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

April 1st, 2010
Renesas Electronics Corporation

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

Send any inquiries to http://www.renesas.com/inquiry.

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.

User's Manual

PDSDK COM Kit

Reference Manual

Renesas Electronics
www.renesas.com

Rev.2.00 2006.09

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.

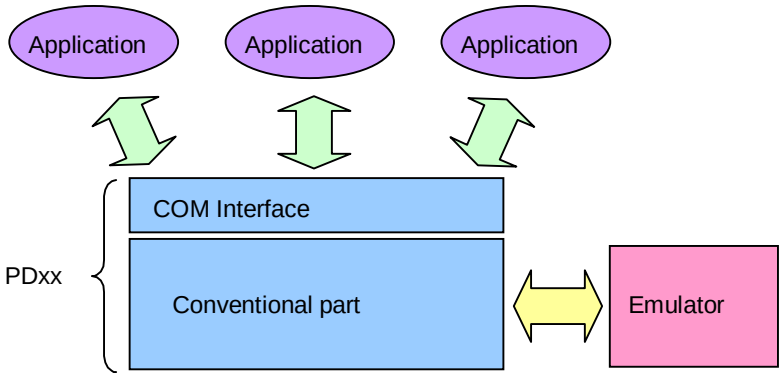
Technical support

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.

[Contents]

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.

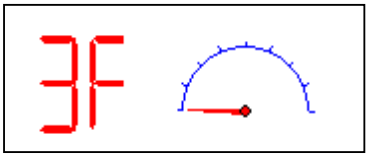


1.1 Development Tools Used

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.

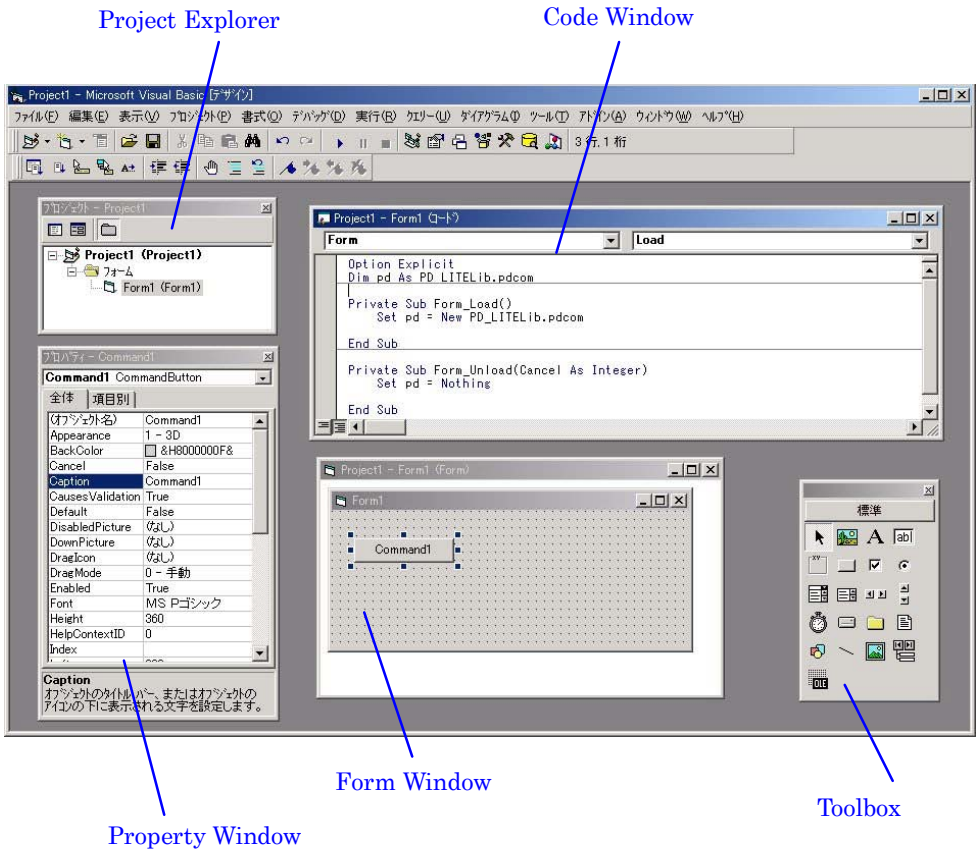
[1] COM (Component Object Model), proposed by Microsoft, is a standard for connecting application programs.

2. Setup

For details on how to set up the PDSDK COM kit, refer to the Release Notes for the PDSDK COM kit.

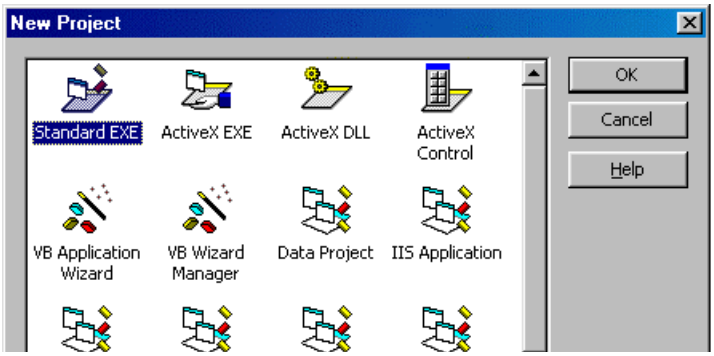
3. Windows Creating (Visual Basic)

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

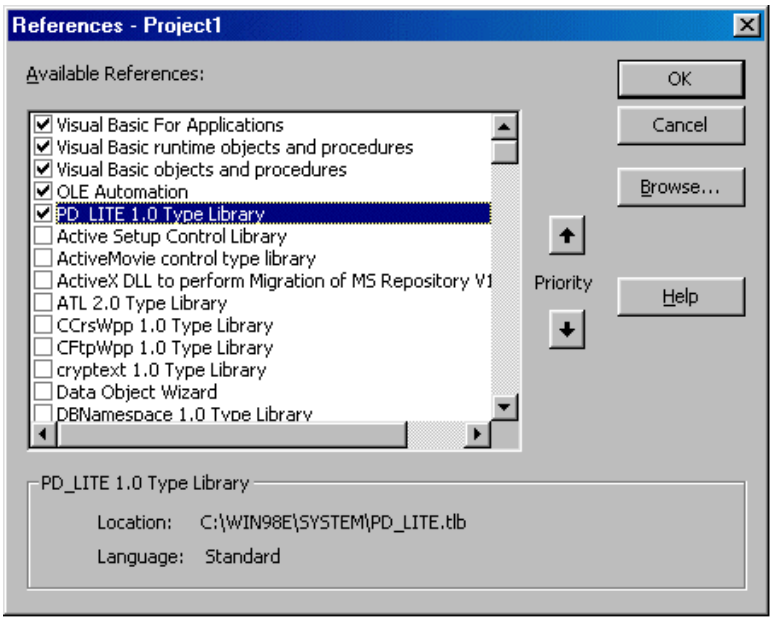
Generate the new project: select Visual Basic Menu “File”, then “New Project”. The “New Project” dialog opens. With this dialog, select “Standard EXE” and click the mouse on the “OK” button.



