RENESAS

CS+ V3.02.00

Integrated Development Environment

User's Manual: Python Console

Target Device

78K0 Microcontroller

RL78 Family

78K0R Microcontroller

V850 Family

RX Family

RH850 Family

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, V850 family, RL78 family, 78K0R microcontrollers, and 78K0 microcontrollers, and provides an outline of its features.

CS+ is an integrated development environment (IDE) for RH850 family, RX family, V850 family, RL78 family, 78K0R microcontrollers, and 78K0 microcontrollers, 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.

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

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.

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

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.

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. The Python Console panel opens.
You can control CS+ and the debugging tool by executing Python functions and control statements in the panel.

Figure 2.1   Python Console Panel
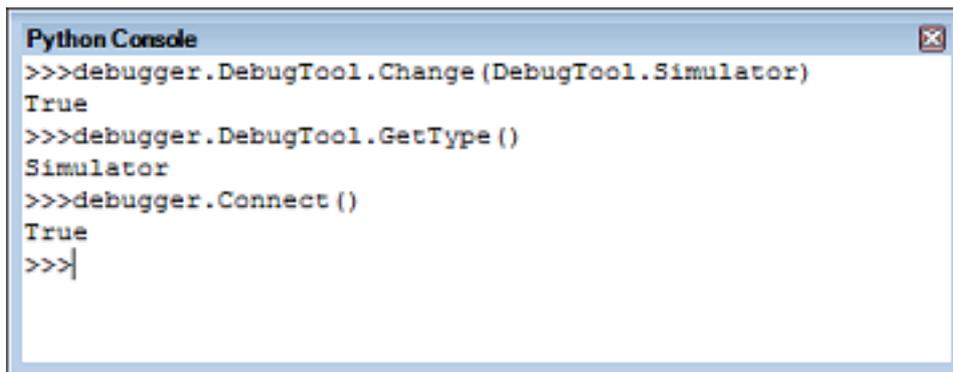
```
Python Console                                              ⊠
>>>debugger.DebugTool.Change(DebugTool.Simulator)
True
>>>debugger.DebugTool.GetType()
Simulator
>>>debugger.Connect()
True
>>>|
```

Remark   See "B. Python CONSOLE/Python FUNCTIONS" for details about the Python console and Python func-
             tions.

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	This panel is used to use IronPython to control CS+ and the debug tool via the command input method.

Python Console panel

This panel is used to use IronPython to control CS+ and the debug tool via the command input method.

Figure A.1   Python Console Panel
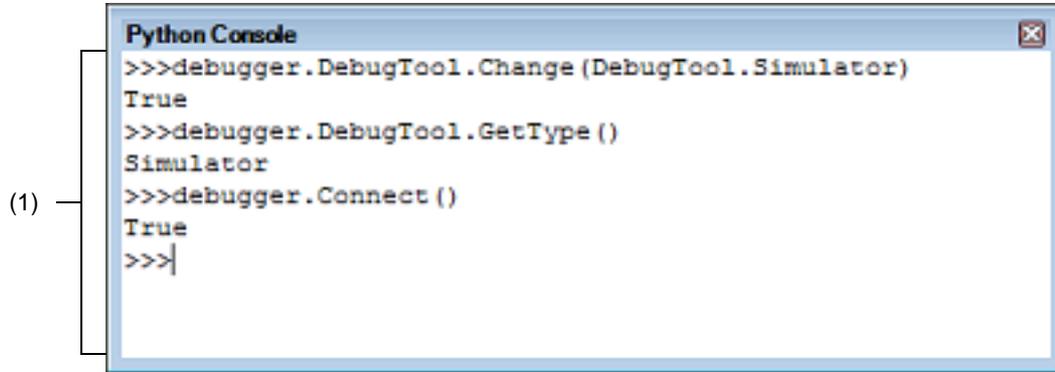
The following items are explained here.

- [How to open]

- [Description of each area]

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

- [Context menu]

[How to open]

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

[Description of each area]

(1)   I/O area
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.

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

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).

[Context menu]

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.

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 parame-
ter 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.

