To our customers,

---

**Old Company Name in Catalogs and Other Documents**

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: [http://www.renesas.com](http://www.renesas.com)

April 1st, 2010
Renesas Electronics Corporation

---

Issued by: Renesas Electronics Corporation ([http://www.renesas.com](http://www.renesas.com))

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.

2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.

4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.

6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.
Active X, Microsoft, MS-DOS, Visual Basic, Visual C++, Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

IBM and AT are registered trademarks of International Business Machines Corporation.

Intel and Pentium are registered trademarks of Intel Corporation.

Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.

All other brand and product names are trademarks, registered trademarks or service marks of their respective holders.

Keep safety first in your circuit designs!

- Renesas Technology Corporation and Renesas Solutions Corporation put the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation, Renesas Solutions Corporation or a third party.

- Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.

- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation and Renesas Solutions Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor for the latest product information before purchasing a product listed herein. The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors. Please also pay attention to information published by Renesas Technology Corporation and Renesas Solutions Corporation by various means, including the Renesas home page (http://www.renesas.com).

- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability or other loss resulting from the information contained herein.

- Renesas Technology semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.

- The prior written approval of Renesas Technology Corporation and Renesas Solutions Corporation is necessary to reprint or reproduce in whole or in part these materials.

- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination. Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.

- Please contact Renesas Technology Corporation or Renesas Solutions Corporation for further details on these materials or the products contained therein.

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

\SUPPORT\Product-name\SUPPORT.TXT

Renesas Tools Homepage http://www.renesas.com/en/tools
Abstract
The PDSDK COM kit is provided for the purpose of extending the functions of the M3T-PDxx and M3T-PDxxSIM debuggers. Using Windows application development tools available on the market, you can create the customize window (application) for the M3T-PDxx and M3T-PDxxSIM, and operate in conjunction with other applications.

This reference manual shows the basic information necessary to use the PDSDK COM kit. For details about the language specifications of and the method for using Windows application development tools, refer to the user's manual included with your product or online help.

Supported debuggers
The PDSDK COM kit cannot be used in all of the M3T-PDxx and M3T-PDxxSIM debuggers. For the debuggers and their versions which can be run in conjunction with the M3T-PDxx and M3T-PDxxSIM using the PDSDK COM kit, refer to the release notes for the PDSDK COM kit in which they are detailed.

Rights to use
The rights to use the PDSDK COM kit come under the provisions of the Software License Agreement for the M3T-PDxx or M3T-PDxxSIM debuggers used. Please also be aware that the PDSDK COM kit can only be used in developing your product, and cannot be used for any other purpose.

Technical support
Technical support for the PDSDK COM kit can be obtained by visiting the web site (URL: http://www.renesas.com/en/tools) at which latest information is available. Please note that we may not be able to reply to all of your questions or requests about the PDSDK COM kit.
mem_get_multi
mem_fill
mem_move
mem_set
mem_set_bit
mem_set_multi
rram_clear
rram_clear_blk
rram_get_area
rram_get_area_blk
rram_get_data
rram_set_area
rram_set_area_blk
reg_get_pc
reg_get_reg
reg_set_pc
reg_set_reg
rtt_check_isfetch
rtt_clear
rtt_get_bus
rtt_get_disasm
rtt_get_range
scrn_command
scrn_print
sym_add_bitsymbol
sym_add_label
sym_add_symbol
sym_add2line
sym_bit2val
sym_get_disp_src
sym_get Scope
sym_set disp src
sym_set_scope
sym_set_scope_addr
sym_set_scope_obj
sym_sym2val
sym_val2bit
sym_val2sym

5.3 Event Number List
1. Abstract

The PDSDK COM kit is provided to expand the function of debugger M3T-PDxx/M3T-PDxxSIM (hereinafter described as “PDxx”). The employment of this kit makes it possible to create the PDxx customized window and link PDxx with commercially available applications. The PDxx has added to it a new interface known as COM\(^1\). The functions of the PDxx can be used through this interface.

1.1 Development Tools Used

To create customize windows or operate in conjunction with other applications, you need to use Windows application development tools which support Microsoft’s Visual Basic or Visual C++ or other COM.

- Many reference books are available on the market, as is the information necessary to create applications.

- The kit comes standard with abundant GUI components. These GUI components can be used as simulate components for the user system. Freeware and shareware control components (ActiveX control) can also be used. Or you can create your original components using Visual Basic or Visual C++.

- The kit can be run in conjunction with applications which have COM interface. For example, Microsoft (R) Excel (hereafter simply Excel) has COM interface. This enables you to send RAM monitor results to Excel and display the results graphically, a feature which was not available with conventional PDxx.

Visual Basic, as is called the RAD (Rapid Application Development) tool, helps even the programming novice to create applications easily.

1.2 Methods To Be Called

Various methods can be called through the PDxx’s COM interface, including those to control execution of the microcomputer, set/reference memory or register contents, and set software breakpoints. Debugger functions can be extended in the same way as for CBxx, a original customize tool.

\[^1\] COM (Component Object Model), proposed by Microsoft, is a standard for connecting application programs.
2. Setup
For details on how to set up the PDSDK COM kit, refer to the Release Notes for the PDSDK COM kit.

3. Windows Creating (Visual Basic)
Explanation is given in this Chapter of the creation of customized windows using Visual Basic 6.0 (hereinafter described as “Visual Basic”). Before using Visual Basic, please be accustomed with its basic language specification and method of use by referring to Visual Basic online help or reference books available on the market.
3.1 Generating Project
Generate the new project: select Visual Basic Menu “File”, then “New Project”. The “New Project” dialog opens. With this dialog, select “Standard EXE” and click the mouse on the “OK” button.

By clicking the mouse on “OK”, project generation is completed.

3.2 Designating Type Library
To use the COM interface of PDxx, designate the type library file for PDxx. The type library file is such that the name of the method that the COM component is opening and its parameter are stored therein. The type library file name for PDxx is “pd_lite.tlb”, which is copied to the system directory of the Windows at PDSDK COM kit setup. To designate the type library, select Visual Basic Menu “Project”, then “References…”. The “Reference” dialog opens; therefore, find out the type library for PDxx from the list box and check the check box on that line, then click the mouse on the “OK” button. For the PDxx type library, “PD_LITE 1.0 Type Library” is displayed.

Type library designation must be set for each project of Visual Basic. Where the project of calling the COM interface of PDxx is newly created, be sure to conduct this work.
3.3 Generating Object

Describe as follows on the VB code window. This code is the basic one for accessing the COM interface of PDxx.

```vbnet
Dim WithEvents pd As PD_LITELib.pdcom

Private Sub Form_Load()
    Set pd = New PD_LITELib.pdcom
End Sub

Private Sub FormUnload(Cancel As Integer)
    Set pd = Nothing
End Sub
```

(Explanation of Each Line)

1st line: Here, it is declared that the type of variable pd is “PD_LITELib.pdcom”. This “PD_LITELib.pdcom” is the COM interface name. Where the method for PDxx that is compatible with emulator PC7501 is accessed, give the interface name as “PD_LITELib.Ipdcom 7501”. Also, designate the description of “ WithEvents” to obtain the event occurring on the PDxx side, such as Program execution start and program stop. It is possible to change the variable name pd to any character string. The variable name pd can be any name.

3rd to 5th lines: This procedure (function) is called at applications startup (form open). Here, the object of “PD_LITELib.pdcom” is substituted for variable pd. The method of PDxx is accessed via this variable pd.

7th to 9th lines: This procedure (function) is accessed at applications end (form closing). Here, the object of variable pd is cancelled. If the object is cancelled, it will become impossible to call the method of PDxx.

This code must be described for each project of Visual Basic. To newly create the program of accessing the COM interface of PDxx, be sure to do this work as in “3.2 Designating Type Library”.
3.4 Method Access

According to the contents given in para. 3.1, 3.2 and 3.3 above, prepare for COM interface access in advance.

The methods which are open for the PDxx can be called by writing "." after the object variable pd for the PDxx defined in Section 3.3 and then specifying a method name. Where the parameter is present in that method, describe that method after the method name.

\[
\text{pd.xxxxx ( Parameter, … )}
\]

For the details on each method, refer to “4.”
Method List. Explanation is given below of the specific procedure for calling the method.

**Reset User Target:**
The following is the method of preparing the customized window for resetting the user target. (The sample is stored in the directory in which you installed the kit.) In this application, a single button control is used.

1. **Adding Button Control**
   - Click the mouse on the Command button of the tool box to create one button control on the form.
2. Button Property Change
   In the property window, alter the properties for the button control you created.

<table>
<thead>
<tr>
<th>Property</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Name)</td>
<td>btnReset</td>
</tr>
<tr>
<td>Caption</td>
<td>Reset</td>
</tr>
</tbody>
</table>

3. Describe the button operation
   Describe as follows in the code window. The bold-faced place indicates the additional part.

   ```vba
   Dim WithEvents pd As PD_LITELib.pdcom
   Private Sub Form_Load()
       Set pd = New PD_LITELib.pdcom
   End Sub

   Private Sub Form_Unload(Cancel As Integer)
       Set pd = Nothing
   End Sub

   Private Sub btnReset_Click()
       ret = pd.cpu_reset
       If ret = 0 Then
           pd.err_disp_message
       End If
   End Sub
   ```

4. Operation Check
   Check for applications performance. First, startup “PDxx”. Next, select Visual Basic Menu [EXEC], then [START], and execute applications. By clicking the mouse on the Reset button in the applications, the user target is reset. Check the operation. To create execution files (EXE file) for this application, choose "Create xxxx.exe" from the [File] menu of Visual Basic. Specify the folder in which to save and the EXE file name. This application can be executed directly from Explorer.
3.5 Acquisition of Event

According to the contents given in para. 3.1, 3.2 and 3.3 above, prepare for COM interface access in advance.

To obtain an event (program execution start, etc.) arising on the PDxx side, use Procedure pd_GotEventMessage. This procedure is the subroutine called out when the event arising on the PDxx side was received. (“pd” at the head of the procedure name denotes the variable name of the object designated at the head of the program.)

```vbscript
Private Sub pd_GotEventMessage(ByVal action As Long)

End Sub
```

Event No. arising on the PDxx side is stored in parameter action. With this procedure, describe the processing to be executed where the event was received from PDxx. For details about the generated event numbers, refer to Section 5.3, "Event Number List."

Get an event arising on PDxx side:

The following is the method of creating the customized window for getting an event arising on the PDxx side and displaying its number. For this application, one label control is used.

1. Add Label Control
   Click the mouse on “Label” of the tool box and create one label control on the form.
2. Change the property
In the property window, alter the properties for the label control you created.

<table>
<thead>
<tr>
<th>Property</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Name)</td>
<td>lblEventNo</td>
</tr>
<tr>
<td>Caption</td>
<td>(nothing)</td>
</tr>
</tbody>
</table>

3. Describe the operation when an event occurred
Describe as follows in the code window. The bold-faced place indicates the addition part.

```vba
Dim WithEvents pd As PD_LITELib.pdcom

Private Sub Form_Load()
    Set pd = New PD_LITELib.pdcom
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Set pd = Nothing
End Sub

Private Sub pd_GotEventMessage(ByVal action As Long)
    lblEventNo.Caption = action
End Sub
```

4. Check for action
Check the applications action. First, run PDxx. Next, select the “Visual Basic” menu [EXEC], then [START], and execute the applications. By manipulating PDxx (go/stop, etc.), the number of an event that occurred in its PDxx is displayed. Check its action.
4. Example of creating PDSDK-Window (Visual C++)

This chapter explains the creation method of the customize window for PDxx using Visual C++ 6.0 (hereinafter described as "VC++"). Before using VC++, please be accustomed with its basic language specification and method of use by referring to VC++ online help or reference books available on the market.

The sample which carries out PDxx RESET is created in this chapter.

4.1 Generating a project

To create a new project select NEW from VC++ FILE-menu. This will open the New Project dialog. Please select MFC AppWizard and push OK.

In this example, the project name is assumed to be "Reset".

The next step defines the application type. Please select Dialog Based Application and select the appropriate language. Then push NEXT.
In step 2 please select AUTOMATION. The other settings can be left on their default setting. Then push again NEXT.

Then you can push FINISH to exit the Application Wizard. All other settings remain on default settings.

