is for setting or referring to whether or not math.h (C89/C99) is enabled. It is a library generate option for the active project.
build.Link.LibraryFile	This property sets or refers to library files of the active project.
build.Link.OutputFolder	This property is for setting or referring to the folder for the output of the results of linkage of the object files from the active project. It is a linker option for the active project.
build.Link.RangeOfDebugMonitorArea	This property is for setting or referring to the range of a debug monitoring area. It is a link option for the active project.
build.Link.SectionAlignment	This property sets or refers to the link options for the active project regarding section alignment.
build.Link.SectionROMtoRAM	This property sets or refers to the link options for the active project regarding sections where symbols are mapped from ROM to RAM.
build.Link.SectionStartAddress	This property sets or refers to the link options for the active project regarding the addresses where sections start.
build.Link.SectionSymbolFile	This property sets or refers to the link options for the active project regarding sections whose external defined symbols are to be output to a file.
build.Link.SetDebugMonitorArea	This property is for setting or referring to whether or not a debug monitoring area is set. It is a link option for the active project.
build.ROMization.OutputObjectFile	This property sets or refers to the setting for output of a ROMized object file, that is, the value of the ROMization process option for the active project.
build.Version	This property is for setting or referring to the version of the compiler package that is in use for the active project.

build.Assemble.AssembleListFileOutputFolder

This property is for setting or referring to the folder for the output of assembly listing files. It is an assembly option for the active project. [CC-RH] [CC-RL]

[Specification format]

```
build.Assemble.AssembleListFileOutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for the output of assembly listing files as strings.

[Reference]

Path to the folder for the output of assembly listing files

[Detailed description]

- This property is for setting or referring to the folder for the output of assembly listing files. It is an assembly option for the active project.

[Example of use]

```
>>>build.Assemble.AssembleListFileOutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.Assemble.AssembleListFileOutputFolder
>>>%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.Assemble.OutputAssembleListFile

This property is for setting or referring to whether to output assembly listing files. It is an assembly option for the active project.

[Specification format]

```
build.Assemble.OutputAssembleListFile = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to output assembly listing files. True: Output assembly listing files. False: Do not output assembly listing files.

[Reference]

If assembly listing files are to be output: True
If assembly listing files are not to be output: False

[Detailed description]

- This property is for setting or referring to whether to output assembly listing files. It is an assembly option for the active project.

[Example of use]

```
>>>build.Assemble.OutputAssembleListFile = True
>>>print build.Assemble.OutputAssembleListFile
True
>>>
```

build.Common.DataEndian

This property is for setting or referring to the endian of data. It is a common option for the build tool for the active project.[CC-RX]

[Specification format]

```
build.Common.DataEndian = endianType
```

[Setting(s)]

Setting	Description	
endianType	Specify the endian of data. The specifiable types are listed below.	
	Type	Description
	EndianType.Big	The byte order of data is big endian.
	EndianType.Little	The byte order of data is little endian.

[Reference]

Set value

[Detailed description]

- This property is for setting or referring to the endian of data. It is a common option for the build tool for the active project.

[Example of use]

```
>>>build.Common.DataEndian = EndianType.Little
>>>print build.Common.DataEndian
Little
>>>
```

build.Common.IntermediateFileOutputFolder

This property is for setting or referring to the folder for the output of intermediate files. It is a common option for the build tool for the active project.

[Specification format]

```
build.Common.IntermediateFileOutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for the output of intermediate files as strings.

[Reference]

Path to the folder for the output of intermediate files

[Detailed description]

- This property is for setting or referring to the folder for the output of intermediate files. It is a common option for the build tool for the active project.

[Example of use]

```
>>>build.Common.IntermediateFileOutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.Common.IntermediateFileOutputFolder
%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.Common.MergedErrorMessageFileOutputFolder

This property is for setting or referring to the folder for the output of error message merge files. It is a common option for the build tool for the active project. [CC-RH] [CC-RL]

[Specification format]

```
build.Common.MergedErrorMessageFileOutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for the output of error message merge files as strings.

[Reference]

Path to the folder for the output of error message merge files

[Detailed description]

- This property is for setting or referring to the folder for the output of error message merge files. It is a common option for the build tool for the active project.

[Example of use]

```
>>>build.Common.MergedErrorMessageFileOutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.Common.MergedErrorMessageFileOutputFolder
%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.Common.MergeErrorMessageFile

This property is for setting or referring to whether to merge error message files. It is a common option for the build tool for the active project. [CC-RH] [CC-RL]

[Specification format]

```
build.Common.MergeErrorMessageFile = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to merge error message files. True: Merge error message files. False: Do not merge error message files.

[Reference]

If error message files are to be merged: True
If error message files are not to be merged: False
If the compiler is not supported: None

[Detailed description]

- This property is for setting or referring to whether to merge error message files. It is a common option for the build tool for the active project.

[Example of use]

```
>>>build.Common.MergeErrorMessageFile = True
>>>print build.Common.MergeErrorMessageFile
True
>>>
```

build.Common.PrecisionOfDoubleType

This property is for setting or referring to the endian of data. It is a common option for the build tool for the active project. [CC-RX]

[Specification format]

```
build.Common.PrecisionOfDoubleType = precision
```

[Setting(s)]

Setting	Description	
precision	Specify the precision of double type and long double type. The following lists the specifiable values.	
	Type	Description
	PrecisionOfType.Single	Handles double type and long double type as single-precision floating-point type (4 bytes).
	PrecisionOfType.Double	Handles double type and long double type as double-precision floating-point type (8 bytes).

[Reference]

Set value

[Detailed description]

- This property is for setting or referring to the endian of data. It is a common option for the build tool for the active project.

[Example of use]

```
>>>build.Common.PrecisionOfDoubleType = PrecisionOfType.Single
>>>print build.Common.PrecisionOfDoubleType
Single
>>>
```

build.Common.UseDPFPU

This property is for setting or referring to whether to use double-precision floating-point operation instructions. It is an assembly option for the active project. [CC-RX]

[Specification format]

```
build.Common.UseDPFPU = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to use double-precision floating-point operation instructions. True: Outputs an object that uses double-precision floating-point operation instructions. False: Does not output an object that uses double-precision floating-point operation instructions.

[Reference]

Use double-precision floating-point operation instructions: True
Does not use double-precision floating-point operation instructions: False

[Detailed description]

- This property is for setting or referring to whether to use double-precision floating-point operation instructions. It is an assembly option for the active project.

[Example of use]

```
>>>build.Common.UseDPFPU = True
>>>print build.Common.UseDPFPU
True
>>>
```

build.Compile.AdditionalOptions

This property sets or refers to the compile options for the active project regarding other additional options.

[Specification format]

```
build.Compile.AdditionalOptions = option
```

[Setting(s)]

Setting	Description
option	Set the additional compile options as strings.

[Reference]

Additional compile options (strings)

[Detailed description]

- This property sets or refers to the compile options for the active project regarding other additional options.
- The options set here are added at the end of the compile options group.

[Example of use]

```
>>>build.Compile.AdditionalOptions = "-o3 -Xvolatile"   ... Set multiple options
>>>print build.Compile.AdditionalOptions
-o3 -Xvolatile
>>>copt = build.Compile.AdditionalOptions + " -v"   ... Refer the current setting and
add an option
>>>build.Compile.AdditionalOptions = copt
>>>print build.Compile.AdditionalOptions
-o3 -Xvolatile -v
>>>
```

build.Compile.AssemblySourceFileOutputFolder

This property is for setting or referring to the folder for the output of assembly source files. It is a compile option for the active project. [CC-RH] [CC-RL]

[Specification format]

```
build.Compile.AssemblySourceFileOutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for the output of assembly source files as strings.

[Reference]

Path to the folder for the output of assembly source files

[Detailed description]

- This property is for setting or referring to the folder for the output of assembly source files. It is a compile option for the active project.