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.

```
1:  Dim WithEvents pd As PD_LITELib.pdcom
2:
3:  Private Sub Form_Load()
4:      Set pd = New PD_LITELib.pdcom
5:  End Sub
6:
7:  Private Sub Form_Unload(Cancel As Integer)
8:      Set pd = Nothing
9:  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.

pd.xxxxx (Parameter, …)

For the details on each method, refer to “4.

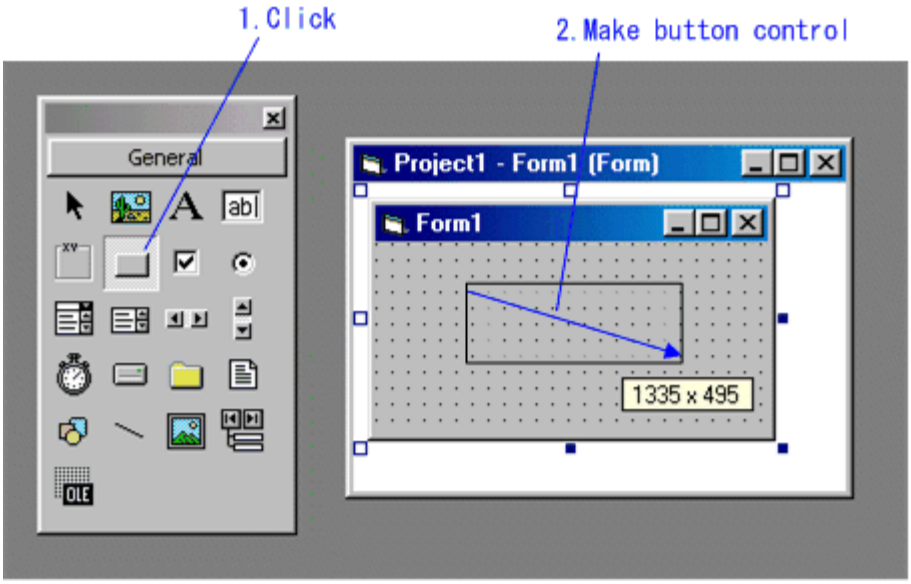
Method List". Explanation is given below of the specific procedure for calling the method.

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

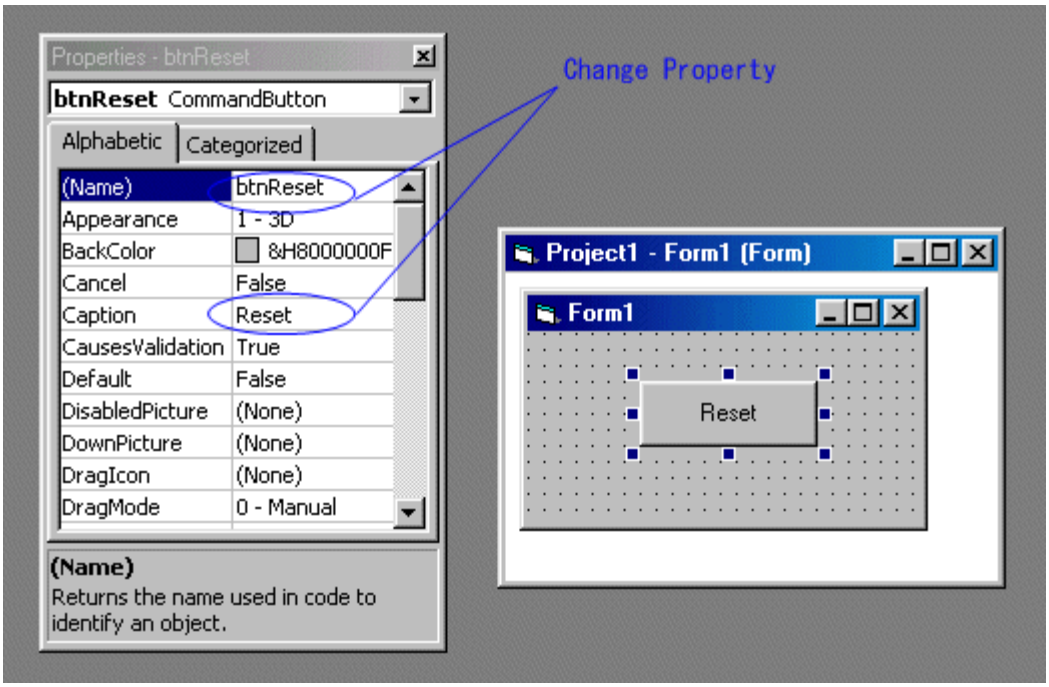
   Click the mouse on the Command button of the tool box to create one button control on the form.



2. Button Property Change

In the property window, alter the properties for the button control you created.

Property	Contents
(Name)	btnReset
Caption	Reset



3. Describe the button operation

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

```
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 xxxxx.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.)

```
Private Sub pd_GotEventMessage(ByVal action As Long)
        :
End Sub
```