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")
```

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)]

Argument	Description
fileName	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.
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

Argument	Description
deviceName	Specify the device name of the microcontroller of a new project by a string.

Argument	Description
projectKind	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.Multicore-BootLoader 　When the microcontroller is multi-core and subproject: ProjectKind.MulticoreApplication

Type	Description
ProjectKind.Application	Project for application
ProjectKind.Library	Project for library
ProjectKind.DebugOnly	Debug-dedicated project
ProjectKind.Empty	Project for empty application
ProjectKind.CppApplication	Project for C++ application
ProjectKind.RI600V4	Project for RI600V4
ProjectKind.RI600PX	Project for RI600PX
ProjectKind.RI850V4	Project for RI850V4
ProjectKind.RI850MP	Project for RI850MP
ProjectKind.RI78V4	Project for RI78V4
ProjectKind.MulticoreBootLoader	Project for boot loader for multi-core
ProjectKind.MulticoreApplication	Project for application for multi-core
ProjectKind.Auto	The type of a project is selected in accord with the specification for *micomType*, *deviceName*, and *subProject* (default).

Argument	Description
compiler	Specify the compiler to be used. If the compiler is not specified, it is selected automatically depending on the microcontroller type.<table><tr><th>Type</th><th>Description</th></tr><tr><td>Compiler.Auto</td><td>The compiler to be used is selected in accord with the specification for *micomType* (default).</td></tr><tr><td>Compiler.CC_RH</td><td>CC-RH If this argument is not specified when *micomType* is set to "MicomType.RH850", CC-RH is selected automatically.</td></tr><tr><td>Compiler.CC_RX</td><td>CC-RX If this argument is not specified when *micomType* is set to "MicomType.RX", CC-RX is selected automatically.</td></tr><tr><td>Compiler.CA850</td><td>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.</td></tr><tr><td>Compiler.CX</td><td>CX If this argument is not specified when *micomType* is set to "MicomType.V850" and *deviceName* is set to "V850E2", CX is selected automatically.</td></tr><tr><td>Compiler.CC_RL</td><td>CC-RL If this argument is not specified when "MicomType.RL78" in CS+ for CC, CC-RL is selected automatically.</td></tr><tr><td>Compiler.CA78K0R</td><td>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.</td></tr><tr><td>Compiler.CA78K0</td><td>CA78K0 If this argument is not specified when *micomType* is set to "MicomType.K0", CA78K0 is selected automatically.</td></tr></table>
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
>>>
```

RENESAS

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.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.
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.

Function Name	Function Description
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.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.ReferTimer	This function displays the timer interrupt setting information.
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.
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.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.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.

Function Name	Function Description
debugger.Timer.Delete	This function deletes 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.IntelIdTag	This function saves the memory data in ID-tagged Intel format.
debugger.Upload.Motorola	This function saves the memory data in Motorola format.
debugger.Upload.MotorolaIdTag	This function saves the memory data in ID-tagged Motorola format.
debugger.Upload.Tektronix	This function saves the memory data in Techtronics format.
debugger.Upload.TektronixIdTag	This function saves the memory data in ID-tagged Techtronics 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.
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.

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-out-
put 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.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.E20Jtag	E20 (JTAG connect)
	DebugTool.E20Serial	E20 (Serial connect)
	DebugTool.E20Lpd	E20 (LPD connect)

[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 proj-
  ect 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)
E20Jtag	E20 (JTAG connect)
E20Serial	E20 (Serial connect)
E20Lpd	E20 (LPD connect)

[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. [RH850]

[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. [RH850]

[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   If two or more parameters are specified, then three parameters must be specified.
It is not possible to specify only *fileName* and *address*.

[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　　If two or more parameters are specified, then three parameters must be specified.
　　　　　　It is not possible to specify only *fileName* and *address*.

[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   If two or more parameters are specified, then three parameters must be specified.
It is not possible to specify only *fileName* and *address*.

[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　　If two or more parameters are specified, then three parameters must be specified.
　　　　　　　It is not possible to specify only *fileName* and *offset*.

[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   If two or more parameters are specified, then three parameters must be specified.
It is not possible to specify only *fileName* and *offset*.

[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   If two or more parameters are specified, then three parameters must be specified.
It is not possible to specify only *fileName* and *offset*.

[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   If two or more parameters are specified, then three parameters must be specified.
It is not possible to specify only *fileName* and *offset*.

