RISCom-8 Windows Function Call Driver

USER'S GUIDE

RISCom-8
Windows Function Call Driver
User's Guide

Revision A - July 1994
Part Number: 77890

New Contact Information

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, OH 44139

Technical Support: 1-888-KEITHLEY
Monday – Friday 8:00 a.m. to 5:00 p.m (EST)
Fax: (440) 248-6168

Visit our website at http://www.keithley.com

Keithley MetraByte Division

Keithley Instruments, Inc.

440 Myles Standish Blvd. Taunton, MA 02780

Telephone: (508) 880-3000 • FAX: (508) 880-0179

Preface

The *RISCom-8 Windows Function Call Driver User's Guide* describes how to write application programs for the RISCom-8 board using Windows™ DLL function calls. The RISCom-8 Function Call Driver supports the following Windows-based languages:

- Microsoft QuickC® for Windows (Version 1.0)
- Microsoft Visual C++™ (Version 1.0)
- Borland® C/C++ (Version 3.1 and higher)
- Borland Turbo Pascal for Windows (Version 1.0 and higher)
- Microsoft Visual Basic for Windows (Version 2.0 and higher)

The manual is intended for application programmers using a RISCom-8 board in an IBM® PC/XT™, PC AT®, or compatible computer. It is assumed that users have read the *RISCom-8 User's Guide* to familiarize themselves with the board's features, and that they have completed the appropriate hardware setup and installation.

It is also assumed that users are experienced in programming in their selected language and that they are familiar with serial communication principles.

The *RISCom-8 Windows Function Call Driver User's Guide* is organized as follows:

- Chapter 1 contains the information needed to install the RISCom-8 Windows Function Call Driver, use the configuration program, and get help, if required.
- Chapter 2 provides background information about the functions included in the RISCom-8 Windows Function Call Driver.
- Chapter 3 describes how to create application programs in the supported languages.
- Chapter 4 contains detailed descriptions of the functions, arranged in alphabetical order.

An index completes this manual.

Table of Contents

List of Figures

List of Tables

1

Getting Started

The RISCom-8 Windows Function Call Driver is a Dynamic Link Library (DLL) of communication functions. This Function Call Driver allows you to access the RISCom-8 board from the following Windows-based languages:

- Microsoft QuickC
- Microsoft Visual C++
- Borland C++
- Turbo Pascal for Windows
- Microsoft Visual Basic for Windows.

Included in this software package are the following:

- Support files, containing program elements, such as function prototypes and definitions of variable types, that are required by the functions.
- Configuration program.
- Language-specific example programs that loop back data on the first port of the RISCom-8. Note that you must attach a loopback connector on the communication port of the RISCom-8 or use a standard RISCom-8 Octal cable with a loopback plug to read the characters.

The following sections describe how to install the Function Call Driver, how to use the configuration program, and how to get additional assistance, if required.

Quick Setup

To install the RISCom-8 Windows Function Call Driver, perform the following steps:

1. Install the RISCom-8 boards as described in the *RISCom-8 User's Guide*, making sure that all boards have a unique base address and interrupt setting.
2. Insert the RISCom-8 software diskette into a floppy disk drive of your computer.
3. Run Windows.
4. From the Program Manager File menu, select Run.
5. In the Command Line text box, type the letter of the drive containing your RISCom-8 diskette, then type SETUP.EXE. For example, if you diskette is in drive A, type the following:

   ```
   A:SETUP.EXE
   ```

6. Select OK.
7. Respond to the installation prompts, as necessary.
8. Run the CFGR8W.EXE configuration program, described in the next section, and answer the prompts as they appear to configure the driver from the Windows program group.

Once you complete this procedure, you can write Windows application programs that use the functions described in Chapters 2 and 4 to communicate with the RISCom-8 boards. Refer to Chapter 3 for language-specific programming information.

Running the Configuration Program

You can use multiple RISCom-8 boards in one computer. If you use four boards, the ports are accessed as ports 0 to 31.

Run the configuration program, CFGR8W.EXE, every time you add a new RISCom-8 board to your system. The program creates and/or edits a configuration file you specify. You can name the configuration file any name you like, such as RISCOM8.CFG.

Specify the following parameters in the configuration file:

- Number of RISCom-8 boards used
- Base address of each RISCom-8 board
- Interrupt (IRQ) setting of each RISCom-8 board
- Interface type (such as, RS-232, RS-422/485) of each RISCom-8 board

The following is an example of a configuration file:

```
[RISCOM8]
BOARD1=P 544, I 11, R RS232
BOARD2=P 576, I 10, R RS232
```

In this example, two RISCom-8, RS-232 boards are configured in the system. One board is set up with a base address of 220h (544 decimal) and an IRQ of 11; the other board is set up with a base address of 240h (576 decimal) and an IRQ of 10.