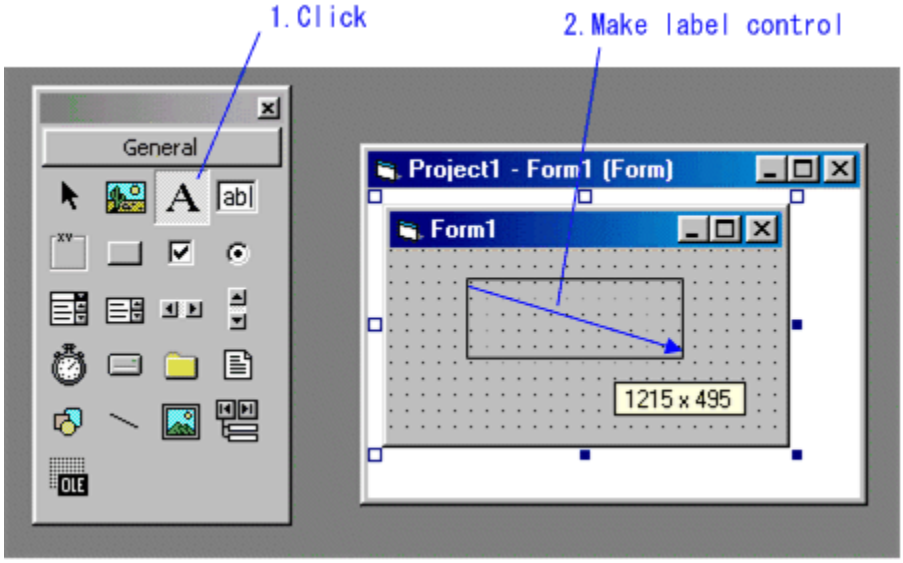
Event No. arising on the PDxx side is stored in parameter action. With this procedure, describe the processing to be executed where the event was received from PDxx. For details about the generated event numbers, refer to Section 5.3, "Event Number List."

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.

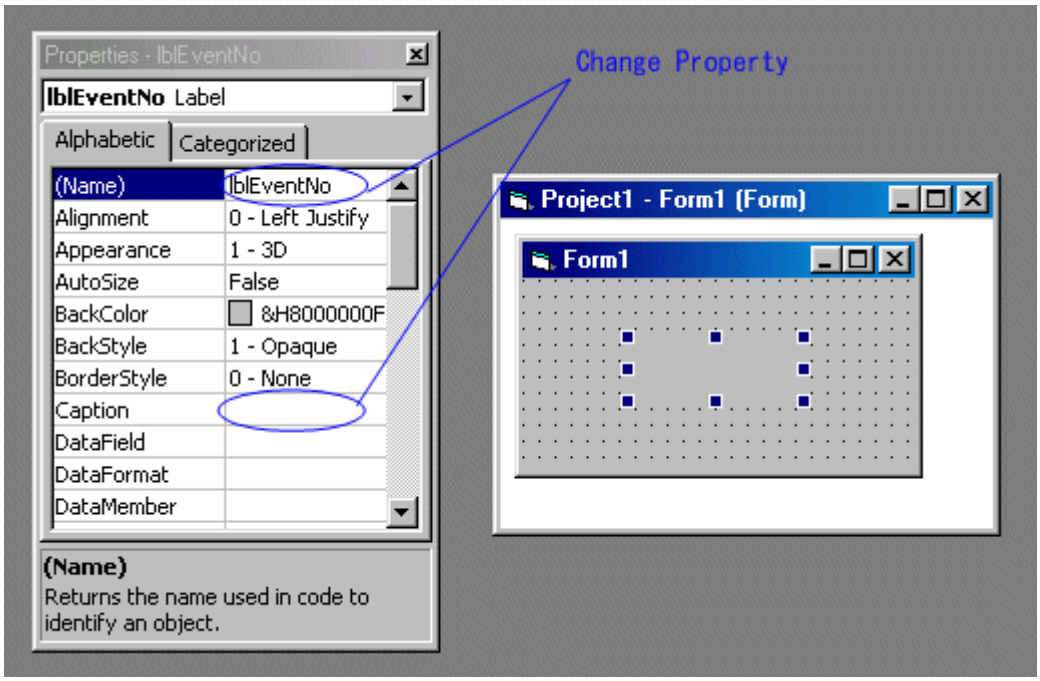
1. Add Label Control

   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.

Property	Contents
(Name)	lblEventNo
Caption	(nothing)



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

```
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
```

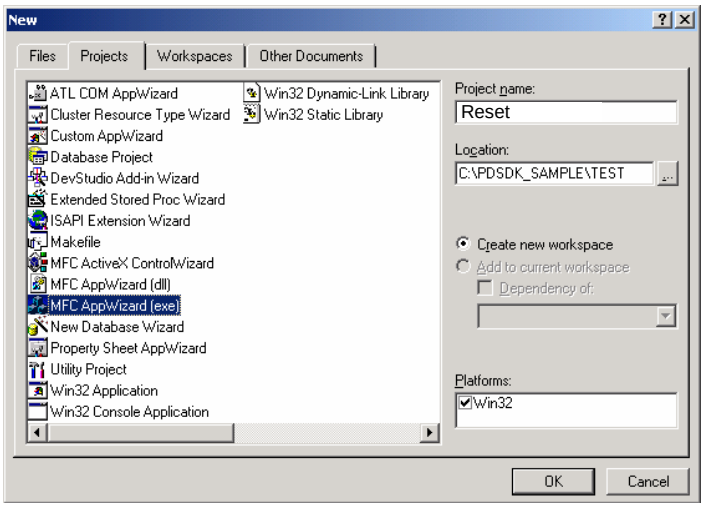
4. Check for action
Check the applications action. First, run PDxx. Next, select the “Visual Basic” menu [EXEC], then [START], and execute the applications. By manipulating PDxx (go/stop, etc.), the number of an event that occurred in its PDxx is displayed. Check its action.