[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)]

Argument	Description		
fileName	Specify a download file.		
downloadOption	Specify an option. The options that can be specified are shown below.		
	Type	Description	
	DownloadOption.NoSymbol	Do not load symbol information.	
	DownloadOption.SymbolOnly	Only load symbol information.	
	DownloadOption.Both	Load both symbol information and object information (default).	
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).		
vendorType	Specify the vendor of the compiler. The types that can be specified are shown below.		
	Type	Description	
	VendorType.Auto	Automatically specify the vendor of the compiler judging from the output contents of debugging information (default).	
	VendorType.Ghs	Make this specification when using a compiler made by Green Hills Software, Inc.	

[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.

RENESAS

[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   IECUBE, IECUBE2, and the simulator do not have functionality to delete code flash memory. For this reason, if you are using IECUBE, IECUBE2, or the simulator, you cannot omit *eraseOption*, or specify "EraseOption.Code".

[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*.

[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 breakNote 4	o	o	o	o	o	o	-	-	-	o
FetchGuard	Fetch guard breakNote 4	o	-	-	o	-	-	-	-	-	-
TraceStop	Trace stopNote 4	o	-	-	o	-	-	-	-	-	-
ExecutionFails	Execution failedNote 5	o	o	-	o	o	-	o	o	o	-

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.   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.

Break-trigger String	Description	RX		V850E2				RH850		
		E1Jtag, E1Serial E20Jtag, E20Serial	Simulator	Iecube2	Minicube[Note 2]	Minicube2[Note 1]	Simulator	Full-spec emulator	E1 / E20	SIM
None	No break	o	-	o	o	o	-	-	-	-
Manual	Forced break	o	o	o	o	o	o	o	o	o
Event	Break due to event	o	o	o	o	o	o	o	o	o
Software	Software break	o	-	o	o	o	-	o	o	-
TraceFull	Break due to trace full	o	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	-	-	-	-	-
FlashMacroService	Flash macro service active	-	-	o	o	o	-	-	-	-
Temporary	Temporary break	o	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	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	-	-	-	-	-	-	-

Break-trigger String	Description	RX		V850E2				RH850		
		E1Jtag, E1Serial E20Jtag, E20Serial	Simulator	Iecube2	Minicube[Note 2]	Minicube2[Note 1]	Simulator	Full-spec emulator	E1 / E20	SIM
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	-	-	-	-	-	-	-
CoverageMemoryAllo-cationFailure	Failed to allocate coverage memory	-	o	-	-	-	-	-	-	-
TraceMemoryAlloca-tionFailure	Failed to allocate trace memory	-	o	-	-	-	-	-	-	-
StepCountOver	Step count over	-	-	-	-	-	-	o	o	o
DebuggingInformation-AcquisitionFailure	Failed to acquire debugging information	-	-	-	-	-	-	o	o	o

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.

[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
Reset	In reset state
Standby	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
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.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 [78K0] [RL78] [78K0R] [IECUBE [V850]] [IECUBE2 [V850]]
	IeType.Dcu	DCU register [IECUBE [V850]]
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]

[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.<table><tr><td>Type</td><td>Description</td></tr><tr><td>EIVectorType.Standard</td><td>Standard mode (default)</td></tr><tr><td>EIVectorType.Expanded</td><td>Expanded mode</td></tr></table>

[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]

[Specification format]

```
debugger.Interrupt.OccurFE(feVectorType)
```

[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
	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

[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.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.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

Argument	Description

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)]

Argument	Description
mapType	Specify a memory type. The memory types that can be specified are shown below. <table><tr><th>Type</th><th>Description</th></tr><tr><td>MapType.EmulationRom</td><td>Emulation ROM area</td></tr><tr><td>MapType.EmulationRam</td><td>Emulation RAM area</td></tr><tr><td>MapType.Target</td><td>Target memory area</td></tr><tr><td>MapType.TargetRom</td><td>Target ROM area</td></tr><tr><td>MapType.Stack</td><td>Stack area</td></tr><tr><td>MapType.Protect</td><td>I/O protect area</td></tr></table>
address1	Specify a map start address.
address2	Specify a map end address.
accessSize	Specify an access size (bit) (default: 8). For V850, specify either 8, 16, or 32. For 78K0R [IECUBE], specify either 8 or 16.
cs	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.