[Example of use]

```
>>>build.Compile.AssemblySourceFileOutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.Compile.AssemblySourceFileOutputFolder
%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.Compile.FloatType

This property is for setting or referring to the method of floating-point calculation. It is a compile option for the active project. [CC-RH]

[Specification format]

```
build.Compile.FloatType = precision
```

[Setting(s)]

Setting	Description	
floatType	Specify the method of floating-point calculation. The following lists the specifiable values.	
	Type	Description
	FloatType.Fpu	Generates floating-point calculation instructions of FPU for floating-point calculations.
	FloatType.Auto	Generates floating-point calculation instructions.
	FloatType.Soft	Generates runtime library call instructions for floating-point calculations.

[Reference]

Set value

[Detailed description]

- This property is for setting or referring to the method of floating-point calculation. It is a compile option for the active project.

[Example of use]

```
>>>build.Compile.FloatType = FloatType.Fpu
>>>print build.Compile.FloatType
Fpu
>>>
```

build.Compile.IncludePath

This property sets or refers to the compile options for the active project regarding additional include paths.

[Specification format]

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

[Setting(s)]

Setting	Description
dirlist	Set the additional include paths as a list of strings.

[Reference]

List of additional include paths

[Detailed description]

- This property sets or refers to the compile options for the active project regarding additional include paths.
- Add or change for the referred list to change the setting.

[Example of use]

```
>>>incpath1 = build.Compile.IncludePath     ... Refer the current setting and add an
include path
>>>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']      ... Set multiple include paths
>>>build.Compile.IncludePath = incpath2
>>>print build.Compile.IncludePath
['include1', 'include2']
```

build.Compile.ListFileOutputFolder

This property is for setting or referring to the folder for the output of assembly listing files. It is a compile option for the active project. [CC-RH] [CC-RL]

[Specification format]

```
build.Compile.ListFileOutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for the output of assembly listing files as strings.

[Reference]

Path to the folder for the output of assembly listing files

[Detailed description]

- This property is for setting or referring to the folder for the output of assembly listing files. It is a compile option for the active project.

[Example of use]

```
>>>build.Compile.ListFileOutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.Compile.ListFileOutputFolder
%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.Compile.Macro

This property sets or refers to the compile options for the active project regarding defined macros.

[Specification format]

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

[Setting(s)]

Setting	Description
macrolist	Set the defined macros as a list of strings.

[Reference]

List of defined macros

[Detailed description]

- This property sets or refers to the compile options for the active project regarding defined macros.
- Add or change for the referred list to change the setting.

[Example of use]

```
>>>macrolist = build.Compile.Macro    ... Refer the current setting and add a defined
macro
>>>print macrolist
['RL78']
>>>macrolist.append('78K')
>>>build.Compile.Macro = macrolist
>>>print build.Compile.Macro
['RL78', '78K']
>>>
>>>macrolist = ['macro1', 'macro2']   ... Set multiple defined macros
>>>build.Compile.Macro = macrolist
>>>print build.Compile.Macro
['macro1', 'macro2']
```

build.Compile.OutputAssemblySourceFile

This property is for setting or referring to whether to output assembly source files. It is a compile option for the active project.

[Specification format]

```
build.Compile.OutputAssemblySourceFile = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to output assembly source files. True: Output assembly source files. False: Do not output assembly source files.

[Reference]

If assembly source files are to be output: True
If assembly source files are not to be output: False

[Detailed description]

- This property is for setting or referring to whether to output assembly source files. It is a compile option for the active project.

[Example of use]

```
>>>build.Compile.OutputAssemblySourceFile = True
>>>print build.Compile.OutputAssemblySourceFile
True
>>>
```

build.Compile.OutputListFile

This property is for setting or referring to whether to output assembly listing files [CC-RH] [CC-RL] or source listing files [CC-RX]. It is a compile option for the active project.

[Specification format]

```
build.Compile.OutputListFile = bool
```

[Setting(s)]

Setting	Description
bool	Set whether to output assembly listing files or source listing files. True: Output assembly listing files or source listing files. False: Do not output assembly listing files or source listing files.

[Reference]

If assembly listing files or source listing files are to be output: True
If assembly listing files or source listing files are not to be output: False

[Detailed description]

- This property is for setting or referring to whether to output assembly listing files or source listing files. It is a compile option for the active project.

[Example of use]

```
>>>build.Compile.OutputListFile = True
>>>print build.Compile.OutputListFile
True
>>>
```

build.Compile.PrecisionOfDoubleType

This property is for setting or referring to the precision of double type and long double type. It is a compile option for the active project. [CC-RH V1.02.00 and later versions]

[Specification format]

```
build.Compile.PrecisionOfDoubleType = precision
```

[Setting(s)]

Setting	Description	
precision	Specify the precision of double type and long double type. The following lists the specifiable values.	
	Type	Description
	PrecisionOfType.Single	Handles double type and long double type as single-precision floating-point type (4 bytes).
	PrecisionOfType.Double	Handles double type and long double type as double-precision floating-point type (8 bytes).

[Reference]

Set value

[Detailed description]

- This property is for setting or referring to the precision of double type and long double type. It is a compile option for the active project.

[Example of use]

```
>>>build.Compile.PrecisionOfDoubleType = PrecisionOfType.Single
>>>print build.Compile.PrecisionOfDoubleType
Single
>>>
```

build.Compile.PreprocessedSourceFileOutputFolder

This property is for setting or referring to the folder for the output of preprocessed source files. It is a compile option for the active project. [CC-RH] [CC-RL]

[Specification format]

```
build.Compile.PreprocessedSourceFileOutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for the output of preprocessed source files as strings.

[Reference]

Path to the folder for the output of preprocessed source files

[Detailed description]

- This property is for setting or referring to the folder for the output of preprocessed source files. It is a compile option for the active project.

[Example of use]

```
>>>build.Compile.PreprocessedSourceFileOutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.Compile.PreprocessedSourceFileOutputFolder
%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.HexOutput.OutputFolder

This property is for setting or referring to the folder for hexadecimal output. It is a hexadecimal output option for the active project.

[Specification format]

```
build.HexOutput.OutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for hexadecimal output as strings.

[Reference]

Path to the folder for hexadecimal output

[Detailed description]

- This property is for setting or referring to the folder for hexadecimal output. It is a hexadecimal output option for the active project.

[Example of use]

```
>>>build.HexOutput.OutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.HexOutput.OutputFolder
%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.IsBuilding

This property confirms whether a build is running.

[Specification format]

```
build.IsBuilding
```

[Setting(s)]

None

[Reference]

If a build is running: True
If a build is not run: False

[Detailed description]

- This property confirms whether a build is running.

[Example of use]

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

build.Link.AdditionalOptions

This property sets or refers to the link options for the active project regarding other additional options.

[Specification format]

```
build.Link.AdditionalOptions = option
```

[Setting(s)]

Setting	Description
option	Set the additional link options as strings.

[Reference]

Additional link options (strings)

[Detailed description]

- This property sets or refers to the link options for the active project regarding other additional options.
- The options set here are added at the end of the link options group.

[Example of use]

```
>>>build.Link.AdditionalOptions = "-stack -Total_size"      ... Set multiple options
>>>print build.Link.AdditionalOptions
-stack -Total_size
>>>lopt = build.Link.AdditionalOptions + " -map=file.bls"   ... Refer the current set-
ting and add an option
>>>build.Link.AdditionalOptions = lopt
>>>print build.Link.AdditionalOptions
-stack -Total_size -map=file.bls
>>>
```

build.Library.EnableMathfH

This property is for setting or referring to whether or not mathf.h (C89/C99) is enabled. It is a library generate option for the active project. [CC-RX]

[Specification format]

```
build.Library.EnableMathfH = bool
```

[Setting(s)]

Setting	Description
bool	Set whether or not mathf.h (C89/C99) is enabled. True: Enables mathf.h (C89/C99) and the runtime library. False: Disables mathf.h (C89/C99).

[Reference]

When mathf.h (C89/C99) and the runtime library are enabled: True
When mathf.h (C89/C99) is disabled: False

[Detailed description]

- This property is for setting or referring to whether or not mathf.h (C89/C99) is enabled. It is a library generate option for the active project.