4. Example of creating PDSDK-Window (Visual C++)

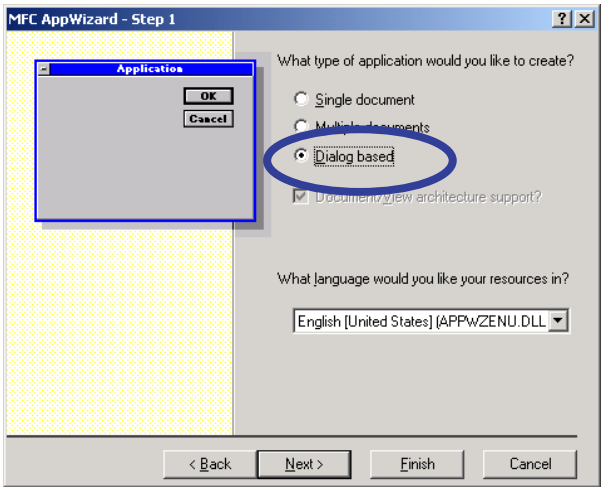
This chapter explains the creation method of the customize window for PDxx using Visual C++ 6.0 (hereinafter described as "VC++"). Before using VC++, please be accustomed with its basic language specification and method of use by referring to VC++ online help or reference books available on the market.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

4.1 Generating a project

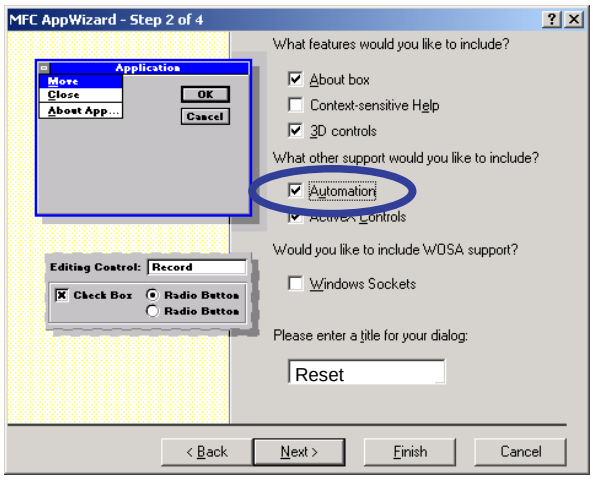
To create a new project select NEW from VC++ FILE-menu. This will open the New Project dialog.
Please select MFC AppWizard and push OK.



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.



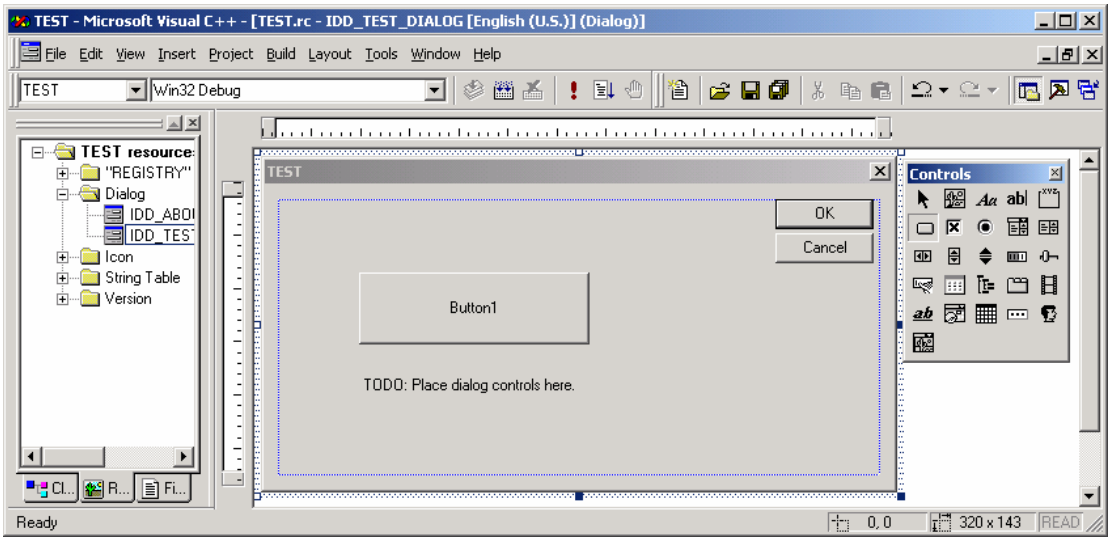
In step 2 please select AUTOMATION. The other settings can be left on their default setting. Then push again NEXT.



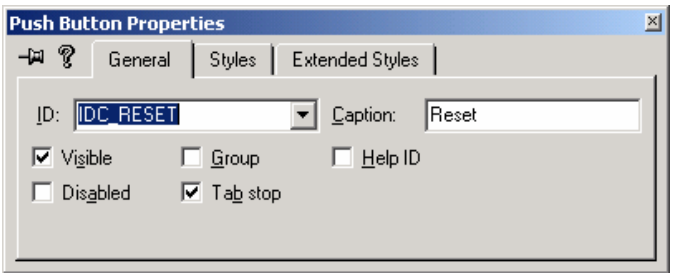
Then you can push FINISH to exit the Application Wizard. All other settings remain on default settings.

4.2 Creating a Button

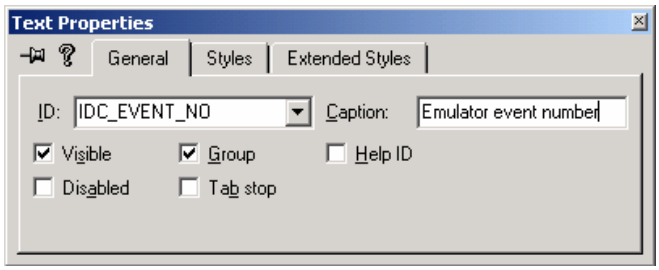
After creating the new project, the VC++ desktop shows the empty dialog window for our RESET application. In this window you can place various objects from the Control box on your application. In this example we simply want to create a push button.
Therefore you have to select the button symbol in the 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.