4.2 Creating a Button
After creating the new project, the VC++ desktop shows the empty dialog window for our RESET application. In this window you can place various objects from the Control box on your application. In this example we simply want to create a push button. Therefore you have to select the button symbol in the control box and draw a button of the desired size in your application window.
Now you select the button with a single click on it and go to VIEW-PROPERTIES. This opens a window with the button properties. Here you should change the ID and the caption to something relating to the function of the button. In this case we selected RESET.

The text box below the button can be used to receive messages from PDxx. Therefore we also first change its references. Select the properties of the text box and change ID and caption as shown below.

4.2.1 Message Handling
In order to create a function, which relates to the new Reset button, you have to open the pull-down menu VIEW and select the CLASS-WIZARD of VC++.

Here you find the ID of the new button, and when you mark this ID you get a number of actions, which can be used. Select BN_CLICKED (Button Clicked) and push Add Function to create a function, which is called on a single click of the new button.
void CResetDlg::OnReset()
{
    // TODO: Add your control notification handler code here
}

This new function is added to the source file of this dialog window, ResetDlg.cpp. Now we have to make some modifications by hand so we can later communicate with the emulator.

#import "PD_LITE.TLB" no_namespace , named_guids   (1)
...  
...  
void CResetDlg::OnReset()
{
    // TODO: Add your control notification handler code here
    IpdcomPtr p(__uuidof(pdcom));    (2)
    p->cpu_reset();      (3)
}

(1) Import the library of PDCOM. This is necessary to be able to use the PDCOM interface and its methods.  
(2) “p” is the smart pointer of the PDCOM interface. A smart pointer name is I interface name Ptr. Since the interface name of PDCOM is pdcom, a smart pointer serves as IpdcomPtr.  
(3) Call the method cpu_reset().

4.3 Acquisition of Events from PDxx
In order to communicate with the PDxx debugger (start/stop target program, read/modify memory or register contents, ...) it is necessary to acquire an event from PDxx. This chapter explains how to mount the acquisition function of an event in the “Reset” window of the previous chapter.

4.3.1 Preparation of Creation of Sink Object (ATL support)
The object (sink object), which receives a messages from PDxx, is created from a server. Since a sink object is created using the function of ATL, it is necessary to add ATL support to the project. From the INSERT menu of VC++ select NEW ATL OBJECT. This opens a dialog asking of you want to add ATL support to your project. Here you select YES.

Next the "ATL object wizard" dialog opens. Here you just have to push CANCEL.
### 4.3.2 Creation of a sink object
A sink object is created by hand. The header file of a sink object is as follows (file name: ResetEvents.h).

```c
#ifndef __RESETEVENTS_H_
#define __RESETEVENTS_H_

#include "resource.h"

class CpdcomEvents :
    public CComObjectRoot,
    public IDispatchImpl<_IpdcomEvents, &DIID__IpdcomEvents, &LIBID_PD_LITELib> (1)
{
public:
    CpdcomEvents() : m_hWnd(NULL) {}

BEGIN_COM_MAP(CpdcomEvents)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY_IID(DIID__IpdcomEvents, IDispatch)
END_COM_MAP()

// Idispatch
STDMETHOD(Invoke)(DISPID dispid, REFIID riid, LCID lcid,      (2)
    WORD wFlags, DISPPARAMS* pdispParams, VARIANT* pvarResult,
    EXCEPINFO* pexcepinfo, UINT* puArgErr);

void SetHWND(HWND hWnd) { m_hWnd = hWnd; }          (3)
private:
    HWND m_hWnd;                     (4)
public:
};
#endif
```

1. IpdcomEvents is the sink object mounted with the server (PDxx).
2. Invoke() is the function called by the server. Processing when an event is notified to this function is mounted.
3. It is the variable which holds the handle of the dialog created this time to member variable m_hWnd. It is not directly related to COM and ATL.
4. The handle of a dialog is set as member variable m_hWnd by SetHWND().
The mounting of a sink object is as follows (file name: ResetEvents.cpp).

```cpp
#include "stdafx.h"
#include "pdLite.tlb" no_namespace, named_guids
#include "ResetEvents.h"
#include "ResetDlg.h"

STDMETHODIMP CpdcomEvents::Invoke(DISPID dispid, REFIID riid, LCID lcid,
WORD wFlags, DISPPARAMS* pdispParams,
VARIANT* pvarResult, EXCEPINFO* pexcepinfo, UINT* puArgErr )
{
    if(riid!= IID_NULL)
        return DISP_E_UNKNOWNINTERFACE;
    HRESULT hr;
    CComVariant varMessage;
    switch (dispid ) {
    case 1:
        if( ! (wFlags & DISPATCH_METHOD))
            return DISP_E_MEMBERNOTFOUND;
        if(pdispParams->cArgs != 1)
            return DISP_E_BADPARAMCOUNT;
        hr = varMessage.ChangeType(VT_BSTR, (&pdispParams->rgvarg[0]));
        if(FAILED(hr))
            return DISP_E_TYPEMISMATCH;
        SetDlgItemText(m_hWnd, IDC_EVENT_NO, OLE2T(varMessage.bstrVal));
        return S_OK;
        default:
            return DISP_E_MEMBERNOTFOUND;
    }
}
```

(1) If a named_guids attribute is specified, an old style will define the GUID variable of the form of LIBID_MyLib, CLSID_MyCoClass, IID_MyInterface, and DIID_MyDispInterface, and the compiler will initialize it.

(2) The event number of PDxx is stored in pdispParams->rgvarg[0]. In this example, in order to display a character on a dialog, it has changed into the VT_BSTR type using ChangeType. It describes like hr = varMessage.ChangeType(VT_I4); (&pdispParams->rgvarg[0]) to change into a numerical value (when changing into VT_I4 and long).

(3) OLE2T change LPOLESTR into LPTSTR and are macroscopic.

(4) In this example, it is displaying on the dialog by making an event into a character sequence using SetDlgItemText() and outputs it in the text box IDC_EVENT_NO of the dialog.

These new files ResetEvents.cpp and ResetEvents.h have to be added to the project.

4.3.3 Change of a Dialog SW

The source of the dialog has to be changed in order to use the sink object. The following declarations are added to the header file ResetDlg.H (underlined part).

```cpp
class CresetDlgAutoProxy;
interface Ipdcmd;
Class CpdcmdEvents;
```

Add a new member variable to ResetDlg.H:

```cpp
private:
    CComPtr <Ipdcmd> m_ppdcom;
    CComObject<CpdcmdEvents>* m_ppdcmdEvents;
    DWORD m_dwCookie;
```
**PDSDK - COM Kit**

m_ppdcom is the discernment child (Cookie) to whom the pointer of an interface and m_ppdcomEvents decide connection of a sink object and a server to be a sink object, and m_dwCookie decides it to be a meaning.

Now the source file ResetDlg.CPP has to be modified.

Import the type library for the PDCOM interface and include the header file of the sink object.

```cpp
#import "PD_LITE.TLB" no_namespace, named_guids
#include "ResetEvents.h"
```

The code which initializes a sink object is added to the constructor of the dialog (underlined part).

```cpp
CResetDlg::CResetDlg(CWnd* pParent /*=NULL*/) : CDialog(CResetDlg::IDD, pParent)
{
    //AFX_DATA_INIT(CResetDlg)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_pAutoProxy = NULL;
    m_ppdcomEvents = NULL;
}
```

The code canceling the connection is inserted in the destructor.

```cpp
CResetDlg::~CResetDlg()
{
    // If there is an automation proxy for this dialog, set
    // its back pointer to this dialog to NULL, so it knows
    // the dialog has been deleted.
    if (m_pAutoProxy != NULL)
        m_pAutoProxy->m_pDialog = NULL;

    if (m_ppdcom != NULL) {
        AtlUnadvise(m_ppdcom, DIID__Ipdcmevents, m_dwCookie);
    }
    m_ppdcomEvents = NULL;
    m_ppdcom = NULL;
}
```
The code of acquisition of creation of an interface object, creation of a sink object, and an IUnknown interface is inserted in OnInitDialog().

```cpp
BOOL CResetDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.
    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);     // Set big icon
    SetIcon(m_hIcon, FALSE);    // Set small icon

    // TODO: Add extra initialization here
    HRESULT hr = CoCreateInstance(CLSID_pcom, NULL, CLSCTX_ALL, IID_Ipdcom,
    (void**) &m_ppdcom);
    if (FAILED(hr))
    {
        PostMessage(WM_CLOSE, 0, 0L);
        return TRUE;
    }

    CComObject<CpcomEvents>::CreateInstance(&m_ppdcomEvents);
    if (!m_ppdcomEvents)
    {
        m_ppdcom = NULL;
        PostMessage(WM_CLOSE, 0, 0L);
        return TRUE;
    }

    CComPtr<IUnknown> pEventUnk = m_ppdcomEvents;

    hr = AtlAdvise(m_ppdcom, pEventUnk, DIID__IpdcomEvents, &m_dwCookie);
    if (FAILED(hr))
    {
        m_ppdcom = NULL;
        pEventUnk = NULL;
        PostMessage(WM_CLOSE, 0, 0L);
        return TRUE;
    }

    m_ppdcomEvents->SetHWND(m_hWnd);
    return TRUE;  // return TRUE unless you set the focus to a control
}
```

`m_ppdcomEvents->SetHWND (m_hWnd)` has set up the handle of a dialog in order to display a character sequence on a dialog. It is not directly related to COM and ATL.
5. Method List

"PDxx" is disclosing the following method (function). For method specification details, refer to “5.2 Method details”. Sample use described in each method below indicates the case where “Visual Basic V. 6.0” by Microsoft Corporation was employed.

5.1 Method Outline

### CPU Control

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_go</td>
<td>(nothing)</td>
<td>Executes the target program from the current PC in free-run mode.</td>
<td>33</td>
</tr>
<tr>
<td>cpu_gb</td>
<td>(nothing)</td>
<td>Executes the target program from the current PC with breaks included.</td>
<td>32</td>
</tr>
<tr>
<td>cpu_wait</td>
<td>(nothing)</td>
<td>Waits for until the target program stops.</td>
<td>41</td>
</tr>
<tr>
<td>cpu_stop</td>
<td>(nothing)</td>
<td>Stops the target program.</td>
<td>40</td>
</tr>
<tr>
<td>cpu_reset</td>
<td>(nothing)</td>
<td>Reset the target program.</td>
<td>35</td>
</tr>
<tr>
<td>cpu_step</td>
<td>(nothing)</td>
<td>Step executes the target program, one instruction at a time, beginning with the current PC.</td>
<td>39</td>
</tr>
<tr>
<td>cpu_src_step</td>
<td>(nothing)</td>
<td>Step executes the target program, one source line at a time, beginning with the current PC.</td>
<td>38</td>
</tr>
<tr>
<td>cpu_over</td>
<td>(nothing)</td>
<td>Step executes the target program, one instruction at a time including subroutines, beginning with the current PC.</td>
<td>34</td>
</tr>
<tr>
<td>cpu_src_over</td>
<td>(nothing)</td>
<td>Over step executes the target program, one source line at a time including subroutines, beginning with the current PC.</td>
<td>37</td>
</tr>
<tr>
<td>cpu_return</td>
<td>(nothing)</td>
<td>Execution to return from the current PC to the calling routine, one instruction at a time.</td>
<td>36</td>
</tr>
<tr>
<td>cpu_check_run</td>
<td>[out] long *status</td>
<td>Checks the state of the target program.</td>
<td>31</td>
</tr>
</tbody>
</table>

### Register

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_get_pc</td>
<td>[out] long * pc</td>
<td>Gets the value of program counter.</td>
<td>78</td>
</tr>
<tr>
<td>reg_set_pc</td>
<td>[in] long pc</td>
<td>Sets a program counter value.</td>
<td>80</td>
</tr>
<tr>
<td>reg_get_reg</td>
<td>[in] long regNo [out] long * regVal</td>
<td>Gets the value of specified register.</td>
<td>79</td>
</tr>
</tbody>
</table>

### Memory

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem_get</td>
<td>[in] long addr [in] long length [out] long * data</td>
<td>Gets a memory data. (1 data)</td>
<td>63</td>
</tr>
<tr>
<td>Method Name</td>
<td>Parameter</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>mem_get_bit</td>
<td>[in] long addr</td>
<td>Gets a bit data.</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>[in] long bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mem_set_bit</td>
<td>[in] long addr</td>
<td>Sets a bit data.</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>[in] long bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mem_clear_bit</td>
<td>[in] long addr</td>
<td>Clears a bit data.</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>[in] long bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mem_fill</td>
<td>[in] long addr</td>
<td>Fills the memory data.</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>[in] long num</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mem_move</td>
<td>[in] long addr</td>
<td>Transfers the memory data.</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>[in] long num</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long top</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### RAM Monitor

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rram_get_area</td>
<td>[out] long* addr</td>
<td>Gets the beginning address of the RAM monitor area.</td>
<td>73</td>
</tr>
<tr>
<td>rram_set_area</td>
<td>[in] long addr</td>
<td>Sets the beginning address of the RAM monitor area.</td>
<td>76</td>
</tr>
<tr>
<td>rram_get_data</td>
<td>[in] long addr</td>
<td>Gets the RAM monitor data and the access attributes.</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>[in] long num</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] VARIANT* data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] VARIANT* attr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rram_clear</td>
<td>(nothing)</td>
<td>Initializes access states of the RAM monitor area.</td>
<td>71</td>
</tr>
<tr>
<td>rram_get_area_blk*</td>
<td>[in] long blkno</td>
<td>Gets the beginning address of the specified RAM monitor block.</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>[out] long* addr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rram_set_area_blk*</td>
<td>[in] long blkno</td>
<td>Sets the beginning address of the specified RAM monitor block.</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>[in] long addr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rram_clear_blk*</td>
<td>[in] long blkno</td>
<td>Initializes access states of the specified RAM monitor block.</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* This method is supported depending on PDxx to be used. For details, please refer to “PDSDK COM Kit Release Note.”