[Example of use]

```
>>>build.Library.EnableMathfH = True
>>>print build.Library.EnableMathfH
True
>>>
```

build.Library.EnableMathH

This property is for setting or referring to whether or not math.h (C89/C99) is enabled. It is a library generate option for the active project. [CC-RX]

[Specification format]

```
build.Library.EnableMathH = bool
```

[Setting(s)]

Setting	Description
bool	Set whether or not math.h (C89/C99) is enabled. True: Enables math.h (C89/C99) and the runtime library. False: False: Disables math.h (C89/C99).

[Reference]

When math.h (C89/C99) and the runtime library are enabled: True
When math.h (C89/C99) is disabled: False

[Detailed description]

- This property is for setting or referring to whether or not math.h (C89/C99) is enabled. It is a library generate option for the active project.

[Example of use]

```
>>>build.Library.EnableMathH = True
>>>print build.Library.EnableMathH
True
>>>
```

build.Link.LibraryFile

This property sets or refers to library files of the active project.

[Specification format]

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

[Setting(s)]

Setting	Description
filelist	Set the library files of the active project as a list of strings.

[Reference]

List of library files

[Detailed description]

- This property sets or refers to library files of the active project.
- Add or change for the referred list to change the setting.

[Example of use]

```
>>>lib1 = build.Link.LibraryFile    ... Refer the current setting and add a library
file
>>>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']    ... Set multiple library files
>>>build.Link.LibraryFile = lib2
>>>print build.Link.LibraryFile
['test1.lib', 'test2.lib']
```

build.Link.OutputFolder

This property is for setting or referring to the folder for the output of the results of linkage of the object files from the active project. It is a linker option for the active project.

[Specification format]

```
build.Link.OutputFolder = folder
```

[Setting(s)]

Setting	Description
folder	Set the path to the folder for the output of the results of linkage of the object files from the active project as strings.

[Reference]

Path to the folder for output

[Detailed description]

- This property is for setting or referring to the folder for the output of the results of linkage of the object files from the active project. It is a linker option for the active project.

[Example of use]

```
>>>build.Link.OutputFolder = "/ProjectDir/Output_Vx.xx.xx"
>>>print build.Link.OutputFolder
%ProjectDir%\Output_Vx.xx.xx
>>>
```

build.Link.RangeOfDebugMonitorArea

This property is for setting or referring to the range of a debug monitoring area. It is a link option for the active project. [CC-RL]

[Specification format]

```
build.Link.RangeOfDebugMonitorArea = area
```

[Setting(s)]

Setting	Description
area	Specify the range of a debug monitoring area in the format "start address - end address".

[Reference]

Range of a debug monitoring area

[Detailed description]

- This property is for setting or referring to the range of a debug monitoring area. It is a link option for the active project.

Caution This property can only be set or referred to when [Set debug monitor area] (build.Link.SetDebugMonitorArea) has been specified for [Yes(Specify address range)] (DebugMonitorArea.SpecifiedAddressRange).

[Example of use]

```
>>>build.Link.RangeOfDebugMonitorArea = "FE00-FFFF"
>>>print build.Link.RangeOfDebugMonitorArea
FE00-FFFF
>>>
```

build.Link.SectionAlignment

This property sets or refers to the link options for the active project regarding section alignment. [CC-RH][CC-RX][CC-RL]

[Specification format]

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

[Setting(s)]

Setting	Description
sectionlist	Set section alignment as a list of strings.

[Reference]

List of section alignment

[Detailed description]

- This property sets or refers to the link options for the active project regarding section alignment.
- Add or change for the referred list to change the setting.

[Example of use]

```
>>>lib1 = build.Link.LibraryFile      ... Refer the current setting and add section
alignment
['R_1']
>>>sec1.append('R_2')
>>>build.Link.SectionAlignment = sec1
>>>print build.Link.SectionAlignment
['R_1', 'R_2']
>>>
>>>sec2 = ['R_1', 'R_2']              ... Set multiple section alignment
>>>build.Link.SectionAlignment = sec2
>>>print build.Link.SectionAlignment
['R_1', 'R_2']
```

build.Link.SectionROMtoRAM

This property sets or refers to the link options for the active project regarding sections where symbols are mapped from ROM to RAM. [CC-RH][CC-RX][CC-RL]

[Specification format]

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

[Setting(s)]

Setting	Description
sectionlist	Set the section that maps symbols from ROM to RAM as a list of strings.

[Reference]

List of the section that maps symbols from ROM to RAM

[Detailed description]

- This property sets or refers to the link options for the active project regarding sections where symbols are mapped from ROM to RAM.
- Add or change for the referred list to change the setting.

[Example of use]

```
>>>sec = build.Link.SectionROMtoRAM     ... Refer the current setting and add the sec-
tion that maps symbols from ROM to RAM
>>>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

This property sets or refers to the link options for the active project regarding the addresses where sections start. [CC-RH][CC-RX][CC-RL]

[Specification format]

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

[Setting(s)]

Setting	Description
section	Set the start address of the section as strings.

[Reference]

Start address of the section (strings)

[Detailed description]

- This property sets or refers to the link options for the active project regarding the addresses where sections start.
- Add or change for the referred strings to change the setting.

[Example of use]

```
>>>sec= build.Link.SectionStartAddress  ... Refer the current setting and change the
start address of the section
>>>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

This property sets or refers to the link options for the active project regarding sections whose external defined symbols are to be output to a file. [CC-RH][CC-RX][CC-RL]

[Specification format]

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

[Setting(s)]

Setting	Description
sectionlist	Set the section whose external defined symbols are output to a file as a list of strings.

[Reference]

List of the section whose external defined symbols are output to a file

[Detailed description]

- This property sets or refers to the link options for the active project regarding sections whose external defined symbols are to be output to a file.
- Add or change for the referred list to change the setting.

[Example of use]

```
>>>sec = build.Link.SectionSymbolFile   ... Refer the current setting and add the sec-
tion whose external defined symbols are output to a file
>>>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.Link.SetDebugMonitorArea

This property is for setting or referring to whether or not a debug monitoring area is set. It is a link option for the active project. [CC-RL]

[Specification format]

```
build.Link.SetDebugMonitorArea = debugMonitorArea
```

[Setting(s)]

Setting	Description	
debugMonitorArea	Select whether or not a debug monitoring area is set. The specifiable types are listed below.	
	Type	Description
	DebugMonitorArea.Default-AddressRange	Specify a debug monitoring area with the default range.
	DebugMonitorArea.Specifie-dAddressRange	Specify the address range of a debug monitoring area.
	DebugMonitorArea.NotSet	A debug monitoring area is not specified.

[Reference]

Set value

[Detailed description]

- This property is for setting or referring to whether or not a debug monitoring area is set. It is a link option for the active project.

[Example of use]

```
>>>build.Link.SetDebugMonitorArea = DebugMonitorArea.SpecifiedAddressRange
>>>print build.Link.SetDebugMonitorArea
SpecifiedAddressRange
>>>
```

build.ROMization.OutputObjectFile

This property sets or refers to the setting for output of a ROMized object file, that is, the value of the ROMization process option for the active project. [CA850][CX][CA78K0R]

[Specification format]

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

[Setting(s)]

Setting	Description
bool	Set whether or not to output the ROMized object file. True: Output the ROMized object file. False: Do not output the ROMized object file.

[Reference]

If the ROMized object file is output: True
If the ROMized object file is not output: False
If the compiler is not supported: None

[Detailed description]

- This property sets or refers to the setting for output of a ROMized object file, that is, the value of the ROMization process option for the active project.

[Example of use]

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

build.Version

This property is for setting or referring to the version of the compiler package that is in use for the active project.

[Specification format]