Getting Help

If you need help installing or using the RISCom-8 Windows Function Call Driver, call your local sales office or the Keithley MetraByte Applications Engineering Department at:

(508) 880-3000

Monday - Friday, 8:00 A.M. - 6:00 P.M., Eastern Time

An applications engineer will help you diagnose and resolve your problem over the telephone. Please make sure that you have the following information available before you call:

RISCom-8 board configuration	Model	_____
	Serial #	_____
	Revision code	_____
	Base address setting	_____
	Interrupt level setting	_____
Computer	Manufacturer	_____
	CPU type	_____
	Clock speed (MHz)	_____
	KB of RAM	_____
	Video system	_____
	BIOS type	_____
Operating system	Windows version	_____
	Windows mode	_____
Software package	Name	_____
	Serial #	_____
	Version	_____
	Invoice/Order #	_____
Compiler (if applicable)	Language	_____
	Manufacturer	_____
	Version	_____
Accessories	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____
	Type	_____

2

Available Operations

This chapter provides the background information you need to use the functions to perform communication operations on the RISCom-8 board. The supported operations are listed in Table 2-1.

Table 2-1. Supported Operations

Operation	Page Reference
Initialization	page 2-1
Data Movement	page 2-4
Port Control	page 2-5
Buffer Management	page 2-6

Initialization Operations

This section describes the functions provided in the RISCom-8 Windows Function Call Driver to perform the following initialization operations:

- Initializing the driver
- Initializing a communication port
- Resetting all communication ports
- Setting the communication protocol

Initializing the Driver

Before you can use any of the functions included in the RISCom-8 Windows Function Call Driver, you must initialize the driver using the **K_R8DevOpen** function.

K_R8DevOpen initializes the driver according to a configuration file you specify and returns an error/status code indicating whether the driver was initialized successfully. Refer to page 1-3 for more information about creating a configuration file for RISCom-8 boards; refer to page 4-5 for information on the error/status codes.

Initializing a Communication Port

If you use four RISCom-8 boards, the communication ports are accessed as ports 0 to 31. To initialize a specified communication port, you can use the following functions:

- **K_R8Init** - Initializes the specified communication port with a specified parity (none, even, or odd), number of stop bits (one or two), number of bits in the word to transfer (five to eight bits), and baud rate (110 to 76800). This function also returns the status of the communication port. The status consists of two bytes, as shown in Figure 2-1.

Low Byte, Port Status

Bit 7 = 1; Timed Out	Bit 6 = 1; TxSR Empty	Bit 5 = 1; TxHR Empty	Bit 4 = 1; Break On	Bit 3 = 1; Frame Error	Bit 2 = 1; Parity Error	Bit 1 = 1; Overrun Error	Bit 0 = 1; Data Ready

High Byte, Modem Status

Bit 7 = 1; RCD On	Bit 6 = 1; RI On	Bit 5 = 1; DSR On	Bit 4 = 1; CTS On	Bit 3 = 1; DCD Changed	Bit 2 = 1; RI Changed	Bit 1 = 1; DSR Changed	Bit 0 = 1; CTS Changed

Figure 2-1. Status of the Communication Port

- **K_R8SetParity** - Sets the parity for a specified communication port to either none, even, or odd. This function also returns the previously programmed parity value.
- **K_R8SetStopBits** - Sets the number of stop bits for a specified communication port to either one or two. This function also returns the previously programmed stop bits value.
- **K_R8SetWordLen** - Sets the number of bits in the word to transfer (the word length) for the specified communication port to either five, six, seven, or eight bits. This function also returns the previously programmed word length.
- **K_R8SetBaudRate** - Sets the baud rate for the specified communication port to either 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 76800. This function also returns the previously programmed baud rate.

Resetting all Communication Ports

To reset all communication ports on the RISCom-8 boards, use the **K_R8Reset** function. This function returns the number of RISCom-8 boards present.

Setting the Communication Protocol

To set a communication protocol for a specified communication port, you can use the following functions:

- **K_R8HandShakeProt** - Specifies whether CTS and RTS (hardware handshaking signals) are enabled or disabled for transmit operations.
- **K_R8XONProt** - Specifies whether XON is enabled or disabled for receive operations.
- **K_R8XONChar** - Specifies the ASCII character that enables the XON operation.
- **K_R8XOFFChar** - Specifies the ASCII character that enables the XOFF operation.

Data Movement Operations

This section describes the functions provided in the RISCom-8 Windows Function Call Driver to perform the following data movement operations:

- Reading data
- Writing data

Reading Data

To read data from a specified communication port, you can use the following functions:

- **K_R8ReceiveChar** - Returns one character. This function also returns the status of the communication port, shown in Figure 2-1.
- **K_R8ReadString** - Reads a specified number of characters from a string. This function also returns the number of characters read from the receive buffer..
- **K_R8GetStatus** - Returns the status of the communication port and the modem as two bytes, shown in Figure 2-1.