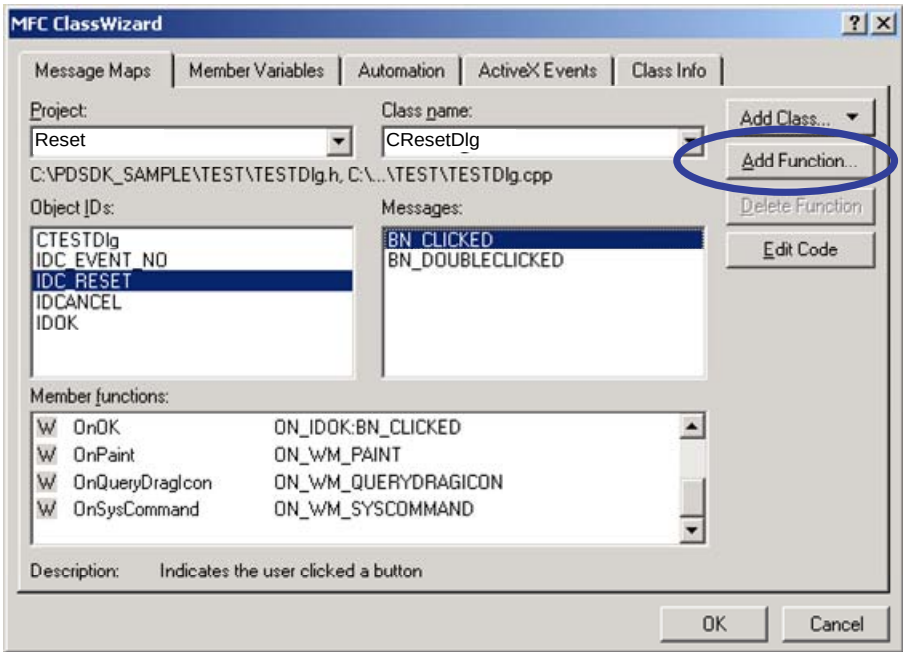


The text box below the button can be used to receive messages from PDxx. Therefore we also first change its references. Select the properties of the text box and change ID and caption as shown below.



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)
}
```

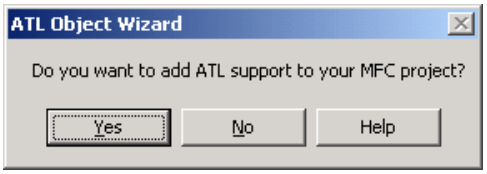
(1) Import the library of PDCOM. This is necessary to be able to use the PDCOM interface and its methods.
(2) "p" is the smart pointer of the PDCOM interface. A smart pointer name is I interface name Ptr. Since the interface name of PDCOM is pdcom, a smart pointer serves as IpdcomPTR.
(3) Call the method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

4.3 Acquisition of Events from PDxx

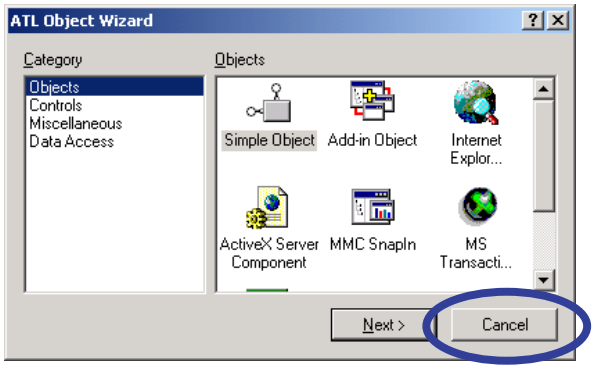
In order to communicate with the PDxx debugger (start/stop target program, read/modify memory or register contents, ...) it is necessary to acquire an event from PDxx.
This chapter explains how to mount the acquisition function of an event in the "Reset" window of the previous chapter.

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).

```
#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).

```
#include "stdafx.h"
#import “pd_lite.tlb” no_namespace, named_guids                          (1)
#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;
  USES_CONVERSION;
  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]));     (2)
    if(FAILED(hr))
      return DISP_E_TYPEMISMATCH;
    SetDlgItemText(m_hWnd, IDC_EVENT_NO, OLE2T(varMessage.bstrVal)); (3)(4)

    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).

```
class CresetDlgAutoProxy;
interface Ipdcom;
class CpdcomEvents;
```

Add a new member variable to ResetDlg.H:

```
private:
  CComPtr <Ipdcom> m_ppdcom;
  CComObject<CpdcomEvents>* m_ppdcomEvents;
  DWORD m_dwCookie;
```

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.

```
#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).

```
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.

```
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__IpdcomEvents, 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().

```
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_pdcom, NULL, CLSCTX_ALL, IID_Ipdcom,
              (void**) &m_ppdcom);

  if (FAILED(hr))
  {
    PostMessage(WM_CLOSE, 0, 0L);
    return TRUE;
  }

  CComObject<CpdcomEvents>::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

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

Register

Method Name	Parameter	Description	Page
reg_get_pc	[out] long* pc	Gets the value of program counter.	78
reg_set_pc	[in] long pc	Sets a program counter value.	80
reg_get_reg	[in] long regNo [out] long* regVal	Gets the value of specified register.	79
reg_set_reg	[in] long regNo [in] long regVal	Sets the value of specified register.	81

Memory

Method Name	Parameter	Description	Page
mem_get	[in] long addr [in] long length [out] long* data	Gets a memory data. (1 data)	63
mem_get_multi	[in] long addr [in] long num [in] long length [out] VARIANT* data	Gets a memory data. (multi data)	65
mem_set	[in] long addr [in] long length [in] long data	Sets a memory data. (1 data)	68
mem_set_multi	[in] long addr [in] long num [in] long length [in] VARIANT data	Sets memory data. (multi data)	70

mem_get_bit	[in] long addr [in] long bit [out] long* data	Gets a bit data.	64
mem_set_bit	[in] long addr [in] long bit	Sets a bit data.	69
mem_clear_bit	[in] long addr [in] long bit	Clears a bit data.	62
mem_fill	[in] long addr [in] long num [in] long length [in] long data	Fills the memory data.	66
mem_move	[in] long addr [in] long num [in] long top [in] long length	Transfers the memory data.	67

RAM Monitor

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

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

Software Breaks

Method Name	Parameter	Description	Page
break_set	[in] long addr	Registers the software breakpoint.	30
break_reset	[in] long addr	Deletes the software breakpoint.	27
break_reset_all	(nothing)	Deletes all software breakpoints.	28
break_disable	[in] long addr	Disables the software breakpoint.	23
break_disable_all	(nothing)	Disables all software breakpoints.	24
break_enable_all	(nothing)	Enables all software breakpoints.	25
break_get	[out] long* addr [out] long* attr [in] long mode	Searches the software breakpoint.	26
break_search	[in] long addr [out] long* attr	Gets the breakpoint attribute.	29

Real-time Trace

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

Coverage

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

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

Symbol

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

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

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

Dwonloads

Method Name	Parameter	Description	Page
down_load	[in] BSTR filename [in] long mode	Downloads the target program.	53

MCU/ PDxx information

Method Name	Parameter	Description	Page
info_cpu	[in] long flag [out] long* status	Gets the information of target MCU.	58
info_service	[in] long flag [out] long* status	Gets the supported information of PDxx.	61

info_get_debugger*	[out] BSTR *name [out] BSTR *dir [out] BSTR *ini	Get the debugger information such as a product name.	59
info_get_mcufile_name*	[out] BSTR *name	Get the MCU file name selected in the INIT dialog box.	60

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

Others

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

* 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

ret = pd.break_disable (addr)

Attribute	Type & Argument	Content
[in]	long addr	Breakpoint address

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_disable ( addr )
If ret = 0 Then
   pd.err_disp_message
End If
```

break_disable_all

Description

This function disables all software breakpoints.