### Software Breaks

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>break_set</td>
<td>[in] long addr</td>
<td>Registers the software breakpoint.</td>
<td>30</td>
</tr>
<tr>
<td>break_reset</td>
<td>[in] long addr</td>
<td>Deletes the software breakpoint.</td>
<td>27</td>
</tr>
<tr>
<td>break_reset_all</td>
<td>(nothing)</td>
<td>Deletes all software breakpoints.</td>
<td>28</td>
</tr>
<tr>
<td>break_disable</td>
<td>[in] long addr</td>
<td>Disables the software breakpoint.</td>
<td>23</td>
</tr>
<tr>
<td>break_disable_all</td>
<td>(nothing)</td>
<td>Disables all software breakpoints.</td>
<td>24</td>
</tr>
<tr>
<td>break_enable_all</td>
<td>(nothing)</td>
<td>Enables all software breakpoints.</td>
<td>25</td>
</tr>
<tr>
<td>break_get</td>
<td>[out] long* addr</td>
<td>Searches the software breakpoint.</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>[out] long* attr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>break_search</td>
<td>[in] long addr</td>
<td>Gets the breakpoint attribute.</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>[out] long* attr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Real-time Trace

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtt_get_range</td>
<td>[out] long *s_cycle;</td>
<td>Gets the trace range of trace area.</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>[out] long *e_cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rtt_get_bus</td>
<td>[in] long cycle;</td>
<td>Gets the BUS character string of specified real-time trace cycle.</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>[out] long *addr;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rtt_get_disasm</td>
<td>[in,out] long *cycle;</td>
<td>Gets the disassemble character string of specified real-time trace cycle.</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>[out] long* next_cycle;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rtt_check_isfetch</td>
<td>[in] long cycle;</td>
<td>Checks fetch cycle of specified real-time trace cycle.</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>[out] long* addr1;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* addr2;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rtt_clear</td>
<td>(nothing)</td>
<td>Initializes the trace data.</td>
<td>83</td>
</tr>
</tbody>
</table>

### Coverage

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cv_get_base*</td>
<td>[out] long * base</td>
<td>Gets the beginning address of the coverage area.</td>
<td>46</td>
</tr>
<tr>
<td>cv_set_base*</td>
<td>[in] long base</td>
<td>Sets the beginning address of the coverage area.</td>
<td>52</td>
</tr>
<tr>
<td>cv_get_data</td>
<td>[in] long st_addr;</td>
<td>Gets the coverage data.</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>[in] long en_addr;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long *rs_addr;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long *re_addr;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] VARIANT *data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cv_clear*</td>
<td>(nothing)</td>
<td>Clears the coverage data.</td>
<td>44</td>
</tr>
<tr>
<td>cv_get_blkcnt*</td>
<td>[out] long *cnt</td>
<td>Gets the coverage block count.</td>
<td>49</td>
</tr>
<tr>
<td>cv_clear_blk*</td>
<td>[in] long blkno</td>
<td>Clears the specified coverage block data.</td>
<td>45</td>
</tr>
<tr>
<td>cv_get_base_all*</td>
<td>[out] VARIANT *base</td>
<td>Gets the beginning address of the all coverage block.</td>
<td>47</td>
</tr>
<tr>
<td>cv_get_base_blk*</td>
<td>[in] long blkno;</td>
<td>Gets the beginning address of the specified coverage block.</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>[out] long *base</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* This method is supported depending on PDxx to be used. For details, please refer to “PDSDK COM Kit Release Note.”

### Symbol

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp_eval</td>
<td>[in] BSTR str;</td>
<td>Gets the value that corresponds to a string character symbol.</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>[in] long radix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_sym2val</td>
<td>[in] long mode;</td>
<td>Gets the value that corresponds to a string character.</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>[in] BSTR symbol</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_val2sym</td>
<td>[in] long mode;</td>
<td>Gets the symbol that corresponds to value.</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>[in] long value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR* symbol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_bit2val</td>
<td>[in] BSTR symbol;</td>
<td>Gets the address and the bit number corresponding to the specified bit symbol.</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>[out] long* addr</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_val2bit</td>
<td>[in] long addr;</td>
<td>Gets the bit symbol that corresponds to address and bit number.</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>[in] long bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *symbol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_addr2line</td>
<td>[in] long addr</td>
<td>Gets the source file name and the line number</td>
<td>92</td>
</tr>
</tbody>
</table>
### Method List

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sym_line2addr</td>
<td>[in] BSTR filename</td>
<td>Gets the address of specified source line information.</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>[in] long line</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long* addr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_add_label</td>
<td>[in] BSTR name</td>
<td>Enters the label as a global label.</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>[in] long value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_add_symbol</td>
<td>[in] BSTR name</td>
<td>Enters the symbol as a global symbol.</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>[in] long value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_add_bitsymbol</td>
<td>[in] BSTR name</td>
<td>Enters the bit symbol as a global bit symbol.</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>[in] long addr</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_get_disp_src</td>
<td>[out] BSTR *filename</td>
<td>Gets the displaying source file name in the program window.</td>
<td>94</td>
</tr>
<tr>
<td>sym_set_disp_src</td>
<td>[in] BSTR filename</td>
<td>Changes the source file at program window.</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>[in] long line</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_set_scope</td>
<td>[in] long addr</td>
<td>Gets the scope (object file name) of specified address.</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *filename</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_set_scope_addr</td>
<td>[in] long addr</td>
<td>Sets the current scope of specified address.</td>
<td>103</td>
</tr>
<tr>
<td>sym_set_scope_obj</td>
<td>[in] BSTR obj</td>
<td>Sets the current scope of specified object file.</td>
<td>104</td>
</tr>
<tr>
<td>sym_get_obj_name*</td>
<td>[out] BSTR *obj</td>
<td>Get object names from a load module.</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>[in] long mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_get_src_name*</td>
<td>[in] BSTR obj</td>
<td>Get source file names from an object.</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *src</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_get_func_name*</td>
<td>[in] BSTR obj</td>
<td>Get function names from an object.</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *func</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[in] long mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_get_func_info*</td>
<td>[in] BSTR func</td>
<td>Get the information of a specified function.</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long *start</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long *end</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *str</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sym_get_variable_info*</td>
<td>[in] BSTR var</td>
<td>Get the information of a specified C variable.</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long *i_data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] long *h_data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *str1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *str2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* This method is supported depending on PDxx to be used. For details, please refer to "PDSDK COM Kit Release Note."

### Downloads

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>down_load</td>
<td>[in] BSTR filename</td>
<td>Downloads the target program.</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>[in] long mode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MCU/ PDxx information

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>info_cpu</td>
<td>[in] long flag</td>
<td>Gets the information of target MCU.</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>[out] long* status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>info_service</td>
<td>[in] long flag</td>
<td>Gets the supported information of PDxx.</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>[out] long* status</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Others

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>com_send</td>
<td>[in] BSTR data</td>
<td>Sends a data to Emulator.</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>[in] long size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>com_receive</td>
<td>[out] BSTR* data</td>
<td>Receives a data from Emulator.</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>[in] long size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>err_disp_message</td>
<td>(nothing)</td>
<td>Displays the PDxx error message.</td>
<td>54</td>
</tr>
<tr>
<td>err_get_message*</td>
<td>[out] BSTR *msg</td>
<td>Get error strings that occurred in a method call.</td>
<td>55</td>
</tr>
<tr>
<td>event_request_mode*</td>
<td>[in] long mode</td>
<td>Sets event request mode</td>
<td>56</td>
</tr>
<tr>
<td>scri_command*</td>
<td>[in] BSTR string</td>
<td>Executes the script command</td>
<td>87</td>
</tr>
<tr>
<td>scri_print*</td>
<td>[in] BSTR string</td>
<td>Displays the string to Script Window</td>
<td>88</td>
</tr>
</tbody>
</table>

* This method is supported depending on PDxx to be used. For details, please refer to "PDSDK COM Kit Release Note."

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>info_get_debugger*</td>
<td>[out] BSTR *name</td>
<td>Get the debugger information such as a product name.</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *dir</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[out] BSTR *ini</td>
<td></td>
<td></td>
</tr>
<tr>
<td>info_get_mcufile_name*</td>
<td>[out] BSTR *name</td>
<td>Get the MCU file name selected in the INIT dialog box.</td>
<td>60</td>
</tr>
</tbody>
</table>

* This method is supported depending on PDxx to be used. For details, please refer to "PDSDK COM Kit Release Note."
5.2 Method details
break_disable

Description
This function disables the software breakpoint.

Parameters
\[
\text{ret} = \text{pd.break_disable( addr )}
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Breakpoint address</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

Dim ret as Long
Dim addr as Long

addr = &hF0000
\[
\text{ret} = \text{pd.break_disable( addr )}
\]
If ret = 0 Then
pd.err_disp_message
End If
**break_disable_all**

**Description**
This function disables all software breakpoints.

**Parameters**

```vba
ret = pd.break_disable_all()
```

There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```
Dim ret as Long

ret = pd.break_disable_all
If ret = 0 Then
    pd.err_disp_message
End If
```
break_enable_all

Description
This function enables all software breakpoints.

Parameters
ret = pd.break_enable_all()

There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example
```
Dim ret as Long

ret = pd.break_enable_all
If ret = 0 Then
    pd.err_disp_message
End If
```
break_get

Description
This function searches the software breakpoint.

Parameters
\( \text{ret} = \text{pd.break_get} ( \text{addr}, \text{attr}, \text{mode} ) \)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>long *addr</td>
<td>Breakpoint address</td>
</tr>
<tr>
<td>[out]</td>
<td>long *attr</td>
<td>Setup attributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256 : IN1_ENABLE_SBRK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>257 : IN1_DISABLE_SBRK</td>
</tr>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Search mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>258 : (IN1_FIRST)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>259 : (IN1_NEXT)</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```plaintext
Public Const IN1_ENABLE_SBRK = 256
Public Const IN1_DISABLE_SBRK = 257
Public Const IN1_FIRST = 258
Public Const IN1_NEXT = 259
Dim ret as Long
Dim addr(64) as Long
Dim attr(64) as Long
Dim mode as Long
Dim i as Integer

ret = pd.break_get ( addr(0), attr(0), IN1_FIRST )
If ret = 0 Then
    pd.err_disp_message
End If

For i = 1 To 63
    ret = hc.break_get(addr(i), attr(i), IN1_NEXT)
    If ret = 0 Then
        GoTo next_step
    End If
Next i%
next_step:
```
break_reset

Description
This function deletes the software breakpoint.