```
build.Version = version
```

[Setting(s)]

Setting	Description
version	Set the version of the compiler package that is in use for the active project as strings.

[Reference]

Version of compiler package used in active project

[Detailed description]

- This property is for setting or referring to the version of the compiler package that is in use for the active project.

[Example of use]

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

B.3.10 CS+ Python property (for debug tool)

Below is a list of CS+ Python properties (for the debug tool).

Table B.10 CS+ Python Property (For Debug Tool)

Property Name	Function Description
debugger.ActionEvent.GetLine	This property sets or refers to the number of action event results.
debugger.ADConvertDataInExecution	This property sets or refers to data collected in debugging.
debugger.DebugTool.SerialNumber	This property sets or refers to the serial number of the emulator.
debugger.DebugTool.SerialNumberList	This property refers to the list of serial numbers of emulators.
debugger.Download.Property	This property sets or refers to conditions of the download file of the debug tool.
debugger.Interrupt.ExceptionCause	This property refers to the exception cause code.
debugger.IsMulticore	This property checks whether or not the microcontroller of the active project is multi-core.
debugger.Memory.NoVerify	This property switches the write-time verification setting.
debugger.Option.AccessDuringExecution debugger.Option.AccessStopExecution debugger.Option.AccumulateTraceTime debugger.Option.AfterTraceMemoryFull debugger.Option.Coverage debugger.Option.CpuEndian debugger.Option.MainClockFrequency debugger.Option.OpenBreak debugger.Option.ResetMask debugger.Option.ReuseCoverageData debugger.Option.SupplyPower debugger.Option.SupplyPowerVoltage debugger.Option.Timer debugger.Option.Trace debugger.Option.TraceBranchPC debugger.Option.TraceDataAccess debugger.Option.TracePriority debugger.Option.TraceTarget debugger.Option.UseTraceData	This property sets or refers to the options of the debug tool.
debugger.ProcessorElement	This property sets or refers to the PE of the multi-core.
debugger.ProcessorElementName	This property sets or refers to the PE of multiple cores with the name.
debugger.SoftwareTraceLPD.PEList	This property refers to a list of PE numbers for which software tracing (LPD output) is available.
debugger.SoftwareTraceLPD.Priority	This property sets or refers to the priority for the acquisition of software trace (LPD output) data.
debugger.SoftwareTraceLPD.RecordingMode	This property sets or refers to the operation of the debug tool when the memory for recording software trace information (LPD output) becomes full.
debugger.XTrace.Addup debugger.XTrace.Complement debugger.XTrace.Mode	This property sets or refers to the tracing options of the debug tool.

debugger.ActionEvent.GetLine

This property sets or refers to the number of action event results.

[Specification format]

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

[Setting(s)]

Setting	Description
number	Set the number of action event results that can be held in the Python console (default: 10000).

[Reference]

Current set value

[Detailed description]

- This property sets or refers to the number of action event results that can be held in the Python console.
- If the number that was set is exceeded, action event results cannot be held. Deletion is performed from old action events. The valid range is from 5000 to 100000.

[Example of use]

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

debugger.ADConvertDataInExecution

This property sets or refers to data collected in debugging. [Smart Analog]

[Specification format]

```
debugger.ADConvertDataInExecution = adConvertDataInExecution
```

[Setting(s)]

Setting	Description
adConvertDataInExecution	Set whether to collect data during debugging. True: Collect data during debugging. False: Do not collect data during debugging.

[Reference]

Setting for data collection during execution

[Detailed description]

- This property sets or refers to data collected in debugging.

[Example of use]

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

debugger.DebugTool.SerialNumber

This property sets or refers to the serial number of the emulator.

[Specification format]

```
debugger.DebugTool.SerialNumber = serialNumber
```

[Setting(s)]

Setting	Description
serialNumber	Set the serial number of the emulator as a string.

[Reference]

Serial number of the emulator (string)

[Detailed description]

- This property sets or refers to the serial number of the emulator.

[Example of use]

```
>>>print debugger.DebugTool.SerialNumber
E1:_00000100
>>>debugger.DebugTool.SerialNumber = "E1:_00200100"
>>>print debugger.DebugTool.SerialNumber
E1:_00200100
>>>
```

debugger.DebugTool.SerialNumberList

This property refers to the list of serial numbers of emulators.

[Specification format]

```
debugger.DebugTool.SerialNumberList
```

[Setting(s)]

None

[Reference]

List of serial numbers of emulators (string)

[Detailed description]

- This property refers to the list of serial numbers of emulators.

[Example of use]

```
>>>dl = debugger.DebugTool.SerialNumberList
>>>print dl
['E1:_00200100', 'E1:_00321221','E1:_00200423']
>>>
```

debugger.Download.Property

This property sets or refers to conditions of the download file of the debug tool.

[Specification format]

```
debugger.Download.Property = downloadlist
```

[Setting(s)]

Setting	Description
downloadlist	Set conditions of the download file of the debug tool as a list. See the DownloadCondition class for details.

[Reference]

List of conditions of the download file

[Detailed description]

- This property sets or refers to conditions of the download file of the debug tool.

Caution The list specified with *downloadlist* should be a C# list and not an IronPython list.
Therefore, first refer to the list of conditions and manipulate that list.

Example

```
di = debugger.Download.Property
```

You can manipulate conditions in the list set for di.
For the usage, refer to [Example of use].

[Example of use]

```
>>>di = debugger.Download.Property
>>>print di[0].FileName
C:\project\test.abs
>>>print di[0].DownloadFileType
LoadModule
>>>dc = DownloadCondition()
>>>dc.FileName = "C:/project/test2.abs"
>>>dc.DownloadFileType = DownloadFileType.LoadModule
>>>di.Add(dc)
>>>debugger.Download.Property = di
>>>
```

debugger.Interrupt.ExceptionCause

This property refers to the exception cause code.

[Specification format]

```
debugger.Interrupt.ExceptionCause
```

[Setting(s)]

None

[Reference]

Exception cause code

[Detailed description]

- This property refers to the exception cause code.
- The exception cause code can be referenced only when the hook function is AfterInterrupt or while the parameter of the callback function (pythonConsoleCallback) is 50.
  See the Hook function for the hook function and callback function.

[Example of use]

(1) Create the script file (C:\test\sample.py).

```
def AfterInterrupt():
    if debugger.Interrupt.ExceptionCause == 0x30:
        print "OK"
    else:
        print "NG"

def pythonConsoleCallback(Id):
    if Id == 50:
        if debugger.Interrupt.ExceptionCause == 0x30:
            print "OK"
        else:
            print "NG"
```

(2) Use a Hook function to register the created script file from the Python console.

```
>>> Hook("C:/test/test.py")
>>>
```

debugger.IsMulticore

This property checks whether or not the microcontroller of the active project is multi-core.

[Specification format]

```
debugger.IsMulticore
```

[Setting(s)]

None

[Reference]

When the microcontroller is multi-core: True
When the microcontroller is not multi-core: False

[Detailed description]

- This property checks whether or not the microcontroller of the active project is multi-core.

Caution This property is used to confirm whether or not multiple CPU cores are present. A core other than a CPU, such as a DSP, is not included as one among multiple cores.

[Example of use]

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

debugger.Memory.NoVerify

This property switches the write-time verification setting. [Except simulator]

[Specification format]

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

[Setting(s)]

Setting	Description
noverify	Set whether to verify during writes. True: Verify during writes. False: Do not verify during writes.

[Reference]

Set value

Caution If a PM+ workspace is converted to a CS+ project, then there will be no debugging tool in the main project. For this reason, "None" will be returned if the main project is the active project.

[Detailed description]

- This property switches the write-time verification setting.

[Example of use]

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

debugger.Option.AccessDuringExecution
debugger.Option.AccessStopExecution
debugger.Option.AccumulateTraceTime
debugger.Option.AfterTraceMemoryFull
debugger.Option.Coverage
debugger.Option.CpuEndian
debugger.Option.MainClockFrequency
debugger.Option.OpenBreak
debugger.Option.ResetMask
debugger.Option.ReuseCoverageData
debugger.Option.SupplyPower
debugger.Option.SupplyPowerVoltage
debugger.Option.Timer
debugger.Option.Trace
debugger.Option.TraceBranchPC
debugger.Option.TraceDataAccess
debugger.Option.TracePriority
debugger.Option.TraceTarget
debugger.Option.UseTraceData

This property sets or refers to the options of the debug tool.

[Specification format]