[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.<table><tr><td>Type</td><td>Description</td></tr><tr><td>MemoryOption.Byte</td><td>Byte unit (8 bits) (default)</td></tr><tr><td>MemoryOption.HalfWord</td><td>Half-word unit (16 bits) [RH850,RX,V850]</td></tr><tr><td>MemoryOption.Word</td><td>Word unit (RL78,78K: 16 bits, RH850,RX,V850: 32 bits)</td></tr></table>

[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.

[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
[35, 67, 50, None, None]
```

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.<table><tr><td>Type</td><td>Description</td></tr><tr><td>MemoryOption.Byte</td><td>Byte unit (8 bits) (default)</td></tr><tr><td>MemoryOption.HalfWord</td><td>Half-word unit (16 bits) [RH850,RX,V850]</td></tr><tr><td>MemoryOption.Word</td><td>Word unit (RL78,78K: 16 bits, RH850,RX,V850: 32 bits)</td></tr></table>

[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.*

[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.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.

[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 program stops.
	RunOption.Normal	Breakpoints enabled; do not wait until 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.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.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.

[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.IntelIdTag

This function saves the memory data in ID-tagged Intel format.

[Specification format]

```
debugger.Upload.IntelIdTag(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 ID-tagged Intel format.

[Example of use]

```
>>>debugger.Upload.IntelIdTag("C:/test/testIdTagIntel.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.Upload.MotorolaIdTag

This function saves the memory data in ID-tagged Motorola format.

[Specification format]

```
debugger.Upload.MotorolaIdTag(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 ID-tagged Motorola format.

[Example of use]

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

debugger.Upload.Tektronix

This function saves the memory data in Techtronics format.

[Specification format]

```
debugger.Upload.Tektronix(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 Techtronics format.

[Example of use]

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

debugger.Upload.TektronixIdTag

This function saves the memory data in ID-tagged Techtronics format.

[Specification format]

```
debugger.Upload.TektronixIdTag(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 ID-tagged Techtronics format.

[Example of use]

```
>>>debugger.Upload.TektronixIdTag("C:/test/testIdTagTektronix.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)]

Argument	Description
variableName	Specify the variable name, register name, or I/O register name/SFR register name to reference.
encode	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
watchOption	Specify an option. The options that can be specified are shown below.

Type	Description
WatchOption.Auto	Automatically detect when displaying (default).
WatchOption.Binary	Display in binary format.
WatchOption.Octal	Display in octal format.
WatchOption.Decimal	Display in decimal format.
WatchOption.SignedDecimal	Display in signed decimal format.
WatchOption.UnsignedDecimal	Display in unsigned decimal format.
WatchOption.Hexdecimal	Display in hexadecimal format.
WatchOption.String	Display as a string.
WatchOption.Sizeof	Display the variable size in decimal format.
WatchOption.Float	Display in float type.
WatchOption.Double	Display in double type.

[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")
>>>
```

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.
BreakCondition	This class creates a break condition.
BreakpointInfo	This class holds break point information.
BuildCompletedEventArgs	This class holds the parameters when a build completes.
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.
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.
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

RENESAS

[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
```

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
>>>
```

```
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]

Variable	Description	
FileName	Specify the full path of the download file.	
DownloadFileType	Specify the type of the download file. The types that can be specified are shown below.	
	Type	Description
	DownloadFileType.LoadModule	Load module file (default)
	DownloadFileType.Hex	Intel HEX file
	DownloadFileType.SRecord	Motorola S-record file
	DownloadFileType.Binary	Binary file

DownloadObject	Specify whether to download object information.
	    True: Download object information.
	    False: Do not download object information.	

DownloadSymbol	This holds whether to download symbol information.
	    True: Download symbol information.
	    False: Do not download symbol information.	

VendorType	Specify the vendor of the compiler. The types that can be specified are shown below.	
	Type	Description
	VendorType.Auto	Automatically specify the vendor of the compiler judging from the output contents of debugging information (default).
	VendorType.Ghs	Make this specification when using a compiler made by Green Hills Software, Inc.

OutputInputCorrection	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.	

[Detailed description]

- "DownloadCondition" is in class format, and it is the structure of the parameter of the debugger.Download.Property property.

[Example of use]