Parameters

ret = pd.break_disable_all ( )

There is no 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

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

Succeeded:	1
Error:	0

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

ret = pd.break_get (addr, attr, mode)

Attribute	Type & Argument	Content
[out]	long *addr	Breakpoint address
[out]	long *attr	Setup attributes **256 :** IN1_ENABLE_SBRK Enable **257 :** IN1_DISABLE_SBRK Disable
[in]	long mode	Search mode **258 :** (IN1_FIRST) First breakpoint **259 :** (IN1_NEXT) Second and following breakpoints

Returned value

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
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)

Attribute	Type & Argument	Content
[in]	long addr	Breakpoint address

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_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

Succeeded:	1
Error:	0

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)

Attribute	Type & Argument	Content
[in]	long addr	Check address
[out]	long *attr	Setting of breakpoint **256 :** (IN1_ENABLE_SBRK) Enable **257 :** (IN1_DISABLE_SBRK) Disable

Returned value

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
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)

Attribute	Type & Argument	Content
[in]	long addr	Breakpoint address

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 = pd.cpu_check_run (status)

Attribute	Type & Argument	Content
[out]	long *status	Status of target program **260 :** (IN1_RUN_CPU) Run **261 :** (IN1_STOP_CPU) Stop

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

ret = pd.cpu_gb ( )

There is no 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

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

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
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

ret = pd.cpu_over ( )

There is no 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

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

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
Dim ret as Long

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

cpu_return

Description

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

Products	Mnemonic
PD32R(SIM)	JMP R14,RTE
PD308(SIM), PD30(SIM)	REIT,RTS,FREIT,EXITD
PD79(SIM)	RTS,RTI,RTL,RTSD,RTLD
PD77(SIM)	RTS,RTI,RTL
PD38(SIM)	RTS,RTI
PD10V	JMP R13, RTE

Parameters

ret = pd.cpu_return ( )

There is no 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

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

Succeeded:	1
Error:	0

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

ret = pd.cpu_src_step ( )

There is no 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

ret = pd.cpu_src_step
If ret = 0 Then
   pd.err_disp_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

Succeeded:	1
Error:	0

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

ret = pd.cpu_stop ( )

There is no 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

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

ret = pd.cpu_wait ( )

There is no 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

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

com_receive

Description

This function receives a data from Emulator.

Parameters

ret = pd.com_receive (data, size)

Attribute	Type & Argument	Content
[out]	BSTR *data	Receive data
[in]	long size	Receive size

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 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

ret = pd.com_send (data, size)

Attribute	Type & Argument	Content
[in]	BSTR data	Send data
[in]	long size	Send size

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 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

Succeeded:	1
Error:	0

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

ret = pd.cv_clear_blk (blkno)

Attribute	Type & Argument	Content
[in]	long blkno	Coverage block number

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 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

ret = pd.cv_get_base (base)

Attribute	Type & Argument	Content
[out]	long *base	Beginning address

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 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)

Attribute	Type & Argument	Content
[out]	VARIANT *base	Beginning address

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)

Attribute	Type & Argument	Content
[in]	long blkno	Coverage block number
[out]	long *base	Beginning address

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 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)

Attribute	Type & Argument	Content
[out]	long *cnt	Coverage block count

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 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”.

Attribute	Type & Argument	Content
[in]	long st_addr	Start address
[in]	long en_addr	End address
[out]	long *rs_addr	Real start address
[out]	long *re_addr	Real end address
[out]	VARIANT *data	Coverage data

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)

7	6	5	4	3	2	1	0
+7	+6	+5	+4	+3	+2	+1	+0

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)

7	6	5	4	3	2	1	0
407	406	405	404	403	402	401	400

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)

7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1

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

```
'
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”.

Attribute	Type & Argument	Content
[in]	long base	Beginning address

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

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 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

ret = pd.down_load (filename, mode)

Attribute	Type & Argument	Content
[in]	BSTR filename	Download filename
[in]	long mode	Download mode **4** : (LOAD_LOAD) All Road module **5** : (LOAD_SYM) Symbol information only **6** : (LOAD_ROM) Machine data only

Returned value

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
'
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

Succeeded:	1
Error:	0

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

ret = pd.err_get_message (msg)

Attribute	Type & Argument	Content
[out]	BSTR *msg	Error message

Returned value

Succeeded:	1
Error:	0

Description example

```
'
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

ret = pd.event_set_request_mode (mode)

Attribute	Type & Argument	Content
[in]	long mode	Request mode **0:** (EVT_DISABLE) Hide event output **1:** (EVT_ENABLE) Event output

Returned value

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
'
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)

Attribute	Type & Argument	Content
[in]	BSTR str	Character Symbol
[in]	long radix	Effective radix **16 :** (EXP_HEX) Hexdecimal **10 :** (EXP_DEX) Decimal **0 :** (EXP_DEFAULT) Default
[in]	long mode	Search priority **0:** (EXP_LABEL) Label first **1:** (EXP_SYMBOL) Symbol first
[out]	long *data	Correspond data

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)

Attribute	Type & Argument	Content	
[in]	long flag	Information number	
		262 : (IN1_ADDRSIZE)	Number of bytes required for string address value
		263 : (IN1_MAXADDR)	Maximum value of address
		264 : (IN1_ADDRCOLM)	Number of digits with which address values are displayed in hexadecimal
		265 : (IN1_ENDIAN)	Endian of the target CPU
		267 : (IN1_WORD_SIZE)	Length in bytes of word
		270 : (IN1_MAX_OBJ)	Maximum length in bytes of one instruction
		271 : (IN1_MAXDATA)	Maximum value of data
		272 : (IN1_MAXSTACK)	Maximum value of stack
		298 : (IN1_BUS_SIZE)	Data bus width
[out]	long *status	Value corresponding to flag	

Returned value

There is no parameter.