```
debugger.Option.AccessDuringExecution = accessDuringExecution
debugger.Option.AccessStopExecution = afterTrace
debugger.Option.AccumulateTraceTime = accumulateTraceTime
debugger.Option.AfterTraceMemoryFull = accessStopExecution
debugger.Option.Coverage = coverage
debugger.Option.CpuEndian = endianType
debugger.Option.MainClockFrequency = mainClockFrequency
debugger.Option.OpenBreak = openBreak
debugger.Option.ResetMask = [targetReset, internalReset]
debugger.Option.ReuseCoverageData = reuseCoverageData
debugger.Option.SupplyPower = supplyPower
debugger.Option.SupplyPowerVoltage = voltage
debugger.Option.Timer = timer
debugger.Option.Trace = trace
debugger.Option.TraceBranchPC = traceBranchPC
debugger.Option.TraceDataAccess = traceDataAccess
debugger.Option.TracePriority = tracePriority
debugger.Option.TraceTarget = traceTarget
debugger.Option.UseTraceData = useTraceDataType
```

[Setting(s)]

<table>
<tr><th>Setting</th><th>Description</th></tr>
<tr><td><code>accessDuringExecution</code></td><td>Set whether to allow access to memory during execution. [RH850][E1/E20/Full-spec emulator/IE850A]
True: Allow access to memory during execution.
False: Do not allow access to memory during execution.</td></tr>
<tr><td><code>afterTrace</code></td><td>Set the operation to be taken after using up trace memory.
The values that can be specified are shown below.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>AfterTraceMemoryFull.NoneStop</td><td>Overwrite trace memory and continue execution.</td></tr>
<tr><td>AfterTraceMemoryFull.StopTrace</td><td>Stop tracing.</td></tr>
<tr><td>AfterTraceMemoryFull.Stop</td><td>Stop execution (stop the program).</td></tr>
</table></td></tr>
<tr><td><code>accumulateTraceTime</code></td><td>Set whether to display the trace time with accumulated time. [Simulator]
True: Display the trace time with accumulated time.
False: Display the trace time with differential value.</td></tr>
<tr><td><code>accessStopExecution</code></td><td>Set whether to instantaneously stop execution and make an access.
True: Stop execution for a moment and make an access.
False: Stop execution for a moment but do not make an access.</td></tr>
<tr><td><code>coverage</code></td><td>Set whether to use the coverage function. [IECUBE][IECUBE2][Simulator]
True: Use the coverage function.
False: Do not use the coverage function.</td></tr>
<tr><td><code>endianType</code></td><td>Sets the endianness of the microcontroller. [RX]
The values that can be specified are shown below.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>EndianType.Big</td><td>The byte order of data is big endian.</td></tr>
<tr><td>EndianType.Little</td><td>The byte order of data is little endian.</td></tr>
</table></td></tr>
<tr><td><code>mainClockFrequency</code></td><td>Set the main clock frequency (numerical value) in units of kHz. [Except RX simulator]</td></tr>
<tr><td><code>openBreak</code></td><td>Set whether to use the open break function.
True: Use the open break function.
False: Do not use the open break function.</td></tr>
<tr><td><code>targetReset</code></td><td>Set whether to mask the TARGET RESET signal. [RL78 [E1/E2/E20/E2 Lite/EZ Emulator/IECUBE]] [RH850 [E1/E2/E20/Full-spec emulator/IE850A]]
True: Mask the TARGET RESET signal.
False: Do not mask the TARGET RESET signal.
Caution Whether "True" or "False" is specifiable differs with the combination of the device and the emulator. See "CS+ Integrated Development Environment User’s Manual: Debug Tool" for details.</td></tr>
<tr><td><code>internalReset</code></td><td>Set whether to mask the INTERNAL RESET signal. [RL78 [E1/E2/E20/E2 Lite/EZ Emulator/IECUBE]] [RH850 [E1/E2/E20/Full-spec emulator/IE850A]]
True: Mask the INTERNAL RESET signal.
False: Do not mask the INTERNAL RESET signal.
Caution Whether "True" or "False" is specifiable differs with the combination of the device and the emulator. See "CS+ Integrated Development Environment User’s Manual: Debug Tool" for details.</td></tr>
<tr><td><code>reuseCoverageData</code></td><td>Set whether to reuse the coverage result.
True: Reuse the coverage result.
False: Do not reuse the coverage result.</td></tr>
</table>

Setting	Description
supplyPower	Select whether power should be supplied from the emulator. [E1/E2/E2 Lite] True: Power should be supplied from the emulator. False: Power should not be supplied from the emulator.
voltage	Specify the voltage in volts to be supplied as power from the emulator. [E1/E2] For example, enter 3.3 to specify 3.3 V.
timer	Set whether to use the timer function. True: Use the timer function. False: Do not use the timer function.
trace	Set whether to use the trace function. [IECUBE][IECUBE2][Simulator] True: Use the trace function. False: Do not use the trace function.
traceBranchPC	Set whether to collect PC values for source/destination instructions of branching during program execution as trace data. True: Collect PC values as trace data. False: Do not collect PC values as trace data. **Caution** When you are using a simulator as the debugging tool, debugger.Option.TraceDataAccess is also set to "True" if the setting of debugger.Option.TraceBranchPC is "True". Conversely, debugger.Option.TraceDataAccess is also set to "False" if the setting of debugger.Option.TraceBranchPC is "False".
traceDataAccess	Set whether to collect data information on access-related events that occurred during program execution as trace data. True: Collect data information as trace data. False: Do not collect data information as trace data. **Caution** When you are using a simulator as the debugging tool, debugger.Option.TraceBranchPC is also set to "True" if the setting of debugger.Option.TraceDataAccess is "True". Conversely, debugger.Option.TraceBranchPC is also set to "False" if the setting of debugger.Option.TraceDataAccess is "False".
tracePriority	Set whether the acquisition of all data or real-time operation should have priority in the acquisition of trace data. [RH850][E1/E2/E20/Full-spec emulator/IE850A] The values that can be specified are shown below.
	Value: TracePriority.SpeedPriority — Description: Real-time operation is given priority in tracing.
	Value: TracePriority.DataPriority — Description: The CPU pipeline is temporarily stopped so that no trace data is lost.
traceTarget	Set the target of tracing. [RH850] The values that can be specified are shown below.
	Value: TraceTarget.DebugOnly — Description: Trace data will only be acquired on the CPU core being debugged.
	Value: TraceTarget.AllCore — Description: Trace data will be acquired on all CPU cores.

Setting	Description	
useTraceDataType	Set which function to use the trace data in. [IECUBE [V850]][IECUBE2] The functions that can be specified are shown below.	
	Type	Description
	UseTraceDataType.RRM	RRM function
	UseTraceDataType.Trace	Trace function
	UseTraceDataType.Coverage	Coverage function

[Reference]

Set value

Caution If a PM+ workspace is converted to a CS+ project, then there will be no debugging tool in the main project. For this reason, "None" will be returned if the main project is the active project.

[Detailed description]

- This property sets or refers to the options of the debug tool.

[Example of use]

```
>>>print debugger.Option.AccessDuringExecution
True
>>>debugger.Option.AccessDuringExecution = False
>>>print debugger.Option.AccessDuringExecution
False
>>>
```

```
>>>print debugger.Option.AccumulateTraceTime
True
>>>debugger.Option.AccumulateTraceTime = False
>>>print debugger.Option.AccumulateTraceTime
False
>>>
```

```
>>>print debugger.Option.MainClockFrequency
10000
>>>debugger.Option.MainClockFrequency = 12000
>>>print debugger.Option.MainClockFrequency
12000
>>>
```

```
>>>print debugger.Option.ResetMask
[False, False]
>>>debugger.Option.ResetMask = [True, False]
>>>print debugger.Option.ResetMask
[True, False]
>>>
```

```
>>>print debugger.Option.SupplyPower
False
>>>debugger.Option.SupplyPower = True
>>>print debugger.Option.SupplyPower
True
>>>
```

```
>>>print debugger.Option.SupplyPowerVoltage
3.3
>>>debugger.Option.SupplyPowerVoltage = 1.8
>>>print debugger.Option.SupplyPowerVoltage
1.8
>>>
```

```
>>>print debugger.Option.TraceBranchPC
True
>>>debugger.Option.TraceBranchPC = False
>>>print debugger.Option.TraceBranchPC
False
>>>
```

```
>>>print debugger.Option.TraceDataAccess
True
>>>debugger.Option.TraceDataAccess = False
>>>print debugger.Option.TraceDataAccess
False
>>>
```

```
>>>print debugger.Option.TracePriority
SpeedPriority
>>>debugger.Option.TracePriority = TracePriority.DataPriority
>>>print debugger.Option.TracePriority
DataPriority
>>>
```

```
>>>print debugger.Option.TraceTarget
AllCore
>>>debugger.Option.TraceTarget = TraceTarget.DebugOnly
>>>print debugger.Option.TraceTarget
DebugOnly
>>>
```

```
>>>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

This property sets or refers to the PE of the multi-core.