Parameters
ret = pd.break_reset ( addr )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Breakpoint address</td>
</tr>
</tbody>
</table>

Returned value

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded</td>
<td>1</td>
</tr>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim addr as Long

addr = &hF0000
ret = pd.break_reset ( addr )
If ret = 0 Then
    pd.err_disp_message
End If
```
**break_reset_all**

**Description**
This function deletes all software breakpoints.

**Parameters**
```
ret = pd.break_reset_all ()
```
There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```
Dim ret as Long

ret = pd.break_reset_all
If ret = 0 Then
    pd.err Disp_message
End If
```
**break_search**

**Description**
This function gets the breakpoint attribute.

**Parameters**
ret = pd.break_search ( addr, attr )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Check address</td>
</tr>
<tr>
<td>[out]</td>
<td>long *attr</td>
<td>Setting of breakpoint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>256: (IN1_ENABLE_SBRK)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>257: (IN1_DISABLE_SBRK)</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**

```vbnet
Public Const IN1_ENABLE_SBRK = 256
Public Const IN1_DISABLE_SBRK = 257
Dim ret as Long
Dim addr as Long
Dim attr as Long
'
addr = &hF0000
ret = pd.break_search ( addr, attr )
If ret = 0 Then
    pd.err_disp_message
End If
If attr = IN1_ENABLE_SBRK Then
    '
Else
    '
End If
```
**break_set**

**Description**
This function registers the software breakpoint.

**Parameters**

```
ret = pd.break_set ( addr )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Breakpoint address</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```
Dim ret as Long
Dim addr as Long

addr = &hF0000
ret = pd.break_set ( addr )
If ret = 0 Then
    pd.err_disp_message
End If
```
cpu_check_run

Description
This function checks the state of the target program.

Parameters
\[ ret = \text{pd.cpu_check_run}(\text{status}) \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>long *status</td>
<td>Status of target program</td>
</tr>
<tr>
<td></td>
<td></td>
<td>260 : (IN1_RUN_CPU)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>261 : (IN1_STOP_CPU)</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim status as Long

ret = pd.cpu_check_run(status)
If ret = 0 Then
    pd.err_disp_message
End If
```
**cpu_gb**

**Description**
This function executes the target program from the current PC with breaks included.

**Parameters**
\[\text{ret} = \text{pd.cpu_gb}()\]

There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vba
Dim ret as Long

ret = pd.cpu_gb
If ret = 0 Then
    pd.err_disp_message
End If
```
cpu_go

Description
This function executes the target program from the current PC in free-run mode.

Parameters
ret = pd.cpu_go()
There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```vbnet
Dim ret as Long

   ret = pd.cpu_go
If ret = 0 Then
   pd.err_disp_message
End If
```


**cpu_over**

**Description**
This function step executes the target program, one instruction at a time including subroutines, beginning with the current PC.

**Parameters**

```vba
ret = pd.cpu_over()
```

There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vba
Dim ret as Long
ret = pd.cpu_over
If ret = 0 Then
    pd.err_disp_message
End If
```
cpu_reset

**Description**
This function reset the target program.

**Parameters**

```
ret = pd.cpu_reset()
```

There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
dim ret as long

ret = pd.cpu_reset
if ret = 0 then
  pd.err_disp_message
end if
```

35
**cpu_return**

**Description**
This function causes program execution to return from the current PC to the calling routine, one instruction at a time.

<table>
<thead>
<tr>
<th>Products</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD32R(SIM)</td>
<td>JMP R14, RTE</td>
</tr>
<tr>
<td>PD308(SIM),</td>
<td>REIT, RTS, FREIT, EXITD</td>
</tr>
<tr>
<td>PD30(SIM)</td>
<td></td>
</tr>
<tr>
<td>PD79(SIM)</td>
<td>RTS, RTI, RTL, RTSD, RTLSD</td>
</tr>
<tr>
<td>PD77(SIM)</td>
<td>RTS, RTI, RTL</td>
</tr>
<tr>
<td>PD38(SIM)</td>
<td>RTS, RTI</td>
</tr>
<tr>
<td>PD10V</td>
<td>JMP R13, RTE</td>
</tr>
</tbody>
</table>

**Parameters**

\[
\text{ret} = \text{pd.cpu_return}()
\]

There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```plaintext
Dim ret as Long

ret = pd.cpu_return
If ret = 0 Then
    pd.err_disp_message
End If
```
cpu_src_over

Description
This function over step executes the target program, one source line at a time including subroutines, beginning with the current PC.

Parameters
ret = pd.cpu_src_over()

There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

Dim ret as Long

ret = pd.cpu_src_over
If ret = 0 Then
    pd.err_disp_message
End If
cpu_src_step

Description
This function step executes the target program, one source line at a time, beginning with the current PC.

Parameters
\[
\text{ret} = \text{pd.cpu_src_step}() \\
\]
There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `errDisp_message`.

Description example

```vba
Dim ret as Long

ret = pd.cpu_src_step
If ret = 0 Then
    pd.errDisp_message
End If
```
cpu_step

Description
This function step executes the target program, one instruction at a time, beginning with the current PC.

Parameters

ret = pd.cpu_step()

There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

Dim ret as Long

ret = pd.cpu_step
If ret = 0 Then
  pd.err_disp_message
End If
**cpu_stop**

**Description**

This function stops the target program.

**Parameters**

```vba
ret = pd.cpu_stop()
```

There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>Error:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vba
Dim ret as Long

ret = pd.cpu_stop
If ret = 0 Then
    pd.err_disp_message
End If
```
cpu_wait

**Description**

This function waits for until the target program stops.

**Parameters**

```plaintext
ret = pd.cpu_wait()
```

There is no parameter.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```plaintext
Dim ret as Long

ret = pd.cpu_wait
If ret = 0 Then
    pd.errDisp_message
End If
```
com_receive

Description
This function receives a data from Emulator.

Parameters
ret = pd.com_receive ( data, size )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>BSTR *data</td>
<td>Receive data</td>
</tr>
<tr>
<td>[in]</td>
<td>long size</td>
<td>Receive size</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>Error:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```vba
Dim ret as Long
Dim data as String
Dim size as Long

data = "(07)"
size = 4
ret = pd.com_send ( data, size )
If ret = 0 Then
    pd.err_disp_message
End If
size = 3
ret = pd.com_receive ( data, size )
If ret = 0 Then
    pd.err_disp_message
End If
```
com_send

Description
This function sends a data to Emulator.

Parameters
\[
\text{ret} = \text{pd.com_send ( data, size )}
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR data</td>
<td>Send data</td>
</tr>
<tr>
<td>[in]</td>
<td>long size</td>
<td>Send size</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```vbnet
Dim ret as Long
Dim data as String
Dim size as Long

data = "(07)"
size = 4
ret = pd.com_send ( data, size )
If ret = 0 Then	pd.err_disp_message
End If
size = 3
ret = pd.com_receive ( data, size )
If ret = 0 Then	pd.err_disp_message
End If
```
cv_clear

Description
This function clears the coverage data.

Parameters
ret = pd.cv_clear()

There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

Description example

```
Dim ret as Long

ret = pd.cv_clear
If ret = 0 Then
    pd.err_disp_message
End If
```
cv_clear_blk

Description
This function clears the specified coverage block data.
(This method is supported to use the PC7501 emulator.)

Parameters

\[
\text{ret} = \text{pd.cv_clear_blk}(\text{blkno})
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long blkno</td>
<td>Coverage block number</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim blkno as Long
blkno = 0
ret = pd.cv_clear_blk(blkno)
If ret = 0 Then
    pd.err_disp_message
End If
```
**cv_get_base**

**Description**
This function gets the beginning address of the coverage area.

**Parameters**

```plaintext
ret = pd.cv_get_base ( base )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>long *base</td>
<td>Beginning address</td>
</tr>
</tbody>
</table>

The beginning address of the coverage area is stored in “base”. The coverage measurement area is the 256k bytes from the beginning address. When the beginning address is 80000h, the coverage area becomes from 80000h to AFFFFh.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**

```plaintext
' Dim ret as Long
Dim base as Long

ret = pd.cv_get_base ( base )
If ret = 0 Then
    pd.err_disp_message
End If
```
cv_get_base_all

Description
This function gets the beginning address of the all coverage area.
(This method is supported to use the PC7501 emulator.)

Parameters

```
ret = pd.cv_get_base_all ( base )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>VARIANT *base</td>
<td>Beginning address</td>
</tr>
</tbody>
</table>

The beginning address of the coverage area is stored in “base”. The coverage measurement area is the 256k bytes from the beginning address. When the biginning address is 80000h, the coverage area becomes from 80000h to AFFFFh.

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim base as Variant

ret = pd.cv_get_base ( base )
If ret = 0 Then
   pd.err_disp_message
End If
```
**cv_get_base_blk**

**Description**
This function gets the beginning address of the specified coverage block. (This method is supported to use the PC7501 emulator.)

**Parameters**

```
ret = pd.cv_get_base_blk ( blkno, base )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long blkno</td>
<td>Coverage block number</td>
</tr>
<tr>
<td>[out]</td>
<td>long *base</td>
<td>Beginning address</td>
</tr>
</tbody>
</table>

The beginning address of the coverage area is stored in “base”. The coverage measurement area is the 256k bytes from the beginning address. When the beginning address is 80000h, the coverage area becomes from 80000h to AFFFFh.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```
Dim ret as Long
Dim base as Long
Dim blkno as Long
blkno = 0
ret = pd.cv_get_base_blk ( blkno, base )
If ret = 0 Then
    pd.err_disp_message
End If
```
cv_get_blkcnt

**Description**
This function gets the coverage block count.
(This method is supported to use the PC7501 emulator.)

**Parameters**
```
ret = pd.cv_get_blkcnt ( cnt )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>long *cnt</td>
<td>Coverage block count</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**
```
Dim ret as Long
Dim cnt as Long

ret = pd.cv_get_blkcnt ( cnt )
If ret = 0 Then
  pd.err_disp_message
End If
```
cv_get_data

Description
This function gets the coverage data.

Parameters
ret = pd.cv_get_data ( st_addr, en_addr, rs_addr, re_addr, data )

This function stores the coverage data that includes an address range specified by “st_addr” and “en_addr” in the area specified by “data”.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long st_addr</td>
<td>Start address</td>
</tr>
<tr>
<td>[in]</td>
<td>long en_addr</td>
<td>End address</td>
</tr>
<tr>
<td>[out]</td>
<td>long *rs_addr</td>
<td>Real start address</td>
</tr>
<tr>
<td>[out]</td>
<td>long *re_addr</td>
<td>Real end address</td>
</tr>
<tr>
<td>[out]</td>
<td>VARIANT *data</td>
<td>Coverage data</td>
</tr>
</tbody>
</table>

However, since data for 8 bytes of addresses from each 8-byte alignment is stored in one byte of “data”, it can happen that a greater range of data than addresses specified by “st_addr” and “en_addr” actually is stored. For example, if addresses from 3h to 19h are specified, data at addresses from 0h to 1Fh actually are stored. The start and end addresses of the actually obtained data are stored in “rs_addr” and “re_addr”, respectively. Note that the values stored in “rs_addr” and “re_addr” can be obtained by calculation using the formula below.

rs_addr = st_addr / 8 * 8
re_addr = en_addr / 8 * 8 + 7

For “data”, specify an array greater than “en_addr” – “st_addr” / 8 + 1. The format of the coverage data stored in one byte of “data” is as follows:(Upper row: Bit offset; Lower row: address offset)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+7</td>
<td>+6</td>
<td>+5</td>
<td>+4</td>
<td>+3</td>
<td>+2</td>
<td>+1</td>
<td>+0</td>
</tr>
</tbody>
</table>

For example, if “st_addr” is 0x400, the coverage results at the addresses offset by the amount corresponding to each bit are stored in “data[0]” as shown below. (Upper row: Bit offset; Lower row: Address)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>407</td>
<td>406</td>
<td>405</td>
<td>404</td>
<td>403</td>
<td>402</td>
<td>401</td>
<td>400</td>
</tr>
</tbody>
</table>

Consequently, if memory is accessed every other byte beginning with “st_addr”, coverage data is stored as shown below.(Upper row: Bit offset; Lower row: Coverage measurement result)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The data stored in “data[0]” is 01010101B, i.e., 0x55.

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.
Description example