Description example

```
'
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)

Attribute	Type & Argument	Content
[out]	BSTR *name	Debugger name + version
[out]	BSTR *dir	Directory of PDxx.exe
[out]	BSTR *ini	Ini filename

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
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)

Attribute	Type & Argument	Content
[out]	BSTR *name	MCU filename

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)

Attribute	Type & Argument	Content	
[in]	long flag	Information number	
		275 : (IN1_SUPPORT_BITSYM)	Bit symbol
		276 : (IN1_SUPPORT_C)	C-language debugging
		277 : (IN1_SUPPORT_RAMMONITOR)	Real-time RAM monitor
		278 : (IN1_SUPPORT_RTT)	Real-time trace
		279 : (IN1_SUPPORT_CV)	Coverage
		280 : (IN1_SUPPORT_PROTCT)	Protect break
[out]	long *status	Value corresponding to flag	

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)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long bit	Bit number

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)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long length	Data length **1**: 1 byte **2**: 2 bytes* **4**: 4 bytes*
[out]	long *data	Data

* 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
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)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long bit	Bit number
[out]	long *data	Bit value

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)

Attribute	Type & Argument	Content
[in]	long addr	start address
[in]	long num	get size
[in]	long length	data length 1: 1byte 2: 2 bytes* 4: 4 bytes*
[out]	VARIANT *data	Saving data

* 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

ret = pd.mem_fill (addr, length, size, data)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long num	Address size
[in]	long length	Data Length 1: 1 byte 2: 2 bytes* 4: 4 bytes*
[in]	long data	Fill data

* 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

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

ret = pd.mem_move (addr, length, top, size)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long num	Address size
[in]	long top	Transfer address
[in]	long length	Data length 1: 1 byte 2: 2 bytes* 4: 4 bytes*

* 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 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
```

mem_set

Description

This function sets a memory data. (1 data)

Parameters

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

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long length	Data length 1: 1 byte 2: 2 bytes* 4: 4 bytes*
[in]	long data	Set data

* 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)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long bit	Bit number

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)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long num	Address size
[in]	long length	Data length 1: 1 byte 2: 2 bytes* 4: 4 bytes*
[in]	VARIANT data	

* 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

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

Succeeded:	1
Error:	0

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

ret = pd.rram_clear_blk (blkno)

Attribute	Type & Argument	Content
[in]	long blkno	RAM monitor block number

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 blkno as Long

blkno = 0
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

ret = pd.rram_get_area (addr)

Attribute	Type & Argument	Content
[out]	long *addr	Top address of RAM monitor

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 biginning address is 400h, the RAM monitor 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
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

ret = pd.rram_get_area (blkno, addr)

Attribute	Type & Argument	Content
[in]	long blkno	RAM monitor block number
[out]	long *addr	Top address of RAM monitor

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 biginning address is 400h, the RAM monitor 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
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

ret = pd.rram_get_data (addr, num, data, attr)

The address outside the RAM monitor area cannot be specified.

Attribute	Type & Argument	Content
[in]	long addr	Start Address
[in]	long num	Obtain data size
[out]	VARIANT *data	RAM monitor data
[out]	VARIANT *attr	RAM monitor access attributes **0 :** No access **1 :** Write **2 :** Read

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 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, area )
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)

Attribute	Type & Argument	Content
[in]	long addr	Top address of RAM monitor

The RAM monitor area is 1K consecutive bytes which starts from 16 bytes boundary. When the biginning 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)

Attribute	Type & Argument	Content
[in]	long blkno	RAM monitor block number
[in]	long addr	Top address of RAM monitor

The RAM monitor area is 1K consecutive bytes which starts from 16 bytes boundary. When the biginning 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
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

ret = pd.reg_get_pc (pc)

Attribute	Type & Argument	Content
[out]	long *pc	Program counter value

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

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

reg_get_reg

Description

This function gets the value of specified register.

Parameters

ret = pd.reg_get_reg (regNo, regVal)

Attribute	Type & Argument	Content
[in]	long regNo	Register number
[out]	long *regVal	Register value

A register number changes with debugger.

PD32R/ PD32RSIM	0:R0, 1:R1, 2:R2, 3:R3, 4:R4, 5:R5, 6:R6, 7:R7, 8:R8, 9:R9, 10:R10, 11:R11, 12:R12, 13:R13, 14:R14, 15:R15, 16:PSW, 17:SPI, 18:SPU, 19:BPC, 20:PC, 21:ACC0H, 22:ACC0L, 23:ACC1H, 24:ACC1L
PD308/ PD308SIM	0:0R0, 1:0R1, 2:0R2, 3:0R3, 4:0A0, 5:0A1, 6:0FB, 7:0SB, 8:1R0, 9:1R1, 10:1R2, 11:1R3, 12:1A0, 13:1A1, 14:1FB, 15:1SB, 16:USP, 17:ISP, 18:FLG, 19:PC, 20:INTB, 21:SVF, 22:SVP, 23:VCT, 24:DMD0, 25:DMD1, 26:DCT0, 27:DCT1, 28:DRC0, 29:DRC1, 30:DMA0, 31:DMA1, 32:DCA0, 33:DCA1, 34:DRA0, 35:DRA1
PD30/ PD30SIM	0:0R0, 1:0R1, 2:0R2, 3:0R3, 4:0A0, 5:0A1, 6:0FB, 7:1R0, 8:1R1, 9:1R2, 10:1R3, 11:1A0, 12:1A1, 13:1FB, 14:USP, 15:ISP, 16:FLG, 17:PC, 18:SB, 19:INTB
PD79/ PD79SIM	0:A, 1:B, 2:X, 3:Y, 4:S, 5:PC, 6:DT, 7:PS, 8:PG, 9:DP0, 10:DP1, 11:DP2, 12:DP3, 13:E, 14:PGPC
PD77/ PD77SIM	0:A, 1:B, 2:X, 3:Y, 4:S, 5:PC, 6:DT, 7:PS, 8:PG, 9:DPR, 10:PGPC
PD38/ PD38SIM	0:A, 1:X, 2:Y, 3:S, 4:PC, 5:F

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 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
```

reg_set_pc

Description

This function sets a program counter value.

Parameters

ret = pd.reg_set_pc (pc)

Attribute	Type & Argument	Content
[in]	long pc	Program counter value

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 (regNo, regVal)

Attribute	Type & Argument	Content
[in]	long regNo	Register number
[in]	long regVal	Register value

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 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

ret = pd.rtt_check_isfetch (cycle, addr1, addr2, count)

Attribute	Type & Argument	Content
[in]	long cycle	Cycle
[out]	long *addr1	1st fetch cycle
[out]	long *addr2	2nd fetch cycle
[out]	long *count	Fetch mnemonic count

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

```
'
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

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
'
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)

Attribute	Type & Argument	Content
[in]	long cycle	Cycle
[out]	long *addr	Address
[out]	BSTR *buffer	BUS character string

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 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)

Attribute	Type & Argument	Content
[in,out]	long *cycle	Cycle
[out]	long *next_cycle	Next fetch cycle
[out]	BSTR *buffer	Disassemble character string
[out]	long *count	Disassemble mnemonic count

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

```
'
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.err_disp_message
End If
label.Caption = buffer
```

rtt_get_range

Description

This function gets the trace range of trace area.