Writing Data

To write data to a specified communication port, you can use the following functions:

- **K_R8SendChar** - Writes one character. This function also returns the status of the communication port. Refer to Figure 2-1 for the format of the value returned.
- **K_R8WriteString** - Writes a specified number of characters in a string. This function also returns the actual number of characters written to the transmission buffer.

Port Control Operations

This section describes the functions provided in the RISCom-8 Windows Function Call Driver to perform the following operations:

- Controlling the receiver
- Controlling the transmitter
- Controlling the modem

Controlling the Receiver

To control the receiver of a specified communication port, you can use the following functions:

- **K_R8RXEnable** - Enables the receiver.
- **K_R8RXDisable** - Disables the receiver.

Controlling the Transmitter

To control the transmitter of a specified communication port, you can use the following functions:

- **K_R8TXEnable** - Enables the transmitter.
- **K_R8TXDisable** - Disables the transmitter.

Controlling the Modem

To control the modem on a specified communication port, you can use the following functions:

- **K_R8SetModSignals** - Sets the RTS and DTR signals on or off.
- **K_R8GetModSignals** - Returns the state of the RTS and DTR signals.

Buffer Management Operations

This section describes the functions provided in the RISCom-8 Windows Function Call Driver to perform the following buffer management functions:

- Managing the receive buffer
- Managing the transmission buffer

Managing the Receive Buffer

To manage the receive buffer of a specified communication port, you can use the following functions:

- **K_R8ClearRecBuf -** Clears the receive buffer.
- **K_R8GetNumRecChar** - Returns the number of characters in the receive buffer.

Managing the Transmission Buffer

To manage the transmission buffer of a specified communication port, you can use the following functions:

- **K_R8ClearTransBuf -** Clears the transmission buffer.
- **K_R8TransBufAvail** - Returns the space available in terms of the number of characters that can be sent to the transmission buffer before an overflow error occurs.

3

Programming with the Function Call Driver

This chapter describes how to create an application program in each of the supported languages using the RISCom-8 Windows Function Call Driver.

Programming in Microsoft QuickC for Windows

To program in Microsoft QuickC for Windows, you need the following files:

File	Description
RISCOM8.DLL	Dynamic Link Library.
RISCOM8.H	Include file.

To create an executable file in Microsoft QuickC for Windows, perform the following steps:

1. Load *filename*.c into the QuickC for Windows environment, where *filename* indicates the name of your application program.

2. Create a project file. The project file should contain all necessary files, including *filename*.c, *filename*.rc, *filename*.def, *filename*.h, and RISC8IMP.LIB, where *filename* indicates the name of your application program.

3. From the Project menu, choose Build to create a stand-alone executable file (.EXE) that you can execute from within Windows.

Programming in Microsoft Visual C++ for Windows

To program in Microsoft Visual C++, you need the following files:

File	Description
RISCOM8.DLL	Dynamic Link Library.
RISCOM8.H	Include file for C.
RISCOM8.HPP	Include file for C++.

To create an executable file in Visual C++, perform the following steps:

1. Create a project file by choosing New from the Project menu. The project file should contain all necessary files, including *filename*.c, *filename*.rc, *filename*.def, and RISC8IMP.LIB, where *filename* indicates the name of your application program.

2. From the Project menu, choose Rebuild All FILENAME.EXE to create a stand-alone executable file (.EXE) that you can execute from within Windows.

Programming in Borland C++ for Windows

To program in Borland C++, you need the following files:

File	Description
RISCOM8.DLL	Dynamic Link Library.
RISCOM8.H	Include file for C.
RISCOM8.HPP	Include file for C++.

To create an executable file in Borland C++, perform the following steps:

1. Create a project file by choosing Open from the Project menu and entering a project name. The project file should contain all necessary files, including *filename*.c, *filename*.rc, *filename*.def, and RISC8IMP.LIB, where *filename* indicates the name of your application program.

2. From the Compile menu, choose Build All FILENAME.EXE to create a stand-alone executable file (.EXE) that you can execute from within Windows.

Programming in Borland Turbo Pascal for Windows

To program in Borland Turbo Pascal for Windows, you need the following files:

File	Description
RISCOM8.DLL	Dynamic Link Library.
RISCOM8.INC	Include file.

To create an executable file in Borland Turbo Pascal for Windows, perform the following steps:

1. Load *filename*.pas into the Borland Turbo Pascal for Windows environment, where *filename* indicates the name of your application program.

2. From the Compile menu, choose Make.

Programming in Microsoft Visual Basic for Windows

To program in Microsoft Visual Basic for Windows, you need the following files:

File	Description
RISCOM8.DLL	Dynamic Link Library.
VBWDECL.BAS	Include file.