```vba
Dim ret as Long
Dim st_addr as Long
Dim en_addr as Long
Dim rs_addr as Long
Dim re_addr as Long
Dim data as Variant
Dim LoDataByte as Long
Dim HiDataByte as Long

st_addr = &hC000
en_addr = &hC007
ret = pd.cv_get_data ( st_addr, en_addr, rs_addr, re_addr, data )
If ret = 0 Then
    pd.err_disp_message
End If
LoDataByte = data(0) And &HF
Select Case LoDataByte Mod 16
    Case 0
        buff1 = " "
    Case 1
        buff1 = "* "
    Case 2
        buff1 = " * "
    Case 3
        buff1 = "** "
    Case 4
        buff1 = "* "
    Case 5
        buff1 = "* *"
    Case 6
        buff1 = "*  *
    Case 7
        buff1 = "***"
    Case 8
        buff1 = " **
    Case 9
        buff1 = "* *"
    Case 10
        buff1 = "* **
    Case 11
        buff1 = "*** *
    Case 12
        buff1 = " **
    Case 13
        buff1 = "* ***
    Case 14
        buff1 = "*** *
    Case 15
        buff1 = "*****
End Select
HiDataByte = (data(0) And &HF0) ¥ &H10
```

**cv_set_base**

**Description**
This function sets the beginning address of the coverage area.

**Parameters**

```
ret = pd.cv_set_base ( base )
```

The beginning address of the coverage measurement is specified for “base”.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long base</td>
<td>Beginning address</td>
</tr>
</tbody>
</table>

The coverage area is 256K consecutive bytes which starts from 64K bytes boundary. When the beginning address is C0000h, the coverage area becomes from C0000h to FFFFFh.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**

```
Dim ret as Long
Dim base as Long

base = &hC0000
ret = pd.cv_set_base ( base )
If ret = 0 Then
    pd.err_disp_message
End If
```
down_load

**Description**
This function downloads the target program.

**Parameters**

```vba
ret = pd.down_load ( filename, mode )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR filename</td>
<td>Download filename</td>
</tr>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Download mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 : (LOAD_LOAD) All Road module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 : (LOAD_SYM) Symbol information only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 : (LOAD_ROM) Machine data only</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vba
Const LOAD_LOAD = 4
Dim ret as Long
Dim filename as String

filename = "sample.x30"
ret = pd.down_load ( filename, LOAD_LOAD )
If ret = 0 Then
    pd.err_disp_message
End If
```
err_disp_message

Description
This function displays the PDxx error message.

Parameters
ret = pd.err_disp_message()
There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

Description example

```
Dim ret as Long
ret = pd.cpu_reset
If ret = 0 Then
    pd.err_disp_message
End If
```
err_get_message

Description
This function gets the PDxx error message.

Parameters
\[ \text{ret} = \text{pd.err_get_message ( msg )} \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>BSTR *msg</td>
<td>Error message</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

Description example

```vba
Dim ret as Long
Dim msg as String
ret = pd.cpu_reset
If ret = 0 Then
    ret = pd.err_get_message (msg)
    MsgBox msg
End If
```
**event_set_request_mode**

**Description**
This function sets the event request mode.

**Parameters**
```python
ret = pd.event_set_request_mode ( mode )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Request mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: (EVT_DISABLE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: (EVT_ENABLE)</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```python
Const EVT_DISABLE = 0
Const EVT_ENABLE = 1
Dim ret as Long

ret = pd.event_request_mode( EVT_DISABLE)
If ret = 0 Then
    pd.err_disp_message
End If
```
exp_eval

Description
This function gets the value that corresponds to a string character symbol.

Parameters
ret = pd.exp_eval ( str, radix, mode, data )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR str</td>
<td>Character Symbol</td>
</tr>
<tr>
<td>[in]</td>
<td>long radix</td>
<td>Effective radix</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 : (EXP_HEX)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 : (EXP_DEX)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 : (EXP_DEFAULT)</td>
</tr>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Search priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 : (EXP_LABEL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 : (EXP_SYMBOL)</td>
</tr>
<tr>
<td>[out]</td>
<td>long *data</td>
<td>Correspond data</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1               |
| Error:     | 0               |

The error message can be displayed by calling method err_disp_message.

Description example

```
Const EXP_LABEL = 0
Const EXP_HEX = 16
Dim ret as Long
Dim str as String
Dim data as Long

ret = pd.exp_eval ( str, EXP_HEX, EXP_LABEL, data )
If ret = 0 Then
    pd.err_disp_message
End If
```
info_cpu

Description
This function gets the information of target MCU.

Parameters
pd.info_cpu ( flag, status )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long flag</td>
<td>Information number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of bytes required for string address value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum value of address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of digits with which address values are displayed in hexadecimal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Endian of the target CPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length in bytes of word</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum length in bytes of one instruction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum value of data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum value of stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data bus width</td>
</tr>
<tr>
<td>[out]</td>
<td>long *status</td>
<td>Value corresponding to flag</td>
</tr>
</tbody>
</table>

Returned value
There is no parameter.

Description example

```vbnet
Const IN1_MAXADDR = 263
Dim ret as Long
Dim maxAddr as Long

flag = IN1_MAXADDR
pd.info_cpu ( flag, maxAddr )
lblMaxAddr = maxAddr
```
info_get_debugger

**Description**
Get the debugger information such as a product name.

**Parameters**
ret = pd.info_get_debugger ( name, dir, ini )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BSTR *name</td>
<td>Debugger name + version</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *dir</td>
<td>Directory of PDxx.exe</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *ini</td>
<td>Ini filename</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**
```
Dim ret As Long
Dim name as String
Dim dir As String
Dim ini As String

ret = pd.info_get_debugger(name, dir, ini)
If ret = 0 Then
    pd.err_disp_message
End If
```
info_get_mcufile_name

Description
Get the MCU file name selected in the INIT dialog box.

Parameters
ret = pd.info_get_mcufile_name ( name )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>BSTR *name</td>
<td>MCU filename</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example
Dim ret As Long
Dim name as String

    ret = pd.info_get_mcufile_name(name)
    If ret = 0 Then
        pd.err_disp_message
    End If
info_service

Description
This function gets the supported information of PDxx.

Parameters
pd.info_service ( flag, status )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long flag</td>
<td>Information number</td>
</tr>
<tr>
<td>275</td>
<td>(IN1_SUPPORT_BITSYM)</td>
<td>Bit symbol</td>
</tr>
<tr>
<td>276</td>
<td>(IN1_SUPPORT_C)</td>
<td>C-language debugging</td>
</tr>
<tr>
<td>277</td>
<td>(IN1_SUPPORT_RAMMONITOR)</td>
<td>Real-time RAM monitor</td>
</tr>
<tr>
<td>278</td>
<td>(IN1_SUPPORT_RTT)</td>
<td>Real-time trace</td>
</tr>
<tr>
<td>279</td>
<td>(IN1_SUPPORT_CV)</td>
<td>Coverage</td>
</tr>
<tr>
<td>280</td>
<td>(IN1_SUPPORT_PROTCT)</td>
<td>Protect break</td>
</tr>
<tr>
<td>[out]</td>
<td>long *status</td>
<td>Value corresponding to flag</td>
</tr>
</tbody>
</table>

 Returned value
There is no parameter.

Description example

Const IN1_SUPPORT_RAMMONITOR = 277
Dim ret as Long
Dim status as Long

flag = IN1_SUPPORT_RAMMONITOR
pd.info_service ( flag, status )
If status = 1 Then
    MsgBox(“Support RAM Monitor.”)
End If
mem_clear_bit

Description
This function clears a bit data.

Parameters
ret = pd.mem_clear_bit ( addr, bit )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long bit</td>
<td>Bit number</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err Disp_message.

Description example

```
Dim ret as Long
Dim addr as Long
Dim bit as Long

addr = &h400
length = 1
ret = pd.mem_clear_bit ( addr, bit )
If ret = 0 Then
    pd.err_disp_message
End If
```
**mem_get**

**Description**
This function gets a memory data. (1 data)

**Parameters**
ret = pd.mem_get ( addr, length, data )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long length</td>
<td>Data length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: 2 bytes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: 4 bytes*</td>
</tr>
<tr>
<td>[out]</td>
<td>long *data</td>
<td>Data</td>
</tr>
</tbody>
</table>

* Reservation parameter

**Returned value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded:</td>
<td>1</td>
</tr>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**

```vbscript
Dim ret as Long
Dim addr as Long
Dim length as Long
Dim data as Long

addr = &h400
length = 1
ret = pd.mem_get ( addr, length, data )
If ret = 0 Then
    pd.err_disp_message
End If
```
**mem_get_bit**

**Description**
This function gets a bit data.

**Parameters**

```
ret = pd.mem_get_bit ( addr, bit, data )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long bit</td>
<td>Bit number</td>
</tr>
<tr>
<td>[out]</td>
<td>long *data</td>
<td>Bit value</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

**Description example**

```
Dim ret as Long
Dim addr as Long
Dim bit as Long
Dim data as Long

addr = &h400
bit = 1
ret = pd.mem_get_bit ( addr, bit, data )
If ret = 0 Then
    pd.err_disp_message
End If
```
mem_get_multi

Description
This function gets a memory data. (multi data)

Parameters
ret = pd.mem_get_multi ( addr, length, size, data )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>start address</td>
</tr>
<tr>
<td>[in]</td>
<td>long num</td>
<td>get size</td>
</tr>
<tr>
<td>[in]</td>
<td>long length</td>
<td>data length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: 2 bytes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: 4 bytes*</td>
</tr>
<tr>
<td>[out]</td>
<td>VARIANT *data</td>
<td>Saving data</td>
</tr>
</tbody>
</table>

* Reservation parameter

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example
```
Dim ret as Long
Dim addr as Long
Dim num as Long
Dim length as Long
Dim data as Variant
Dim a as Long

addr = &h400
num = 4
length = 1
ret = pd.mem_get_multi ( addr, num, length, data )
If ret = 0 Then
    pd.err_disp_message
End If
a = data(0)
```
mem_fill

Description
This function fills the memory data.

Parameters
\[ \text{ret} = \text{pd.mem_fill}(\text{addr, length, size, data}) \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long num</td>
<td>Address size</td>
</tr>
<tr>
<td>[in]</td>
<td>long length</td>
<td>Data Length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: 2 bytes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: 4 bytes*</td>
</tr>
<tr>
<td>[in]</td>
<td>long data</td>
<td>Fill data</td>
</tr>
</tbody>
</table>

* Reservation parameter

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method \text{err_disp_message}.

Description example

```plaintext
Dim ret as Long
Dim addr as Long
Dim num as Long
Dim length as Long
Dim data as Variant

addr = &h400
num = 4
length = 1
data = &h10
ret = pd.mem_fill(addr, num, length, data)
If ret = 0 Then
    pd.err_disp_message
End If
```
mem_move

Description
This function transfers the memory data.

Parameters

\[
\text{ret} = \text{pd.mem_move}(\text{addr, length, top, size})
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long num</td>
<td>Address size</td>
</tr>
<tr>
<td>[in]</td>
<td>long top</td>
<td>Transfer address</td>
</tr>
<tr>
<td>[in]</td>
<td>long length</td>
<td>Data length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: 2 bytes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: 4 bytes*</td>
</tr>
</tbody>
</table>

* Reservation parameter

Returned value

| Succeeded: | 1 |
| Error:      | 0 |

The error message can be displayed by calling method `err_disp_message`.

Description example

```vbscript
Dim ret as Long
Dim addr as Long
Dim num as Long
Dim top as Long
Dim length as Long

addr = &h400
num = 4
top = &h500
length = 1
ret = pd.mem_move(addr, num, top, length)
If ret = 0 Then
    pd.err_disp_message
End If
```
**Description**
This function sets a memory data. (1 data)

**Parameters**
ret = pd.mem_set( addr, length, data )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long length</td>
<td>Data length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: 2 bytes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: 4 bytes*</td>
</tr>
<tr>
<td>[in]</td>
<td>long data</td>
<td>Set data</td>
</tr>
</tbody>
</table>

* Reservation parameter

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

**Description example**

```
Dim ret as Long
Dim addr as Long
Dim length as Long
Dim data as Long

addr = &h400
length = 1
data = &h12
ret = pd.mem_set( addr, length, data )
If ret = 0 Then
    pd.err_disp_message
End If
```
mem_set_bit

Description
This function sets a bit data.