[Specification format]

```
debugger.ProcessorElement = number
```

[Setting(s)]

Setting	Description
number	Set the PE number with the number.

[Reference]

Current set value

[Detailed description]

- This property sets or refers to the PE of the multi-core.

Caution When the PE is set, it must be connected to the debugging tool.

[Example of use]

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

debugger.ProcessorElementName

This property sets or refers to the PE of multiple cores with the name.

[Specification format]

```
debugger.ProcessorElementName = name
```

[Setting(s)]

Setting	Description
name	Set the PE name as a string.

[Reference]

Current set value

[Detailed description]

- This property sets or refers to the PE of the multi-core.
- The specifiable strings can be obtained by calling debugger.GetProcessorElementNames.

Caution When the PE is set, it must be connected to the debugging tool.

[Example of use]

```
>>>print debugger.ProcessorElementName
CPU1
>>>debugger.ProcessorElementName = 'CPU2'
>>>print debugger.ProcessorElementName
CPU2
>>>
```

debugger.SoftwareTraceLPD.PEList

This property refers to a list of PE numbers for which software tracing (LPD output) is available. [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.PEList
```

[Setting(s)]

None

[Reference]

A list of PE numbers for which software tracing (LPD output) is available

[Detailed description]

- This property refers to a list of PE numbers for which software tracing (LPD output) is available.

[Example of use]

```
>>>print debugger.SoftwareTraceLPD.PEList
[0, 1, 2]
>>>
```

debugger.SoftwareTraceLPD.Priority

This property sets or refers to the priority for the acquisition of software trace (LPD output) data. [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.Priority = tracePriority
```

[Setting(s)]

Setting	Description	
tracePriority	Set the priority for the acquisition of software trace (LPD output) data. The values that can be specified are shown below.	
	Type	Description
	TracePriority.SpeedPriority	Real-time operation is given priority in tracing.
	TracePriority.DataPriority	The CPU pipeline is temporarily stopped so that no trace data is lost.

[Reference]

The current priority setting for the acquisition of software trace (LPD output) data

[Detailed description]

- This property sets or refers to the current priority setting for the acquisition of software trace (LPD output) data.

[Example of use]

```
>>>print debugger.SoftwareTraceLPD.Priority
SpeedPriority
>>>debugger.SoftwareTraceLPD.Priority = TracePriority.DataPriority
>>>print debugger.SoftwareTraceLPD.Priority
DataPriority
>>>
```

debugger.SoftwareTraceLPD.RecordingMode

This property sets or refers to the operation of the debug tool when the memory for recording software trace information (LPD output) becomes full. [RH850][E2]

[Specification format]

```
debugger.SoftwareTraceLPD.RecordingMode = recordingMode
```

[Setting(s)]

Setting	Description	
recordingMode	Set the operation of the debug tool when the memory for recording software trace information (LPD output) becomes full. The values that can be specified are shown below.	
	Type	Description
	TraceMode.FullBreak	Execution of the program and the writing of trace data stop when the recording memory is full.
	TraceMode.FullStop	The writing of trace data stop when the recording memory is full.
	TraceMode.NonStop	Trace data continue to be overwritten even after the recording memory is full.

[Reference]

The current setting for the operation of the debug tool when the memory for recording software trace information (LPD output) becomes full

[Detailed description]

- This property sets or refers to the operation of the debug tool when the memory for recording software trace information (LPD output) becomes full.

[Example of use]

```
>>>print debugger.SoftwareTraceLPD.RecordingMode
NonStop
>>>debugger.SoftwareTraceLPD.RecordingMode = TraceMode.FullStop
>>>print debugger.SoftwareTraceLPD.RecordingMode
FullStop
>>>
```

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

This property sets or refers to the tracing options of the debug tool. [IECUBE][IECUBE2][Simulator]

[Specification format]

```
debugger.XTrace.Addup = addup [Simulator]
debugger.XTrace.Complement = complement [IECUBE[V850]][IECUBE2[V850]]
debugger.XTrace.Mode = traceMode [Simulator][IECUBE][IECUBE2]
```

[Setting(s)]

Setting	Description	
addup	Set whether to add up times/tags. True: Add up times/tags. False: Do not add up times/tags.	
complement	Set whether to supplement the trace. True: Supplement the trace. False: Do not supplement the trace.	
traceMode	Set the trace control mode. The trace control modes that can be specified are shown below.	
	Type	Description
	TraceMode.FullBreak	Stop program execution and writing of trace data after all trace data has been used up.
	TraceMode.FullStop	Stop writing trace data after all trace data has been used up.
	TraceMode.NonStop	Continue writing trace data even if all trace data has been used up.

[Reference]

Set value

Caution If a PM+ workspace is converted to a CS+ project, then there will be no debugging tool in the main project. For this reason, "None" will be returned if the main project is the active project.

[Detailed description]

- This property sets or refers to the tracing options of the debug tool.

[Example of use]

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

B.3.11 CS+ Python event

Below is a list of CS+ Python events.

Table B.11 CS+ Python Event

Event Name	Function Description
build.BuildCompleted	This event informs that a build has been completed.

build.BuildCompleted

This event informs that a build has been completed.

[Handler format]

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

[Handler argument(s)]

Argument	Description
sender	The sender of the build event are passed.
e	The parameters at the end of build execution are passed.

[Return value]

None

[Detailed description]

- This event informs that a build has been completed.

[Example of use]

```
>>>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       ... Event connection
>>>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 Cautions for Python Console

(1) Caution for Japanese input
The Japanese input feature cannot be activated from the Python Console. To enter Japanese text, write it in an external text editor or the like, and copy and paste it into the console.

(2) Caution for prompt displays
The Python Console prompt of ">>>" may be displayed multiply, as ">>>>>>", or results may be displayed after the ">>>", and there may be no ">>>" prompt before the caret. If this happens, it is still possible to continue to enter functions.

(3) Caution for executing scripts for projects without load modules
If a script is specified in the startup options that uses a project without a load module file, or if *project_filename*.py is placed in the same folder as the project file, then although the script will be executed automatically after normal project loading, it will not be executed if there is no load module file.

(4) Cautions for forced termination
If the following operations are performed while a script like an infinite loop is running, then the results of function execution may be an error, because the function execution will be terminated forcibly.

- Forcible termination by selecting "Forcibly terminate" from the context menu or pressing Ctrl+D in the Python Console
- Changing the active project in a project with multiple projects

C. External Communications with the Python 3 Execution Environment/csplus Module Functions

This section describes the csplus module functions which are used for external communications with the Python 3 execution environment.

Table C.12 csplus Module Functions

Function Name	Function Description
csplus.add_address_breakpoint	This command adds an address breakpoint with additional type setting.
csplus.add_event_listener	This command adds the specified listener to the list of functions to be called when an event happen in CS+.
csplus.connect	Establish socket communication between Python 3 execution environment and CS+.
csplus.download_loadmodule	This function downloads a file.
csplus.get_all_breakpoints	This function gets information of all breakpoints.
csplus.get_expression_info	This function gets size, type and value of an expression.
csplus.get_float_expression	This functions gets value of a float expression.
csplus.get_integer_expression	This function gets value of a integer expression.
csplus.get_register	This function gets value of a register.
csplus.get_source_line_address	This function gets address of a source line.
csplus.get_symbol_address	This function gets address of a symbol.
csplus.get_variable_size	This function gets the size of a variable.
csplus.get_variable_type	This function gets the type of a variable.
csplus.is_debug_session_running	This function checkes the debug sesion is running or not.
csplus.launch_debug_session	This function launches a debug session.
csplus.read_memory	This function refers to the memory.
csplus.remove_all_breakpoints	This function removes an address breakpoint.
csplus.remove_event_listener	This function removes the specified listener from the list of functions to be called when an event happen in CS+.
csplus.reset_debug_session	This function resets a debug session.
csplus.resume_debug_session	This function resumes a debug session.
csplus.set_float_expression	This function sets value of a float expression.
csplus.set_integer_expression	This function sets value of a integer expression.
csplus.skip_all_breakpoints	This function skips all breakpoints.
csplus.step_in	This function steps through source codes instructions.
csplus.suspend_debug_session	This function suspends a debug session.
csplus.terminate	This function terminates the connected CS+, CS+ will be closed.
csplus.terminate_debug_session	This function terminates debug session.
csplus.write_memory	This function writes memory value.

csplus.add_address_breakpoint

This command adds an address breakpoint with additional type setting.

[Specification format]