```
>>>dc = DownloadCondition()
>>>dc.FileName = "C:/project/test.abs"
>>>dc.DownloadFileType = DownloadFileType.LoadModule
>>>dc.DownloadObject = True
>>>dc.DownloadSymbol = True
>>>dc.VendorType = VendorType.Auto
>>>debugger.Download.Property = [dc]
>>>
```

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
>>>
```

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]

Variable	Description	
StartAddress	Specify an address starting timer measurement. Must be specified.	
StartData	Specify a data condition (number) of an address starting timer measurement. This specification is ignored if "TimerType.Execution" is specified for Start-TimerType.	
StartTimerType	Specify the type of timers which start timer measurement. The types that can be specified are shown below.	
	Type	Description
	TimerType.Execution	Start a timer at execution (default)
	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	Specify the type of timers which end timer measurement. Must be specified.	
EndData	Specify a data condition (number) of an address ending timer measurement. This specification is ignored if "TimerType.Execution" is specified for EndTimer-Type.	
EndTimerType	Specify the type of timers which end timer measurement. The types that can be specified are shown below.	
	Type	Description
	TimerType.Execution	End a timer at execution (default)
	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]

- "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 exe-
cuted.

[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 Start-TraceType.		
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
```

[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.

[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.Refer-Timer 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.Compile.IncludePath	This property sets or refers to the compile options for the active project regarding additional include paths.
build.Compile.Macro	This property sets or refers to the compile options for the active project regarding defined macros.
build.IsBuilding	This property confirms whether a build is running.
build.Link.LibraryFile	This property sets or refers to library files of 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.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 refers to the version of the compiler package.

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.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.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.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.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.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 refers to the version of the compiler package.

[Specification format]

```
build.Version
```

[Setting(s)]

None

[Reference]

Version of compiler package used in active project

[Detailed description]

 - This property refers to the version of the compiler package used in the active project.

[Example of use]

```
>>>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.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.AccessStopExecution debugger.Option.AfterTraceMemoryFull debugger.Option.Coverage debugger.Option.OpenBreak debugger.Option.ReuseCoverageData debugger.Option.Timer debugger.Option.Trace 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.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

adConvertDataInExecu-tion	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.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.

[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.DownloadType = 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 function checks whether or not the microcontroller of the active project is multi-core.

[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.AccessStopExecution
debugger.Option.AfterTraceMemoryFull
debugger.Option.Coverage
debugger.Option.OpenBreak
debugger.Option.ReuseCoverageData
debugger.Option.Timer
debugger.Option.Trace
debugger.Option.UseTraceData
```

This property sets or refers to the options of the debug tool.

[Specification format]

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

[Setting(s)]

Setting	Description		
afterTrace	Set the operation to be taken after using up trace memory. The values that can be specified are shown below.		
	Value		Description
	AfterTraceMemoryFull.NoneStop		Overwrite trace memory and continue execution.
	AfterTraceMemoryFull.StopTrace		Stop tracing.
	AfterTraceMemoryFull.Stop		Stop execution (stop the program).
accessStopExecution	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.		
coverage	Set whether to use the coverage function. [IECUBE][IECUBE2][Simulator] 　True: Use the coverage function. 　False: Do not use the coverage function.		
openBreak	Set whether to use the open break function. 　True: Use the open break function. 　False: Do not use the open break function.		
reuseCoverageData	Set whether to reuse the coverage result. 　True: Reuse the coverage result. 　False: Do not reuse the coverage result.		
timer	Set whether to use the timer function. 　True: Use the timer function. 　False: Do not use the timer function.		

Setting	Description		
trace	Set whether to use the trace function. [IECUBE][IECUBE2][Simulator] 　True: Use the trace function. 　False: Do not use the trace function.		
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.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. [RH850]

[Specification format]

```
debugger.ProcessorElement = number
```

[Setting(s)]

Setting	Description
number	Set the PE number with the number.

[Reference]

Current set value

[Detailed description]

 - This function 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.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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Aug 01, 2015	-	First Edition issued

RENESAS

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2015 Renesas Electronics Corporation. All rights reserved.
Colophon 4.0

CS+ V3.02.00