Parameters
ret = pd.mem_set_bit ( addr, bit )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long bit</td>
<td>Bit number</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim addr as Long
Dim bit as Long

addr = &h400
length = 1
ret = pd.mem_set_bit ( addr, bit )
If ret = 0 Then
    pd.err_disp_message
End If
```
mem_set_multi

Description
This function sets memory data. (multi data)

Parameters
ret = pd.mem_set_multi ( addr, length, size, data )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long num</td>
<td>Address size</td>
</tr>
<tr>
<td>[in]</td>
<td>long length</td>
<td>Data length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: 2 bytes*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: 4 bytes*</td>
</tr>
<tr>
<td>[in]</td>
<td>VARIANT data</td>
<td>Reservation parameter</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim addr as Long
Dim num as Long
Dim length as Long
Dim data as Variant

data(0) = 1
data(1) = 2
data(2) = 3
data(3) = 4
addr = &h400
num = 4
length = 1
ret = pd.mem_set_multi ( addr, num, length, data )
If ret = 0 Then
    pd.err_disp_message
End If
```
rram_clear

Description
This function initializes access states of the RAM monitor area.

Parameters
ret = pd.rram_clear()

There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
ret = pd.rram_clear
If ret = 0 Then
    pd.err_disp_message
End If
```
**rram_clear_blk**

**Description**
This function initializes access states of the specified RAM monitor block.
(This method is supported to use the PC7501 emulator.)

**Parameters**

```plaintext
ret = pd.rram_clear_blk ( blkno )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long blkno</td>
<td>RAM monitor block number</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```plaintext
ret = pd.rram_clear_blk ( blkno )
If ret = 0 Then
    pd.err_disp_message
End If
```
**rram_get_area**

**Description**
This function gets the beginning address of the RAM monitor area.

**Parameters**

```vbnet
ret = pd.rram_get_area ( addr )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>long *addr</td>
<td>Top address of RAM monitor</td>
</tr>
</tbody>
</table>

The beginning address of the RAM monitor area is stored in “addr”. The RAM monitor area is the 1k bytes from the beginning address. When the beginning address is 400h, the RAM monitor area becomes from 400h to 7FFh.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**

```vbnet
Dim ret as Long
Dim addr as Long
ret = pd.rram_get_area ( addr )
If ret = 0 Then
    pd.err_disp_message
End If
```
rram_get_area_blk

**Description**
This function gets the beginning address of the specified RAM monitor block. (This method is supported to use the PC7501 emulator.)

**Parameters**

```plaintext
ret = pd.rram_get_area ( blkno, addr )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long blkno</td>
<td>RAM monitor block number</td>
</tr>
<tr>
<td>[out]</td>
<td>long *addr</td>
<td>Top address of RAM monitor</td>
</tr>
</tbody>
</table>

The beginning address of the RAM monitor area is stored in “addr”. The RAM monitor area is the 1k bytes from the beginning address. When the beginning address is 400h, the RAM monitor area becomes from 400h to 7FFh.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```plaintext
Dim ret as Long
Dim addr as Long
Dim blkno as Long

blkno = 1
ret = pd.rram_get_area_blk ( blkno, addr )
If ret = 0 Then
    pd.err_disp_message
End If
```
**rram_get_data**

**Description**
This function gets the RAM monitor data and the access attributes.

**Parameters**
\[
\text{ret} = \text{pd.rram_get_data ( addr, num, data, attr )}
\]

The address outside the RAM monitor area cannot be specified.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Start Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long num</td>
<td>Obtain data size</td>
</tr>
<tr>
<td>[out]</td>
<td>VARIANT *data</td>
<td>RAM monitor data</td>
</tr>
<tr>
<td>[out]</td>
<td>VARIANT *attr</td>
<td>RAM monitor access attributes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: No access</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: Read</td>
</tr>
</tbody>
</table>

**Returned value**

Succeeded: 1
Error: 0
The error message can be displayed by calling method err_disp_message.

**Description example**

```vba
Dim ret as Long
Dim addr as Long
Dim num as Long
Dim data as Variant
Dim attr as Variant
Dim aaa as Long
Dim bbb as Long

addr = &h400
num = 4
ret = pd.rram_get_data ( addr, num, data, attr )
If ret = 0 Then
    pd.err_disp_message
End If
aaa = data(0)
bbb = attr(0)
```
rram_set_area

Description
This function sets the beginning address of the RAM monitor area.

Parameters
ret = pd.rram_get_area ( addr )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Top address of RAM monitor</td>
</tr>
</tbody>
</table>

The RAM monitor area is 1K consecutive bytes which starts from 16 bytes boundary. When the beginning address is 408h, the coverage area becomes from 400h to 7FFh.

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim addr as Long

addr = &h400
ret = pd.rram_set_area ( addr )
If ret = 0 Then
    pd.err_disp_message
End If
```
rram_set_area_blk

Description
This function sets the beginning address of the specified RAM monitor block.
(This method is supported to use the PC7501 emulator.)

Parameters
ret = pd.rram_get_area_blk(blkno, addr)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long blkno</td>
<td>RAM monitor block number</td>
</tr>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Top address of RAM monitor</td>
</tr>
</tbody>
</table>

The RAM monitor area is 1K consecutive bytes which starts from 16 bytes boundary. When the beginning address is 408h, the coverage area becomes from 400h to 7FFh.

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim addr as Long
Dim blkno as Long

blkno = 0
addr = &h400
ret = pd.rram_set_area_blk(blkno, addr)
If ret = 0 Then
    pd.err_disp_message
End If
```
**reg_get_pc**

**Description**
This function gets the value of program counter.

**Parameters**

```plaintext
ret = pd.reg_get_pc ( pc )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>long *pc</td>
<td>Program counter value</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**

```plaintext
Dim ret as Long
Dim pc as Long

ret = pd.reg_get_pc ( pc )
If ret = 0 Then
    pd.err_disp_message
End If
```

78
**reg_get_reg**

**Description**
This function gets the value of specified register.

**Parameters**

\[
\text{ret} = \text{pd.reg_get_reg} \left( \text{regNo, regVal} \right)
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long regNo</td>
<td>Register number</td>
</tr>
<tr>
<td>[out]</td>
<td>long *regVal</td>
<td>Register value</td>
</tr>
</tbody>
</table>

A register number changes with debugger.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PD32R/</td>
<td>PD32RSIM</td>
</tr>
<tr>
<td>PD308/</td>
<td>PD308SIM</td>
</tr>
<tr>
<td>PD30/</td>
<td>PD30SIM</td>
</tr>
<tr>
<td>PD79/</td>
<td>PD79SIM</td>
</tr>
<tr>
<td>PD38/</td>
<td>PD38SIM</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0:R0</td>
<td>0:R0</td>
</tr>
<tr>
<td>1:R1</td>
<td>1:R1</td>
</tr>
<tr>
<td>2:R2</td>
<td>2:R2</td>
</tr>
<tr>
<td>3:R3</td>
<td>3:R3</td>
</tr>
<tr>
<td>4:R4</td>
<td>4:R4</td>
</tr>
<tr>
<td>5:R5</td>
<td>5:R5</td>
</tr>
<tr>
<td>6:R6</td>
<td>6:R6</td>
</tr>
<tr>
<td>7:R7</td>
<td>7:R7</td>
</tr>
<tr>
<td>8:R8</td>
<td>8:R8</td>
</tr>
<tr>
<td>9:R9</td>
<td>9:R9</td>
</tr>
<tr>
<td>10:R10</td>
<td>10:R10</td>
</tr>
<tr>
<td>11:R11</td>
<td>11:R11</td>
</tr>
<tr>
<td>12:R12</td>
<td>12:R12</td>
</tr>
<tr>
<td>13:R13</td>
<td>13:R13</td>
</tr>
<tr>
<td>14:R14</td>
<td>14:R14</td>
</tr>
<tr>
<td>15:R15</td>
<td>15:R15</td>
</tr>
<tr>
<td>16:PSW</td>
<td>16:PSW</td>
</tr>
<tr>
<td>17:SPI</td>
<td>17:SPI</td>
</tr>
<tr>
<td>18:SPU</td>
<td>18:SPU</td>
</tr>
<tr>
<td>19:BPC</td>
<td>19:BPC</td>
</tr>
<tr>
<td>20:PC</td>
<td>20:PC</td>
</tr>
<tr>
<td>21:ACC0H</td>
<td>21:ACC0H</td>
</tr>
<tr>
<td>22:ACC0L</td>
<td>22:ACC0L</td>
</tr>
<tr>
<td>23:ACC1H</td>
<td>23:ACC1H</td>
</tr>
<tr>
<td>24:ACC1L</td>
<td>24:ACC1L</td>
</tr>
<tr>
<td>0:R0</td>
<td>0:R0</td>
</tr>
<tr>
<td>1:R1</td>
<td>1:R1</td>
</tr>
<tr>
<td>2:R2</td>
<td>2:R2</td>
</tr>
<tr>
<td>3:R3</td>
<td>3:R3</td>
</tr>
<tr>
<td>4:R4</td>
<td>4:R4</td>
</tr>
<tr>
<td>5:R5</td>
<td>5:R5</td>
</tr>
<tr>
<td>6:R6</td>
<td>6:R6</td>
</tr>
<tr>
<td>7:R7</td>
<td>7:R7</td>
</tr>
<tr>
<td>8:R8</td>
<td>8:R8</td>
</tr>
<tr>
<td>9:R9</td>
<td>9:R9</td>
</tr>
<tr>
<td>10:R10</td>
<td>10:R10</td>
</tr>
<tr>
<td>11:R11</td>
<td>11:R11</td>
</tr>
<tr>
<td>12:R12</td>
<td>12:R12</td>
</tr>
<tr>
<td>13:R13</td>
<td>13:R13</td>
</tr>
<tr>
<td>14:R14</td>
<td>14:R14</td>
</tr>
<tr>
<td>15:R15</td>
<td>15:R15</td>
</tr>
<tr>
<td>16:PSW</td>
<td>16:PSW</td>
</tr>
<tr>
<td>17:SPI</td>
<td>17:SPI</td>
</tr>
<tr>
<td>18:SPU</td>
<td>18:SPU</td>
</tr>
<tr>
<td>19:BPC</td>
<td>19:BPC</td>
</tr>
<tr>
<td>20:PC</td>
<td>20:PC</td>
</tr>
<tr>
<td>21:ACC0H</td>
<td>21:ACC0H</td>
</tr>
<tr>
<td>22:ACC0L</td>
<td>22:ACC0L</td>
</tr>
<tr>
<td>23:ACC1H</td>
<td>23:ACC1H</td>
</tr>
<tr>
<td>24:ACC1L</td>
<td>24:ACC1L</td>
</tr>
<tr>
<td>0:R0</td>
<td>0:R0</td>
</tr>
<tr>
<td>1:R1</td>
<td>1:R1</td>
</tr>
<tr>
<td>2:R2</td>
<td>2:R2</td>
</tr>
<tr>
<td>3:R3</td>
<td>3:R3</td>
</tr>
<tr>
<td>4:R4</td>
<td>4:R4</td>
</tr>
<tr>
<td>5:R5</td>
<td>5:R5</td>
</tr>
<tr>
<td>6:R6</td>
<td>6:R6</td>
</tr>
<tr>
<td>7:R7</td>
<td>7:R7</td>
</tr>
<tr>
<td>8:R8</td>
<td>8:R8</td>
</tr>
<tr>
<td>9:R9</td>
<td>9:R9</td>
</tr>
<tr>
<td>10:R10</td>
<td>10:R10</td>
</tr>
<tr>
<td>11:R11</td>
<td>11:R11</td>
</tr>
<tr>
<td>12:R12</td>
<td>12:R12</td>
</tr>
<tr>
<td>13:R13</td>
<td>13:R13</td>
</tr>
<tr>
<td>14:R14</td>
<td>14:R14</td>
</tr>
<tr>
<td>15:R15</td>
<td>15:R15</td>
</tr>
<tr>
<td>16:PSW</td>
<td>16:PSW</td>
</tr>
<tr>
<td>17:SPI</td>
<td>17:SPI</td>
</tr>
<tr>
<td>18:SPU</td>
<td>18:SPU</td>
</tr>
<tr>
<td>19:BPC</td>
<td>19:BPC</td>
</tr>
<tr>
<td>20:PC</td>
<td>20:PC</td>
</tr>
<tr>
<td>21:ACC0H</td>
<td>21:ACC0H</td>
</tr>
<tr>
<td>22:ACC0L</td>
<td>22:ACC0L</td>
</tr>
<tr>
<td>23:ACC1H</td>
<td>23:ACC1H</td>
</tr>
<tr>
<td>24:ACC1L</td>
<td>24:ACC1L</td>
</tr>
<tr>
<td>0:R0</td>
<td>0:R0</td>
</tr>
<tr>
<td>1:R1</td>
<td>1:R1</td>
</tr>
<tr>
<td>2:R2</td>
<td>2:R2</td>
</tr>
<tr>
<td>3:R3</td>
<td>3:R3</td>
</tr>
<tr>
<td>4:R4</td>
<td>4:R4</td>
</tr>
<tr>
<td>5:R5</td>
<td>5:R5</td>
</tr>
<tr>
<td>6:R6</td>
<td>6:R6</td>
</tr>
<tr>
<td>7:R7</td>
<td>7:R7</td>
</tr>
<tr>
<td>8:R8</td>
<td>8:R8</td>
</tr>
<tr>
<td>9:R9</td>
<td>9:R9</td>
</tr>
<tr>
<td>10:R10</td>
<td>10:R10</td>
</tr>
<tr>
<td>11:R11</td>
<td>11:R11</td>
</tr>
<tr>
<td>12:R12</td>
<td>12:R12</td>
</tr>
<tr>
<td>13:R13</td>
<td>13:R13</td>
</tr>
<tr>
<td>14:R14</td>
<td>14:R14</td>
</tr>
<tr>
<td>15:R15</td>
<td>15:R15</td>
</tr>
<tr>
<td>16:PSW</td>
<td>16:PSW</td>
</tr>
<tr>
<td>17:SPI</td>
<td>17:SPI</td>
</tr>
<tr>
<td>18:SPU</td>
<td>18:SPU</td>
</tr>
<tr>
<td>19:BPC</td>
<td>19:BPC</td>
</tr>
<tr>
<td>20:PC</td>
<td>20:PC</td>
</tr>
<tr>
<td>21:ACC0H</td>
<td>21:ACC0H</td>
</tr>
<tr>
<td>22:ACC0L</td>
<td>22:ACC0L</td>
</tr>
<tr>
<td>23:ACC1H</td>
<td>23:ACC1H</td>
</tr>
<tr>
<td>24:ACC1L</td>
<td>24:ACC1L</td>
</tr>
<tr>
<td>0:A</td>
<td>0:A</td>
</tr>
<tr>
<td>1:B</td>
<td>1:B</td>
</tr>
<tr>
<td>2:X</td>
<td>2:X</td>
</tr>
<tr>
<td>3:Y</td>
<td>3:Y</td>
</tr>
<tr>
<td>4:S</td>
<td>4:S</td>
</tr>
<tr>
<td>5:PC</td>
<td>5:PC</td>
</tr>
<tr>
<td>6:DT</td>
<td>6:DT</td>
</tr>
<tr>
<td>7:PS</td>
<td>7:PS</td>
</tr>
<tr>
<td>8:PG</td>
<td>8:PG</td>
</tr>
<tr>
<td>9:DP0</td>
<td>9:DP0</td>
</tr>
<tr>
<td>10:DP1</td>
<td>10:DP1</td>
</tr>
<tr>
<td>11:DP2</td>
<td>11:DP2</td>
</tr>
<tr>
<td>12:DP3</td>
<td>12:DP3</td>
</tr>
<tr>
<td>13:E</td>
<td>13:E</td>
</tr>
<tr>
<td>14:PGPC</td>
<td>14:PGPC</td>
</tr>
</tbody>
</table>

**Returned value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded:</td>
<td>1</td>
</tr>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
Dim ret as Long
Dim regNo as Long
Dim regVal as Long

regNo = 0
ret = pd.reg_get_reg ( regNo, regVal )
If ret = 0 Then
    pd.err_disp_message
End If
```
**PDSDK - COM Kit**