```
csplus.add_address_breakpoint(address, breakpoint_type="", temporary=False)
```

[Argument(s)]

Argument	Description
address	The address to set breakpoint at
breakpoint_type	The type of breakpoint to be set, either "Hardware" or "Software" (case-insensitive). If not inputted or inputted with other value, the type will be decided by CS+.
temporary	A flag indicating whether the breakpoint should be hit only once or not

[Return value]

The index of the added breakpoint in the list of all breakpoints

[Detailed description]

- The returned value is just the index at the time the breakpoint is added. It may be changed when other breakpoints are added/removed.

[Example of use]

```
>>> csplus.add_address_breakpoint(0x12, "hardware", True)
2
```

csplus.add_event_listener

This command adds the specified listener to the list of functions to be called when an event happen in CS+.

[Specification format]

```
csplus.add_event_listener(listener)
```

[Argument(s)]

Argument	Description
listener	A function which accept a String parameter.

[Return value]

None

[Detailed description]

- The function has format function_name(string_parameter)
- The value which will be passed to the String parameter of listener has the format debugger_state;0;event.

Value	Description
debugger_state	A fixed string indicating that the event is related to debug activity.
event	The name of the event. The information has one of following value: Program is downloaded: downloaded Debug session is reset: reset Debug session is about to resume: resuming Debug session is suspended: suspended

[Example of use]

```
>>> def listener1(event): print("Listener 1: " + event)
...
>>> csplus.add_event_listener(listener1)
>>> csplus.download_loadmodule(session_id)
>>> Listener 1: debugger state;0;reset
Listener 1: debugger state;0;resuming
```

csplus.connect

Establish socket communication between Python 3 execution environment and CS+.

[Specification format]

```
csplus.connect()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- If the connection cannot be established, an exception will be thrown.
- This function must be called before calling other functions.

[Example of use]

```
>>> import sys
>>> sys.path.append("C:\Program Files (x86)\Renesas
Electronics\CS+\CC\Plugins\PythonConsole\integration_service")
>>> import csplus
>>> csplus.connect()
>>>
```

csplus.download_loadmodule

This function downloads a file.

[Specification format]

```
csplus.download_loadmodule(session_id, file_path="", offset=0, load_image=True,
load_symbols=True, clear_old_symbols=False, core_name="")
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
file_path	Full path of the file to download.
offset	The offset to download the file at.
load_image	Specify whether to download image from the specified file.
load_symbols	Specify whether to download symbols from the specified file.
clear_old_symbols	Specify whether to add symbols from the specified file to current symbol table or overwrite symbol table in debugger.
core_name	The name of the core to download to.

[Return value]

None

[Detailed description]

- If file_path is not specified or empty, download all loadmodules defined in launch configuration of the specified debug session, ignoring all other parameter's value.
- No reset happens after downloading, "csplus.reset_debug_session()" command need to be called manually.

[Example of use]

```
>>> import sys
>>> sys.path.append("C:\Program Files (x86)\Renesas
Electronics\CS+\CC\Plugins\PythonConsole\integration_service")
>>> import csplus
>>> csplus.connect()
>>> session_id = csplus.launch_debug_session("", True)
>>> csplus.download_loadmodule(session_id)
>>>
```

csplus.get_all_breakpoints

This function gets information of all breakpoints.

[Specification format]

```
csplus.get_all_breakpoints()
```

[Argument(s)]

None

[Return value]

List: The returned list contains information of a breakpoint in the format.
'index, enabled, type, temporary, location'

Value	Description
index	The index of this breakpoint in the list of all breakpoints.
enabled	Whether this breakpoint is enabled or not
type	The type of this breakpoint. Can be "hardware", "software" or "regular"
temporary	Whether this breakpoint is temporary breakpoint or not
location	Information about the location the breakpoint is set. Can be a number (e.g. "0xffc00000"), a function (e.g. "main") or a source line (e.g. "ccrx.c:30")

[Detailed description]

- This function gets information of all breakpoints. Only address breakpoint, function breakpoint and line breakpoint will be included in the returned list.

[Example of use]

```
>>> csplus.get_all_breakpoints()
['0,true,hardware,false,RL78_F13.c:26', '1,true,hardware,false,RL78_F13.c:30']
```

csplus.get_expression_info

This function gets size, type and value of an expression.

[Specification format]

```
csplus.get_expression_info(session_id, expression)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
expression	The expression to be evaluated

[Return value]

Size, type, value of the expression with format: size;type;value
Each information in the return string is described in below.

Value	Description
size	The size in byte of the target expression
type	The type of the target expression
value	The value of the specified expression

[Detailed description]

- This function gets size, type and value of an expression.

[Example of use]

```
>>> csplus.get_expression_info(0, "floatVar")
'4;float;10.1234'
```

csplus.get_float_expression

This functions gets value of a float expression.

[Specification format]

```
csplus.get_float_expression(session_id, expression)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
expression	The expression to be evaluated

[Return value]

The value of the specified expression.

[Detailed description]

- This functions gets value of a float expression.

[Example of use]

```
>>> csplus.get_float_expression(0, "floatVar")
10.1234
```

csplus.get_integer_expression

This function gets value of a integer expression.

[Specification format]

```
csplus.get_integer_expression(session_id, expression)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
expression	The expression to be evaluated

[Return value]

The value of the specified expression.

[Detailed description]

- This function gets value of a integer expression.

[Example of use]

```
>>> csplus.get_integer_expression(0, "intVar")
10
```

csplus.get_register

This function gets value of a register.

[Specification format]

```
csplus.get_register(session_id, register_name)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
register_name	The register name value.

[Return value]

The register value.

[Example of use]

```
>>> csplus.get_register(session_id, "pc")
1234
>>>
```

csplus.get_source_line_address

This function gets address of a source line.

[Specification format]

```
csplus.get_source_line_address(session_id, source_file, line_number)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
source_file	The source file to get address from. Can be either file name or file path If multiple load modules have been downloaded, set the parameter as “<load module name>$<file name>” to specify an additional load module. Example : csplus.get_source_line_address(0, “loadmodule2.abs$main2.c”, 40)
line_number	The line number in the specified source file to get address at

[Return value]

The address of the source line at the specified line number in the specified source file

[Detailed description]

- This function gets address of a source line.
- If the specified source line does not have address, None will be returned.

[Example of use]

```
>>> csplus.get_source_line_address(0, "ccrx.c", 30)
1234
```

csplus.get_symbol_address

This function gets address of a symbol.