To create an executable file from the Microsoft Visual Basic for Windows environment, choose Make EXE File from the Run menu.

4

Function Call Reference

This chapter describes the functions supported by the RISCom-8 Windows Function Call Driver, in alphabetical order.

Table 4-1 lists the specific functions associated with each type of operation as well as the page number where the function is described.

Table 4-1. RISCom-8 Functions

Operation Type	Function	Page Number
Initialization	K_R8DevOpen	page 4-5
	K_R8Reset	page 4-21
	K_R8Init	page 4-15
	K_R8SetBaudRate	page 4-26
	K_R8SetStopBits	page 4-32
	K_R8SetWordLen	page 4-34
	K_R8SetParity	page 4-30
	K_R8XONProt	page 4-46
	K_R8XONChar	page 4-44
	K_R8XOFFChar	page 4-42
	K_R8HandShakeProt	page 4-13

Table 4-1. RISCom-8 Functions (cont.)

Operation Type	Function	Page Number
Data Movement	K_R8SendChar	page 4-24
	K_R8ReceiveChar	page 4-20
	K_R8GetStatus	page 4-11
	K_R8ReadString	page 4-18
	K_R8WriteString	page 4-40
Port Control	K_R8RXEnable	page 4-23
	K_R8RXDisable	page 4-22
	K_R8TXEnable	page 4-39
	K_R8TXDisable	page 4-38
	K_R8GetModSignals	page 4-7
	K_R8SetModSignals	page 4-28
Buffer Management	K_R8ClearRecBuf	page 4-3
	K_R8GetNumRecChar	page 4-9
	K_R8ClearTransBuf	page 4-4
	K_R8TransBufAvail	page 4-36

Purpose

Clears the receive buffer of the specified communication port.

Prototype

C/C++

void pascal far K_R8ClearRecBuf (int *ComNum*);

Turbo Pascal

procedure K_R8ClearRecBuf (*ComNum* : integer); far;
external 'riscom8';

Visual Basic

Declare Sub K_R8ClearRecBuf Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer)

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

None

See Also

K_R8ClearTransBuf

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
...
K_R8ClearRecBuf (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
...
K_R8ClearRecBuf (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
...
K_R8ClearRecBuf (0)
```

K_R8ClearTransBuf

Purpose

Clears the transmission buffer of the specified communication port.

Prototype

C/C++

void pascal far K_R8ClearTransBuf (int *ComNum*);

Turbo Pascal

procedure K_R8ClearTransBuf (*ComNum* : integer); far;
external 'riscom8';

Visual Basic

Declare Sub K_R8ClearTransBuf Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer)

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

None

See Also

K_R8ClearRecBuf

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
...
K_R8ClearTransBuf (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
...
K_R8ClearTransBuf (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
...
K_R8ClearTransBuf (0)
```

Purpose

Opens the specified configuration file, reads the data in the file, and performs initialization tests.

Your program must call this function before calling any other function in this library.

Prototype

C/C++

unsigned int pascal far K_R8DevOpen (char far **CfgFileName*);

Turbo Pascal

function K_R8DevOpen (var *CfgFileName* : char) : word; far;
external 'riscom8';

Visual Basic

Declare Function K_R8DevOpen Lib "RISCOM8.DLL"
(ByVal *CfgFileName* As String) As Integer

Parameters

CfgFileName — Configuration file name.
Valid values: The name of a configuration file.

Return Value

K_R8DevOpen returns one of the following error/status codes:

Error/Status Code	Description
0	No error
1	General error
2	Cannot open file or file not found
3	Invalid file format
4	Base address error
5	Interrupt level error
6	RS type error
7	Board not found at the specified address

K_R8DevOpen (cont.)

See Also K_R8Init

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int R_Err;
...
R_Err = K_R8DevOpen ("RISCOM8.CFG");
```

Turbo Pascal

```
{$I RISCOM8.INC}
szCfgName : string;
R_Err : word;
...
szCfgName := 'RISCOM8.CFG' + #0;
...
R_Err := K_R8DevOpen (szCfgName[1]);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM R_Err AS INTEGER
...
R_Err = K_R8DevOpen ("riscom8.cfg")
```

Purpose

Returns the status of the modem signals from the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8GetModSignals (int *ComNum*);

Turbo Pascal

procedure K_R8GetModSignals (*ComNum* : integer); far;
external 'riscom8';

Visual Basic

Declare Function K_R8GetModSignals Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer) As Integer

Parameters

ComNum — Communication port.
Valid values: **0** to **31**

Return Value

K_R8GetModSignals returns the status of the modem signals. Values are as follows:

Bit	Value	Description
0	0	RTS Off
	1	RTS On
1	0	DTR Off
	1	DTR On

See Also

K_R8SetModSignals

K_R8GetModSignals (cont.)

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int ModSignals;
...
ModSignals = K_R8GetModSignals (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
ModSignals : Integer;
...
ModSignals := K_R8GetModSignals (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM ModSignals AS INTEGER
...
ModSignals = K_R8GetModSignals (0)
```

K_R8GetNumRecChar

Purpose

Returns the number of characters in the receive buffer of the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8GetNumRecChar (int *ComNum*);

Turbo Pascal

function K_R8GetNumRecChar (*ComNum* : integer) : word; far;
external 'riscom8';

Visual Basic

Declare Function K_R8GetNumRecChar Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

K_R8GetNumRecChar returns the number of characters currently in the receive buffer.