**reg_set_pc**

**Description**
This function sets a program counter value.

**Parameters**

```
ret = pd.reg_set_pc ( pc )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long pc</td>
<td>Program counter value</td>
</tr>
</tbody>
</table>

**Returned value**

- Succeeded: 1
- Error: 0

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```
Dim ret as Long
Dim pc as Long

pc = &hF0000
ret = pd.reg_set_pc ( pc )
If ret = 0 Then
    pd.err_disp_message
End If
```
**reg_set_reg**

**Description**
This function sets the value of specified register.

**Parameters**

\[ ret = pd.reg_set_reg \left( \text{regNo, regVal} \right) \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long regNo</td>
<td>Register number</td>
</tr>
<tr>
<td>[in]</td>
<td>long regVal</td>
<td>Register value</td>
</tr>
</tbody>
</table>

**Returned value**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded:</td>
<td>1</td>
</tr>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
' Dim ret as Long
Dim regNo as Long
Dim regVal as Long

regNo = 0
regVal = 3
ret = pd.reg_set_reg ( regNo, regVal )
If ret = 0 Then
    pd.err_disp_message
End If
```
rtt_check_isfetch

Description
This function checks fetch cycle of specified real-time trace cycle.

Parameters

\[
\text{ret} = \text{pd.rtt_check_isfetch}( \text{cycle}, \text{addr1}, \text{addr2}, \text{count} )
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long cycle</td>
<td>Cycle</td>
</tr>
<tr>
<td>[out]</td>
<td>long *addr1</td>
<td>1st fetch cycle</td>
</tr>
<tr>
<td>[out]</td>
<td>long *addr2</td>
<td>2nd fetch cycle</td>
</tr>
<tr>
<td>[out]</td>
<td>long *count</td>
<td>Fetch mnemonic count</td>
</tr>
</tbody>
</table>

When the specified cycle is a fetch cycle, the count of fetch mnemonic is stored in "count", the fetch address of the first point is stored in "addr1", and fetch address of the second point is stored in "addr2". When the specified cycle is not a fetch cycle, 0 is stored in "count".

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```vbs
Dim ret as Long
Dim cycle as Long
Dim addr1 as Long
Dim addr2 as Long
Dim count as Long

cycle = -10
ret = pd.rtt_check_isfetch( cycle, addr1, addr2, count )
If ret = 0 Then
    pd.err_disp_message
End If
If count <> 0 Then
    label.Caption = "Fetch cycle."
End If
```
rtt_clear

Description
This function initializes the trace data.

Parameters
ret = pd.rtt_clear()

There is no parameter.

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```vba
Dim ret as Long

ret = pd.rtt_clear
If ret = 0 Then
    pd.err_disp_message
End If
```
rtt_get_bus

Description
This function gets the address and BUS character string of specified real-time trace cycle.

Parameters
ret = pd.rtt_get_bus ( cycle, addr, buffer )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long cycle</td>
<td>Cycle</td>
</tr>
<tr>
<td>[out]</td>
<td>long *addr</td>
<td>Address</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *buffer</td>
<td>BUS character string</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

Dim ret as Long
Dim cycle as Long
Dim addr as Long
Dim buffer as String

cycle = -10
ret = pd.rtt_get_bus ( cycle, addr, buffer )
If ret = 0 Then
    pd.err_disp_message
End If
label.Caption = buffer
rtt_get_disasm

Description
This function gets the disassemble character string of specified real-time trace cycle.

Parameters
ret = pd.rtt_get_disasm ( cycle, next_cycle, buffer, count )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in,out]</td>
<td>long *cycle</td>
<td>Cycle</td>
</tr>
<tr>
<td>[out]</td>
<td>long *next_cycle</td>
<td>Next fetch cycle</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *buffer</td>
<td>Disassemble character string</td>
</tr>
<tr>
<td>[out]</td>
<td>long *count</td>
<td>Disassemble mnemonic count</td>
</tr>
</tbody>
</table>

The fetch cycle is stored in "cycle", the fetch cycle of the next fetch cycle is stored in "next_cycle", the disassemble character string is stored in "buffer", and the count of disassemble mnemonic is stored in "count".

Returned value

| Succeeded:  | 1 |
| Error:      | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```vbnet
Dim ret as Long
Dim cycle as Long
Dim next_cycle as Long
Dim buffer as String
Dim count as Long

cycle = -10
ret = pd.rtt_get_disasm ( cycle, next_cycle, buffer, count )
If ret = 0 Then
    pd.errDispMessage
End If
label.Caption = buffer
```
**rtt_get_range**

**Description**
This function gets the trace range of trace area.

**Parameters**

```plaintext
ret = pd.rtt_get_range ( s_cycle, e_cycle )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>long *s_cycle</td>
<td>Start cycle</td>
</tr>
<tr>
<td></td>
<td>long *e_cycle</td>
<td>End cycle</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>Error:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```plaintext
Dim ret as Long
Dim s_cycle as Long
Dim e_cycle as Long

ret = pd.rtt_get_range ( s_cycle, e_cycle )
If ret = 0 Then
    pd.err_disp_message
End If
```
**scri_command**

**Description**

This function executes the script command.

**Parameters**

```plaintext
ret = pd.scri_command ( string )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR string</td>
<td>Script command string</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:      | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```plaintext
Dim ret as Long
Dim buff as String

buff = "DumpByte 00000,00FFF"
ret = pd.scri_command( buff )
If ret = 0 Then
    pd.err_disp_message
End If
```
scri_print

Description
This function displays the specified string to Script Window.

Parameters
\[ \text{ret} = \text{pd.scri_print} \left( \text{string} \right) \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR string</td>
<td>Display string</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

Description example
```vbnet
dim ret as long
dim buff as string

buff = "2002/05/31"
ret = pd.scri_print( buff )
if ret = 0 then
    pd.err_disp_message
end if
```
**sym_add_bitsymbol**

**Description**
This function enters the bit symbol as a global bit symbol.

**Parameters**
ret = pd.sym_add_bitsymbol ( name, addr, bit )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR name</td>
<td>Enter bit symbol string</td>
</tr>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long bit</td>
<td>Bit number</td>
</tr>
</tbody>
</table>

When the global bit symbol of this name already exists, it is an error.

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**
```
Dim ret as Long
Dim name as String
Dim addr as Long
Dim bit as Long

name = "bitsym1"
addr = &h401
bit = 0
ret = pd.sym_add_bitsymbol ( name, addr, bit )
If ret = 0 Then
    pd.err_disp_message
End If
```
sym_add_label

**Description**
This function enters the label as a global label.

**Parameters**

```vbnet
dim ret as long
dim name as string
	name = &hF0001

type as long

type = "label1"

ret = pd.sym_add_label ( name, value )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR name</td>
<td>Enter label string</td>
</tr>
<tr>
<td>[in]</td>
<td>long value</td>
<td>Address</td>
</tr>
</tbody>
</table>

When the global label of this name already exists, it is an error.

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
Dim ret as Long
Dim name as String

name = &hF0001

val = "label1"

ret = pd.sym_add_label ( name, value )
If ret = 0 Then
    pd.err_disp_message
End If
```
**sym_add_symbol**

**Description**
This function enters the symbol as a global symbol.

**Parameters**

\[
\text{ret} = \text{pd.sym_add_symbol ( name, value )}
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR name</td>
<td>Enter symbol string</td>
</tr>
<tr>
<td>[in]</td>
<td>long value</td>
<td>Value</td>
</tr>
</tbody>
</table>

When the global symbol of this name already exists, it is an error.