[Specification format]

```
csplus.get_symbol_address(session_id, symbol_name)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
symbol_name	The symbol to get address.

[Return value]

The address of the specified symbol

[Example of use]

```
>>> csplus.get_symbol_address(session_id, "main")
00001234
>>>
```

csplus.get_variable_size

This function gets the size of a variable.

[Specification format]

```
csplus.get_variable_size(session_id, variable)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
variable	The name of the target variable

[Return value]

The size of the target variable

[Detailed description]

- This function gets the size of a variable.

[Example of use]

```
>>> csplus.get_variable_size(0, "x")
2
```

csplus.get_variable_type

This function gets the type of a variable.

[Specification format]

```
csplus.variable_size(session_id, variable)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
variable	The name of the target variable

[Return value]

The type of the target variable

[Detailed description]

- This function gets the type of a variable.

[Example of use]

```
>>> csplus.get_variable_type(0, "x")
int
```

csplus.is_debug_session_running

This function checkes the debug sesion is running or not.

[Specification format]

```
csplus.is_debug_session_running(session_id)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.

[Return value]

True if debug session is running, False if debug session is suspending.

[Detailed description]

- If is_debug_session_running is called unsuccessfully, an exception will be thrown. In case of multicore device, only the first core is checked.

[Example of use]

```
>>> csplus.is_debug_session_running(session_id)
True
>>>
```

csplus.launch_debug_session

This function launches a debug session.

[Specification format]

```
csplus.launch_debug_session(debug_config_name, block_download=False)
```

[Argument(s)]

Argument	Description
debug_config_name	This argument is ignored in this version.
block_download	Whether to block the download of load module files

[Return value]

This version always returns 0.

[Detailed description]

- If block_download is False, a new debug session will be launched using the specified launch config.
- If block_download is True, do not download load modules.

[Example of use]

```
>>> import sys
>>> sys.path.append("C:\Program Files (x86)\Renesas
Electronics\CS+\CC\Plugins\PythonConsole\integration_service")
>>> import csplus
>>> csplus.connect()
>>> session_id = csplus.launch_debug_session("")
>>>
```

csplus.read_memory

This function refers to the memory.

[Specification format]

```
csplus.read_memory(session_id, address, length)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
address	The address to read memory at
length	The number of byte to read

[Return value]

The memory value in hexadecimal format at the specified address.

[Detailed description]

- The byte order in the returned string is same as the byte order in memory regardless endian setting (depend on the endian of IO registers area, the byte order can be different).

[Example of use]

```
>>> csplus.read_memory(session_id, 0x0, 1)
'10'
>>>
```

csplus.remove_all_breakpoints

This function removes all breakpoints.

[Specification format]

```
csplus.remove_all_breakpoints()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- This function removes all breakpoints.

[Example of use]

```
>>> csplus.remove_all_breakpoints()
>>>
```

csplus.remove_event_listener

This function removes the specified listener from the list of functions to be called when an event happen in CS+.

[Specification format]

```
csplus.remove_event_listener(listener)
```

[Argument(s)]

Parameter	Description
listener	Specifies the listener function to remove.

[Return value]

None

[Example of use]

```
>>> def listener1(event): print("Listener 1: " + event)
...
>>> csplus.add_event_listener(listener1)
>>> csplus.remove_event_listener(listener1)
>>>
```

csplus.reset_debug_session

This function resets a debug session.

[Specification format]

```
csplus.reset_debug_session(session_id)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.

[Return value]

None

[Detailed description]

- If reset_debug_session is called unsuccessfully, an exception will be thrown.

[Example of use]

```
>>> csplus.reset_debug_session(session_id)
>>>
```

csplus.resume_debug_session

This function resumes a debug session.

[Specification format]

```
csplus.resume_debug_session(session_id, wait_break=False)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
wait_break	Specify whether to wait for the debug session to suspend or not.

[Return value]

None

[Detailed description]

- If the debug session is running, calling this command does nothing.
- If wait_break is true, this command will return after the suspending of the debug session. The program must be suspended to use the next command.

[Example of use]

```
>>> csplus.resume_debug_session(session_id)
>>>
```

csplus.set_float_expression

This function sets value of a float expression.

[Specification format]

```
csplus.set_float_expression(session_id, expression, value)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
expression	The expression to be evaluated
value	The float value to set to the specified expression

[Return value]

None

[Detailed description]

- This function sets value of a float expression.

[Example of use]

```
>>> csplus.set_float_expression(0, "floatVar", 10.1234)
>>> csplus.get_float_expression(0, "floatVar")
10.1234
```

csplus.set_integer_expression

This function sets value of a integer expression.

[Specification format]

```
csplus.set_integer_expression(session_id, expression, value)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.
expression	The expression to be evaluated
value	The integer value to set to the specified expression

[Return value]

None

[Detailed description]

- This function sets value of a integer expression.

[Example of use]

```
>>> csplus.set_integer_expression(0, "intVar", 10)
>>> csplus.get_integer_expression(0, "intVar")
10
```

csplus.skip_all_breakpoints

This function skips all breakpoints.

[Specification format]

```
csplus.skip_all_breakpoints(skip=True)
```

[Argument(s)]

Argument	Description
skip	Specify whether all breakpoints in the current workspace should be skipped

[Return value]

None

[Detailed description]

- This command just ignore all breakpoints when running without changing the enabled state of breakpoints.

[Example of use]

```
>>> csplus.skip_all_breakpoints(True)
>>> csplus.resume_debug_session(0)
```

csplus.step_in

This function steps through source codes instructions.

[Specification format]

```
csplus.step_in(session_id, instruction_step=False)
```

[Argument(s)]

引数	Description
session_id	This argument is ignored in this version.
instruction_step	A flag to use instruction step or not

[Return value]

None

[Example of use]

```
>>> csplus.rstep_in(session_id)
>>>
```

csplus.suspend_debug_session

This function suspends a debug session.

[Specification format]

```
csplus.suspend_debug_session(session_id)
```

[Argument(s)]

Argument	Description
session_id	This argument is ignored in this version.

[Return value]

None

[Detailed description]

- If suspend_debug_session is called unsuccessfully, an exception will be thrown.

[Example of use]

```
>>> csplus.suspend_debug_session(session_id)
>>>
```

csplus.terminate

This function terminates the connected CS+, CS+ will be closed.

[Specification format]

```
csplus.terminate()
```

[Argument(s)]

None

[Return value]

None

[Detailed description]

- If no CS+ instance has been connected before via connect() command, an exception will be thrown.

[Example of use]

```
>>> import sys
>>> sys.path.append("C:\Program Files (x86)\Renesas
Electronics\CS+\CC\Plugins\PythonConsole\integration_service")
>>> import csplus
>>> csplus.connect()
>>> csplus.terminate()
>>>
```

csplus.terminate_debug_session

This function terminates debug session.

[Specification format]

```
csplus.terminate_debug_session(session_id)
```

[Argument(s)]

引数	説明
session_id	This argument is ignored in this version.

[Return value]

None

[Detailed description]

- If terminate_debug_session is called unsuccessfully, an exception will be thrown.

[Example of use]

```
>>> csplus.terminate_debug_session(session_id)
>>>
```

csplus.write_memory

This function writes memory value.

[Specification format]

```
csplus.write_memory(session_id, address, length, data)
```

[Argument(s)]

引数	説明
session_id	This argument is ignored in this version.
address	The address to write memory to
length	The number of byte to write
data	The memory value to be written in hexadecimal format

[Return value]

None

[Detailed description]

- The byte order of data written in memory is same as the byte order of the input data regardless endian setting (depend on the endian of IO registers area, the byte order can be different).

[Example of use]

```
>>> csplus.write_memory(session_id, 0x2, 2, "1234")
>>> csplus.read_memory(session_id, 0x2, 2)
'1234'
>>>
```

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jun 01, 2024	-	First Edition issued

CS+ V8.12.00 User's Manual:
Python Console

Publication Date: Rev.1.00 Jun 01, 2024
Published by: Renesas Electronics Corporation

CS+ V8.12.00

Renesas Electronics Corporation

R20UT5497EJ0100