See Also

K_R8ClearRecBuf

K_R8GetNumRecChar (cont.)

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int NumofChar;
...
NumofChar = K_R8GetNumRecChar (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
NumofChar : word;
...
NumofChar := K_R8GetNumRecChar (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM NumofChar AS INTEGER
...
NumofChar = K_R8GetNumRecChar (0)
```

Purpose

Returns the status of the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8GetStatus (int *ComNum*);

Turbo Pascal

function K_R8GetStatus (*ComNum* : integer) : word; far;
external 'riscom8';

Visual Basic

Declare Function K_R8GetStatus Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

K_R8GetStatus returns the status of the communication port. The status consists of two bytes, as shown in Figure 4-1.

Low Byte, Port Status

Bit 7 = 1; Timed Out	Bit 6 = 1; TxSR Empty	Bit 5 = 1; TxHR Empty	Bit 4 = 1; Break On	Bit 3 = 1; Frame Error	Bit 2 = 1; Parity Error	Bit 1 = 1; Overrun Error	Bit 0 = 1; Data Ready

High Byte, Modem Status

Bit 7 = 1; RCD On	Bit 6 = 1; RI On	Bit 5 = 1; DSR On	Bit 4 = 1; CTS On	Bit 3 = 1; DCD Changed	Bit 2 = 1; RI Changed	Bit 1 = 1; DSR Changed	Bit 0 = 1; CTS Changed

Figure 4-1. Value Returned by K_R8GetStatus

K_R8GetStatus (cont.)

See Also K_R8Init, K_R8SendChar

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int Status;
...
Status = K_R8GetStatus (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
Status : word;
...
Status := K_R8GetStatus (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM Status AS INTEGER
...
Status = K_R8GetStatus (0)
```

Purpose

Specifies whether hardware handshaking is enabled or disabled for transmit operations.

Prototype

C/C++

unsigned int pascal far K_R8HandShakeProt (int *ComNum*, int *ProtoCall*);

Turbo Pascal

function K_R8HandShakeProt (*ComNum* : integer; *ProtoCall* : integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8HandShakeProt Lib "RISCOM8.DLL" (ByVal *ComNum* As Integer, ByVal *ProtoCall* As Integer) As Integer

Parameters

ComNum — Communication port.
Valid values: **0** to **31**

ProtoCall — Status of hardware handshaking.
Valid values:

Bit	Value	Description
0	0	CTS disabled
	1	CTS enabled
1	0	RTS disabled
	1	RTS enabled

Return Value

K_R8HandShakeProt returns the previously programmed protocol. See the description of *ProtoCall* above for more information.

See Also

K_R8XONProt

K_R8HandShakeProt (cont.)

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldProtoCall;
...
OldProtoCall = K_R8HandShakeProt (0, 2);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldProtoCall : word;
...
OldProtoCall := K_R8HandShakeProt (0, 2);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldProtoCall AS INTEGER
...
OldProtoCall = K_R8HandShakeProt (0, 2)
```

Purpose

Initializes the specified communication port with the specified parity, number of stop bits, number of bits in the word, and baud rate.

Prototype

C/C++

unsigned int pascal far K_R8Init (int *ComNum*, int *Parity*, int *StopBits*, int *WordLen*, int *BaudRate*);

Turbo Pascal

function K_R8Init (*ComNum* : integer; *Parity* : integer; *StopBits* : integer; *WordLen* : integer; *BaudRate* : integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8Init Lib "RISCOM8.DLL" (ByVal *ComNum* As Integer, ByVal *Parity* As Integer, ByVal *StopBits* As Integer, ByVal *WordLen* As Integer, ByVal *BaudRate* As Integer) As Integer

Parameters

ComNum
Communication port.
Valid values: **0** to **31**

Parity
Type of parity checking.
Valid values: **0** = No Parity
1 = Even Parity
2 = Odd Parity

StopBits
Number of stop bits.
Valid values: **0** = one stop bit
1 = two stop bits

WordLen
Number of bits in the word.
Valid values: **0** = five bits
1 = six bits
2 = seven bits
3 = eight bits

BaudRate Baud rate.
Valid values:

Code	Baud Rate	Code	Baud Rate
0	110	6	4800
1	150	7	9600
2	300	8	19200
3	600	9	38400
4	1200	10	57600
5	2400	11	76800

Return Value K_R8Init returns the previously programmed status of the communication port. The status consists of two bytes, as shown in Figure 4-2.

Low Byte, Port Status

Bit 7 = 1; Timed Out	Bit 6 = 1; TxSR Empty	Bit 5 = 1; TxHR Empty	Bit 4 = 1; Break On	Bit 3 = 1; Frame Error	Bit 2 = 1; Parity Error	Bit 1 = 1; Overrun Error	Bit 0 = 1; Data Ready

High Byte, Modem Status

Bit 7 = 1; RCD On	Bit 6 = 1; RI On	Bit 5 = 1; DSR On	Bit 4 = 1; CTS On	Bit 3 = 1; DCD Changed	Bit 2 = 1; RI Changed	Bit 1 = 1; DSR Changed	Bit 0 = 1; CTS Changed

Figure 4-2. Value Returned by K_R8Init

See Also

K_R8SetParity, K_R8SetStopBits, K_R8SetWordLen, K_R8SetBaudRate

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldSettings;
...
OldSettings = K_R8Init (0, 0, 0, 3, 7);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldSettings : word;
...
OldSettings := K_R8Init (0, 0, 0, 3, 7);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldSettings AS INTEGER
...
OldSettings = K_R8Init (0, 0, 0, 3, 7)
```