Parameters

ret = pd.rtt_get_range (s_cycle, e_cycle)

Attribute	Type & Argument	Content
[out]	long *s_cycle	Start cycle
[out]	long *e_cycle	End cycle

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 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

ret = pd.scri_command (string)

Attribute	Type & Argument	Content
[in]	BSTR string	Script command string

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 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

ret = pd.scri_print (string)

Attribute	Type & Argument	Content
[in]	BSTR string	Display string

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 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)

Attribute	Type & Argument	Content
[in]	BSTR name	Enter bit symbol string
[in]	long addr	Address
[in]	long bit	Bit number

When the global bit 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

```
'
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

ret = pd.sym_add_label (name, value)

Attribute	Type & Argument	Content
[in]	BSTR name	Enter label string
[in]	long value	Address

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

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

name = &hF0001
value = "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

ret = pd.sym_add_symbol (name, value)

Attribute	Type & Argument	Content
[in]	BSTR name	Enter symbol string
[in]	long value	Value

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

```
'
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

ret = pd.sym_addr2line (addr, line, filaname)

Attribute	Type & Argument	Content
[in]	long addr	Address
[out]	long *line	Source line number
[out]	BSTR *filename	Source file name

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 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

ret = pd.sym_bit2val (symbol, addr, bit)

Attribute	Type & Argument	Content
[in]	BSTR symbol	Bit Symbol String
[out]	long *addr	Address
[out]	long *bit	Bit number

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 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
```

sym_get_disp_src

Description

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

Parameters

ret = pd.sym_get_disp_src (filaname)

Attribute	Type & Argument	Content
[out]	BSTR *filename	Source file name

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 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

ret = pd.sym_get_func_info (func, type, start, end, str)

Attribute	Type & Argument	Content
[in]	BSTR func	Function name
[out]	BSTR *type	Function type
[out]	long *start	Start address
[out]	long *end	End address
[out]	BSTR *str	Address range string

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 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

ret = pd.sym_get_func_name (obj, func, mode)

Attribute	Type & Argument	Content	
[in]	BSTR obj	Object name	
[out]	BSTR *func	Function name	
[in]	long mode	Search mode	
		2 : (LOAD_FIRST)	First
		3 : (LOAD_NEXT)	Second and following function name

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

ret = pd.sym_get_obj_name (obj, mode)

Attribute	Type & Argument	Content
[out]	BSTR *obj	Object name
[in]	long mode	Search mode **2** : (LOAD_FIRST) First **3** : (LOAD_NEXT) Second and following object name

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 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, filaname)

Attribute	Type & Argument	Content
[in]	long addr	Address
[out]	BSTR *filename	Correspond object file name

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

ret = pd.sym_get_src_name (obj, src, mode)

Attribute	Type & Argument	Content
[in]	BSTR obj	Object name
[out]	BSTR *src	Source filename
[in]	long mode	Search mode **2** : (LOAD_FIRST) First **3** : (LOAD_NEXT) Second and following source name

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

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 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

ret = pd.sym_get_variable_info (var, type, l_data, h_data, str1, str2)

Attribute	Type & Argument	Content
[in]	BSTR var	Variable name
[out]	BSTR *type	Variable type
[out]	long *l_data	Variable data(low 32 bits)
[out]	long *h_data	Variable data(high 32 bits)
[out]	BSTR *str1	Variable information 1
[out]	BSTR *str2	Variable information 2

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

ret = pd.sym_line2addr (line, filaname, addr)

Attribute	Type & Argument	Content
[in]	BSTR filename	Source file name
[in]	long line	Source line number
[out]	long *addr	Correspond address

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 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_disp_message
End If
```

sym_set_disp_src

Description

This function changes the source file at program window.

Parameters

ret = pd.sym_set_disp_src (filaname, line, addr)

Attribute	Type & Argument	Content
[in]	BSTR filename	Source file name
[in]	long line	Source line number
[in]	long addr	Address

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

```
'
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)

Attribute	Type & Argument	Content
[in]	long addr	Scope address

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_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)

Attribute	Type & Argument	Content
[in]	BSTR obj	Object file name

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 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

ret = pd.sym_sym2val (mode, symbol, value)

Attribute	Type & Argument	Content
[in]	long mode	Search mode **0** : (LOAD_LABEL) Label first **1** : (LOAD_SYMBOL) Symbol first
[in]	BSTR symbol	Symbol
[out]	long *value	Value

Returned value

Succeeded:	1
Error:	0

The error message can be displayed by calling method err_disp_message.

Description example

```
'
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

ret = pd.sym_val2bit (addr, bit, symbol)

Attribute	Type & Argument	Content
[in]	long addr	Address
[in]	long bit	Bit number
[out]	BSTR *symbol	Area in which bit symbol is stored

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 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)

Attribute	Type & Argument	Content
[in]	long mode	Search mode **0** : (LOAD_LABEL) Label first **1** : (LOAD_SYMBOL) Symbol first
[in]	long value	Value
[out]	BSTR *symbol	Area in which symbol is stored

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.

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

PDSDK COM Kit
Reference Manual

Publication Date:	Sep. 16, 2006	Rev.2.00
Published by:	Sales Strategic Planning Div. Renesas Technology Corp.	
Edited by:	Microcomputer Tool Development Department Renesas Solutions Corp.	

© 2005. Renesas Technology Corp. and Renesas Solutions Corp., All rights reserved. Printed in Japan.

PDSDK COM Kit
Reference Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J0079-0200