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
Dim ret as Long
Dim name as String
Dim value as Long

name = &h401
value = "sym1"
ret = pd.sym_add_symbol ( name, value )
If ret = 0 Then
    pd.err_disp_message
End If
```
**sym_addr2line**

**Description**
This function gets the source file name and the line number corresponding to the specified address.

**Parameters**

```vbnet
ret = pd.sym_addr2line ( addr, line, filename )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[out]</td>
<td>long *line</td>
<td>Source line number</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *filename</td>
<td>Source file name</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
Dim ret as Long
Dim addr as Long
Dim line as Long
Dim filename as String

addr = &hF0000
ret = pd.sym_addr2line ( addr, line, filename )
If ret = 0 Then
    pd.err_disp_message
End If
```
**sym_bit2val**

**Description**
This function gets the address and the bit number corresponding to the specified bit symbol.

**Parameters**

\[
\text{ret} = \text{pd.sym_bit2val}(\text{symbol, addr, bit})
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR symbol</td>
<td>Bit Symbol String</td>
</tr>
<tr>
<td>[out]</td>
<td>long *addr</td>
<td>Address</td>
</tr>
<tr>
<td>[out]</td>
<td>long *bit</td>
<td>Bit number</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```plaintext```
Dim ret as Long
Dim symbol as String
Dim addr as Long
Dim bit as Long

symbol = "bitsym1"
ret = pd.sym_bit2val (symbol, addr, bit)
If ret = 0 Then
    pd.err_disp_message
End If
```
```plaintext```
sym_get_disp_src

**Description**
This function gets the displaying source file name in the program window.

**Parameters**

```vbscript
call ret = pd.sym_get_disp_src( filename )
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>BSTR *filename</td>
<td>Source file name</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbscript
dim ret as Long
Dim filename as String

ret = pd.sym_get_disp_src( filename )
If ret = 0 Then
    pd.err_disp_message
End If
```
**sym_get_func_info**

**Description**
Get the information of a specified function.

**Parameters**

\[\text{ret} = \text{pd.sym_get_func_info} \left( \text{func}, \text{type}, \text{start}, \text{end}, \text{str} \right)\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR func</td>
<td>Function name</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *type</td>
<td>Function type</td>
</tr>
<tr>
<td>[out]</td>
<td>long *start</td>
<td>Start address</td>
</tr>
<tr>
<td>[out]</td>
<td>long *end</td>
<td>End address</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *str</td>
<td>Address range string</td>
</tr>
</tbody>
</table>

**Returned value**

<table>
<thead>
<tr>
<th></th>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

**Description example**

```vbs```
Dim ret As Long
Dim func as String
Dim data As Long
Dim type As String
Dim sAddr As Long
Dim eAddr As Long
Dim str As String

func="main"
ret = pd.sym_get_func_info(func, type, sAddr, eAddr, str)
If ret = 0 Then
    pd.err_disp_message
End If```
```
sym_get_func_name

Description
Get function names from an object.

Parameters
\[ \text{ret} = \text{pd.sym_get_func_name}(\text{obj, func, mode}) \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR obj</td>
<td>Object name</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *func</td>
<td>Function name</td>
</tr>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Search mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: (LOAD_FIRST)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: (LOAD_NEXT)</td>
</tr>
</tbody>
</table>

An object name is not omissible.

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Const LOAD_FIRST = 2
Const LOAD_NEXT = 3
Dim ret As Long
Dim obj As String
Dim func As String
Dim sw As Long

sw = LOAD_FIRST
obj = “main”

Do
    ret = pd.sym_get_func_name(obj, func, sw)
    If ret = 0 Then
        Exit Do
    Else
        MsgBox func
        sw = LOAD_NEXT
    End If
Loop
```
**sym_get_obj_name**

**Description**
Get object names from a load module.

**Parameters**

\[
\text{ret} = \text{pd.sym_get_obj_name}(\text{obj, mode})
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>BSTR *obj</td>
<td>Object name</td>
</tr>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Search mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 : (LOAD_FIRST) First</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 : (LOAD_NEXT) Second and following object name</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbs
Const LOAD_FIRST = 2
Const LOAD_NEXT = 3
Dim ret As Long
Dim obj As String
Dim sw As Long

sw = LOAD_FIRST
Do
    ret = pd.sym_get_obj_name(obj, sw)
    If ret = 0 Then
        Exit Do
    Else
        MsgBox obj
        sw = LOAD_NEXT
    End If
Loop
```
sym_get_scope

Description
This function gets the scope (object file name) of specified address.

Parameters

ret = pd.sym_get_scope ( addr, filename )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *filename</td>
<td>Correspond object file name</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

Dim ret as Long
Dim addr as Long
Dim filename as String
addr = &hF0000
ret = pd.sym_get_scope ( addr, filename )
If ret = 0 Then
    pd.err_disp_message
End If
sym_get_src_name

Description
Get source file names from an object.

Parameters
\[
\text{ret} = \text{pd.sym_get_src_name}(\text{obj}, \text{src}, \text{mode})
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR obj</td>
<td>Object name</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *src</td>
<td>Source filename</td>
</tr>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Search mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2: \text{(LOAD_FIRST)}) First</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3: \text{(LOAD_NEXT)}) Second and following source name</td>
</tr>
</tbody>
</table>

When NULL is given to an object name, it searches from all objects.

Returned value

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded:</td>
<td>1</td>
</tr>
<tr>
<td>Error:</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```vba
Const LOAD_FIRST = 2
Const LOAD_NEXT = 3
Dim ret As Long
Dim obj As String
Dim src As String
Dim sw As Long

sw = LOAD_FIRST
obj = “main”

Do
    ret = pd.sym_get_src_name(obj, src, sw)
    If ret = 0 Then
        Exit Do
    Else
        MsgBox src
        sw = LOAD_NEXT
    End If
Loop
```
sym_get_variable_info

Description
Get the information of a specified C variable.

Parameters
\[
\text{ret} = \text{pd.sym_get_variable_info} \left( \text{var}, \text{type, l_data, h_data, str1, str2} \right)
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR var</td>
<td>Variable name</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *type</td>
<td>Variable type</td>
</tr>
<tr>
<td>[out]</td>
<td>long *l_data</td>
<td>Variable data(low 32 bits)</td>
</tr>
<tr>
<td>[out]</td>
<td>long *h_data</td>
<td>Variable data(high 32 bits)</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *str1</td>
<td>Variable information 1</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *str2</td>
<td>Variable information 2</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret As Long
Dim var as String
Dim type As String
Dim lData As Long
Dim hData As Long
Dim str1 As String
Dim str2 As String

var="aaa"
ret = pd.sym_get_func_info(var, type, l_data, h_data, str1, str2)
If ret = 0 Then
  pd.err_disp_message
End If
```
**sym_line2addr**

**Description**
This function gets the address of specified source line information.

**Parameters**

\[ \text{ret} = \text{pd.sym\_line2addr}( \text{line}, \text{filaname}, \text{addr} ) \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR filename</td>
<td>Source file name</td>
</tr>
<tr>
<td>[in]</td>
<td>long line</td>
<td>Source line number</td>
</tr>
<tr>
<td>[out]</td>
<td>long *addr</td>
<td>Correspond address</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```c
Dim ret as Long
Dim addr as Long
Dim line as Long
Dim filename as String

filename = "samp.c"
line = 10
ret = pd.sym_line2addr ( filename, line, addr )
If ret = 0 Then
    pd.err dispersion_message
End If
```
sym_set_disp_src

Description
This function changes the source file at program window.

Parameters
\[
\text{ret} = \text{pd.sym_set_disp_src} ( \text{filename, line, addr} )
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR filename</td>
<td>Source file name</td>
</tr>
<tr>
<td>[in]</td>
<td>long line</td>
<td>Source line number</td>
</tr>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
</tbody>
</table>

If selected line of the selected source file cannot be displayed, the file is displayed in the disassemble mode beginning with the address specified by “addr”.

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```vbnet
Dim ret as Long
Dim filename as String
Dim line as Long
dim addr as Long

filename = "main.c"
line = 5
addr = &hF0000
ret = pd.sym_set_disp_src ( filename, line, addr )
If ret = 0 Then
    pd.err_disp_message
End If
```
sym_set_scope_addr

Description
This function sets the current scope of specified address.

Parameters
ret = pd.sym_set_scope_addr ( addr )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Scope address</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Succeeded:</th>
<th>Error:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long
Dim addr as Long
Dim filename as String

addr = &hF0000
ret = pd.sym_set_scope_addr ( addr )
If ret = 0 Then
    pd.err_disp_message
End If
```
sym_set_scope_obj

Description
This function sets the current scope of specified object file.

Parameters
ret = pd.sym_set_scope_obj ( obj )

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>BSTR obj</td>
<td>Object file name</td>
</tr>
</tbody>
</table>

Returned value

<table>
<thead>
<tr>
<th>Succeeded</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The error message can be displayed by calling method err_disp_message.

Description example

```vbnet
Dim ret as Long
Dim obj as String

obj = "main.r30"
ret = pd.sym_set_scope_obj ( obj )
If ret = 0 Then
    pd.err_disp_message
End If
```
sym_sym2val

**Description**
This function gets the value that corresponds to a string character.

**Parameters**

\[
\text{ret} = \text{pd.sym_sym2val}( \text{mode}, \text{symbol}, \text{value} )
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Search mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 : (LOAD_LABEL) Label first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 : (LOAD_SYMBOL) Symbol first</td>
</tr>
<tr>
<td>[in]</td>
<td>BSTR symbol</td>
<td>Symbol</td>
</tr>
<tr>
<td>[out]</td>
<td>long *value</td>
<td>Value</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
Const LOAD_SYMBOL = 1
Dim ret as Long
Dim symbol as String
Dim value as Long

symbol = "data1"
ret = pd.sym_sym2val (LOAD_SYMBOL, str, value)
If ret = 0 Then
    pd.err_disp_message
End If
```
**sym_val2bit**

**Description**
This function gets the bit symbol that corresponds to address and bit number.

**Parameters**

\[
\text{ret} = \text{pd.sym_val2bit ( addr, bit, symbol )}
\]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long addr</td>
<td>Address</td>
</tr>
<tr>
<td>[in]</td>
<td>long bit</td>
<td>Bit number</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *symbol</td>
<td>Area in which bit symbol is stored</td>
</tr>
</tbody>
</table>

**Returned value**

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method `err_disp_message`.

**Description example**

```vbnet
Const LOAD_LABEL = 0
Dim ret as Long
Dim addr as Long
Dim bit as Long
Dim symbol as String

addr = &h400
bit = 0
ret = pd.sym_val2bit ( addr, bit, symbol )
If ret = 0 Then
    pd.err_disp_message
End If
```
sym_val2sym

Description
This function gets the symbol that corresponds to value.

Parameters
ret = pd.sym_val2sym (mode, value, symbol)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type &amp; Argument</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>long mode</td>
<td>Search mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 : (LOAD_LABEL) Label first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 : (LOAD_SYMBOL) Symbol first</td>
</tr>
<tr>
<td>[in]</td>
<td>long value</td>
<td>Value</td>
</tr>
<tr>
<td>[out]</td>
<td>BSTR *symbol</td>
<td>Area in which symbol is stored</td>
</tr>
</tbody>
</table>

Returned value

| Succeeded: | 1 |
| Error:     | 0 |

The error message can be displayed by calling method err_disp_message.

Description example

```
Const LOAD_LABEL = 0
Dim ret as Long
Dim symbol as String
Dim value as Long

value = &hF0000
ret = pd.sym_val2sym (LOAD_SYMBOL, value, symbol)
If ret = 0 Then
    pd.err_disp_message7
End If
```
5.3 Event Number List

The events generated by the PDxx are listed in the table below.

<table>
<thead>
<tr>
<th>Event Number</th>
<th>Event Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>(EVENT_GO) Program Start</td>
</tr>
<tr>
<td>1002</td>
<td>(EVENT_STOP) Program Stop</td>
</tr>
<tr>
<td>1003</td>
<td>(EVENT_RESET) Program Reset</td>
</tr>
<tr>
<td>1004</td>
<td>(EVENT_STEP) Step Execution</td>
</tr>
<tr>
<td>1005</td>
<td>(EVENT_OVER) Over Step Execution</td>
</tr>
<tr>
<td>1006</td>
<td>(EVENT_RETURN) Return Execution</td>
</tr>
<tr>
<td>1007</td>
<td>(EVENT_PUT_REG) Change Register</td>
</tr>
<tr>
<td>1008</td>
<td>(EVENT_REG_PC) Change PC</td>
</tr>
<tr>
<td>1009</td>
<td>(EVENT_PUT_MEM) Change Data</td>
</tr>
<tr>
<td>1010</td>
<td>(EVENT_LOAD) Program Download</td>
</tr>
<tr>
<td>1013</td>
<td>(EVENT_SBRK) Change S/W Break Point</td>
</tr>
<tr>
<td>1014</td>
<td>(EVENT_TRACE_START) Start Real-time Trace</td>
</tr>
<tr>
<td>1015</td>
<td>(EVENT_TRACE_END) End Real-time Trace</td>
</tr>
<tr>
<td>1016</td>
<td>(EVENT_TRACE_PASS) Pass Real-time Trace Point</td>
</tr>
<tr>
<td>1021</td>
<td>(EVENT_MAP) Change Memory Mapping</td>
</tr>
</tbody>
</table>