K_R8ReadString

Purpose

Reads the characters in a string from the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8ReadString (int *ComNum*, char far **String*, int *NumOfChar*);

Turbo Pascal

function K_R8ReadString (*ComNum* : integer; Var *RdString* : char; *NumOfChar* : integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8ReadString Lib "RISCOM8.DLL" (ByVal *ComNum* As Integer, ByVal *StringBuffer* As String, ByVal *NumOfChar* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
String	String to read.
NumOfChar	Number of characters in the string to read.

Return Value

K_R8ReadString returns the actual number of characters read from the receive buffer.

See Also

K_R8WriteString

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int NumofChar;
char Data[128];
...
NumofChar = K_R8ReadString (0, Data, 10);
```

Turbo Pascal

```
{$I RISCOM8.INC}
szdata : array[0..79] of char;
NumofChar : integer;
...
NumofChar := K_R8ReadString (0, szdata[0] ,10);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM szdata AS STRING
DIM NumofChar AS INTEGER
...
NumofChar = K_R8ReadString (0, szdata, 10)
```

K_R8ReceiveChar

Purpose

Returns one character from the specified communication port.

Prototype

C/C++

char pascal far K_R8ReceiveChar (int *ComNum*);

Turbo Pascal

function K_R8ReceiveChar (*ComNum* : integer) : char; far;
external 'riscom8';

Visual Basic

Declare Function K_R8ReceiveChar Lib "RISCOM8.DLL" (ByVal *ComNum* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

K_R8ReceiveChar returns the status of the communication port and the character read. If the upper byte is 0, then the lower byte is the character received. If the upper byte is nonzero, no character was received.

See Also

K_R8SendChar

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
char CharToRec;
...
CharToRec = K_R8ReceiveChar (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
CharToRec : char;
...
CharToRec := K_R8ReceiveChar (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM CharToRec AS STRING
CharToRec(0) = K_R8ReceiveChar (0)
```

Purpose

Initializes all the communication ports that are available.

Prototype

C/C++

```
unsigned int pascal far K_R8Reset ();
```

Turbo Pascal

```
function K_R8Reset : word; far; external 'riscom8';
```

Visual Basic

```
Declare Function K_R8Reset Lib "RISCOM8.DLL" () As Integer
```

Parameters

None

Return Value

K_R8Reset returns the number of boards present.

See Also

K_R8Init

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int NumofBds;
...
NumofBds = K_R8Reset ();
```

Turbo Pascal

```
{$I RISCOM8.INC}
NumofBds : word;
...
NumofBds := K_R8Reset ();
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM NumofBds AS INTEGER
...
NumofBds = K_R8Reset ()
```

K_R8RXDisable

Purpose

Disables the receiver.

Prototype

C/C++

void pascal far K_R8RXDisable (int *ComNum*);

Turbo Pascal

procedure K_R8RXDisable (*ComNum* : integer); far; external 'riscom8';

Visual Basic

Declare Sub K_R8RXDisable Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer)

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

None

See Also

K_R8RXEnable

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
...
K_R8RXDisable (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
...
K_R8RXDisable (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
...
K_R8RXDisable (0)
```

Purpose

Enables the receiver.

Prototype

C/C++

void pascal far K_R8RXEnable (int *ComNum*);

Turbo Pascal

procedure K_R8RXEnable (*ComNum* : integer); far; external 'riscom8';

Visual Basic

Declare Sub K_R8RXEnable Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer)

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

None

See Also

K_R8RXDisable

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
...
K_R8RXEnable (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
...
K_R8RXEnable (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
...
K_R8RXEnable (0)
```

K_R8SendChar

Purpose

Writes a specified character to the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8SendChar (int *ComNum*, char *CharToSend*);

Turbo Pascal

function K_R8SendChar (*ComNum* : integer; *CharToSend* : char) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8SendChar Lib "RISCOM8.DLL" Alias "K_R8SendBasicChar" (ByVal *ComNum* As Integer, ByVal *CharToSend* As String) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
CharToSend	Character to send.

Return Value

K_R8SendChar returns the status of the communication port. The status consists of two bytes, as shown in Figure 4-3.

Low Byte, Port Status

Bit 7 = 1;	Bit 6 = 1;	Bit 5 = 1;	Bit 4 = 1;	Bit 3 = 1;	Bit 2 = 1;	Bit 1 = 1;	Bit 0 = 1;
Timed Out	TxSR Empty	TxHR Empty	Break On	Frame Error	Parity Error	Overrun Error	Data Ready

High Byte, Modem Status

Bit 7 = 1;	Bit 6 = 1;	Bit 5 = 1;	Bit 4 = 1;	Bit 3 = 1;	Bit 2 = 1;	Bit 1 = 1;	Bit 0 = 1;
RCD On	RI On	DSR On	CTS On	DCD Changed	RI Changed	DSR Changed	CTS Changed

Figure 4-3. Value Returned by K_R8SendChar

See Also K_R8ReceiveChar

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int status;
char CharToSend;
...
status = K_R8SendChar (0, CharToSend);
```

Turbo Pascal

```
{$I RISCOM8.INC}
status : word;
CharToSend : char;
...
status := K_R8SendChar (0, CharToSend);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM status AS INTEGER
DIM CharToSend AS STRING
...
status = K_R8SendChar (0, CharToSend(0))
```

K_R8SetBaudRate

Purpose

Sets the baud rate for the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8SetBaudRate (int *ComNum*, int *BaudRate*);

Turbo Pascal

function K_R8SetBaudRate (*ComNum* : integer;
BaudRate : integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8SetBaudRate Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *BaudRate* As Integer) As Integer

Parameters

ComNum — Communication port.
Valid values: **0** to **31**

BaudRate — Baud rate.
Valid values:

Code	Baud Rate	Code	Baud Rate
0	110	6	4800
1	150	7	9600
2	300	8	19200
3	600	9	38400
4	1200	10	57600
5	2400	11	76800

Return Value

K_R8SetBaudRate returns the previously programmed baud rate.

See Also

K_R8Init

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldRate;
...
OldRate = K_R8SetBaudRate (0, 7);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldRate : word;
...
OldRate := K_R8SetBaudRate (0, 7);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldRate AS INTEGER
...
OldRate = K_R8SetBaudRate (0, 7)
```

K_R8SetModSignals

Purpose

Sets the state of the modem signals on the specified communication port.

Prototype

C/C++

void pascal far K_R8SetModSignals (int *ComNum*, int *ModemSignals*);

Turbo Pascal

procedure K_R8SetModSignals (*ComNum* : integer;
ModemSignals : integer); far; external 'riscom8';

Visual Basic

Declare Sub K_R8SetModSignals Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *ModemSignals* As Integer)

Parameters

ComNum — Communication port.
Valid values: **0** to **31**

ModemSignals — State of the modem signals.
Valid values:

Bit	Value	Description
0	0	RTS Off
	1	RTS On
1	0	DTR Off
	1	DTR On

Return Value

None

See Also

K_R8GetModSignals

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
...
K_R8SetModSignals (0, 2);
```

Turbo Pascal

```
{$I RISCOM8.INC}
...
K_R8SetModSignals (0, 2);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
...
K_R8SetModSignals (0, 2)
```

K_R8SetParity

Purpose

Sets the parity for the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8SetParity (int *ComNum*, int *Parity*);

Turbo Pascal

function K_R8SetParity (*ComNum* : integer; *Parity* : integer) : word; far;
external 'riscom8';

Visual Basic

Declare Function K_R8SetParity Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *Parity* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
Parity	Type of parity checking. Valid values: **0** = No Parity **1** = Even Parity **2** = Odd Parity

Return Value

K_R8SetParity returns the previously programmed parity value.

See Also

K_R8Init

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldParity;
...
OldParity = K_R8SetParity (0, 0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldParity : word;
...
OldParity := K_R8SetParity (0, 0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldParity AS INTEGER
...
OldParity = K_R8SetParity (0, 0)
```

K_R8SetStopBits

Purpose

Sets the number of stop bits for the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8SetStopBits (int *ComNum*, int *StopBits*);

Turbo Pascal

function K_R8SetStopBits (*ComNum* : integer; *StopBits* : integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8SetStopBits Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *StopBits* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
StopBits	Number of stop bits. Valid values: **0** = one stop bit **1** = two stop bits

Return Value

K_R8SetStopBits returns the previously programmed number of stop bits.

See Also

K_R8Init

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldStopBits;
...
OldStopBits = K_R8SetStopBits (0, 1);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldStopBits : word;
...
OldStopBits := K_R8SetStopBits (0, 1);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldStopBits AS INTEGER
...
OldStopBits = K_R8SetStopBits (0, 1)
```

K_R8SetWordLen

Purpose

Sets the number of bits in the word to transfer for the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8SetWordLen (int *ComNum*, int *WordLen*);

Turbo Pascal

function K_R8SetWordLen (*ComNum* : integer;
WordLen : integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8SetWordLen Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *WordLen* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
WordLen	Number of bits in the word Valid values: **0** = five bits **1** = six bits **2** = seven bits **3** = eight bits

Return Value

K_R8SetWordLen returns the previously programmed word length.

See Also

K_R8Init

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldLen;
...
OldLen = K_R8SetWordLen (0, 3);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldLen : word;
...
OldLen := K_R8SetWordLen (0, 3);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldLen AS INTEGER
...
OldLen = K_R8SetWordLen (0, 3)
```

K_R8TransBufAvail

Purpose

Returns the number of characters that are available in the transmission buffer of the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8TransBufAvail (int *ComNum*);

Turbo Pascal

procedure K_R8TransBufAvail (*ComNum* : integer); far;
external 'riscom8';

Visual Basic

Declare Function K_R8TransBufAvail Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

K_R8TransBufAvail returns the remaining number of characters available in the transmission buffer (before an overflow occurs).

See Also

K_R8ClearTransBuf

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int CharLeft;
...
CharLeft = K_R8ClearTransBuf (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
CharLeft : Integer;
...
CharLeft := K_R8ClearTransBuf (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM CharLeft AS INTEGER
...
CharLeft = K_R8ClearTransBuf (0)
```

K_R8TXDisable

Purpose

Disables the transmitter.

Prototype

C/C++

void pascal far K_R8TXDisable (int *ComNum*);

Turbo Pascal

procedure K_R8TXDisable (*ComNum* : integer); far; external 'riscom8';

Visual Basic

Declare Sub K_R8TXDisable Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer)

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

None

See Also

K_R8TXEnable

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
...
K_R8TXDisable (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
...
K_R8TXDisable (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
...
K_R8TXDisable (0)
```

Purpose

Enables the transmitter.

Prototype

C/C++

void pascal far K_R8TXEnable (int *ComNum*);

Turbo Pascal

procedure K_R8TXEnable (*ComNum* : integer); far; external 'riscom8';

Visual Basic

Declare Sub K_R8TXEnable Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer)

Parameters

ComNum	Communication port. Valid values: **0** to **31**

Return Value

None

See Also

K_R8TXDisable

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
...
K_R8TXEnable (0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
...
K_R8TXEnable (0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
...
K_R8TXEnable (0)
```

K_R8WriteString

Purpose

Writes the characters in a string to the specified communication port.

Prototype

C/C++

unsigned int pascal far K_R8WriteString (int *ComNum*,
char far **String*, int *NumOfChar*);

Turbo Pascal

function K_R8WriteString (*ComNum* : integer; var *WrtString*: char;
NumOfChar: integer); far; external 'riscom8';

Visual Basic

Declare Function K_R8WriteString Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *StringToOutput* As String,
ByVal *NumOfChar* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
String	String to write.
NumOfChar	Number of characters in the string to write.

Return Value

K_R8WriteString returns the actual number of characters written to the transmission buffer.

See Also

K_R8ReadString

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int NumofChar;
char Data[80];
...
NumofChar = K_R8WriteString (0, Data, 10);
```

Turbo Pascal

```
{$I RISCOM8.INC}
NumofChar : Integer;
szdata : array[0..79] of char;
...
NumofChar := K_R8WriteString (0, szdata[0], 10);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM NumofChar AS INTEGER
DIM szdata AS STRING
...
NumofChar = K_R8WriteString (0, szdata, 10)
```

K_R8XOFFChar

Purpose

Specifies the character that enables the XOFF operation.

Prototype

C/C++

unsigned int pascal far K_R8XOFFChar (int *ComNum*, int *XOFFChar*);

Turbo Pascal

function K_R8XOFFChar (*ComNum* : integer;
XOFFChar: integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8XOFFChar Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *XOFFChar* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
XOFFChar	XOFF character.

Return Value

K_R8XOFFChar returns the previously programmed protocol.

See Also

K_R8XONChar

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldProtoCall;
...
OldProtoCall = K_R8XOFFChar (0, 1);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldProtoCall : word;
...
OldProtoCall:= K_R8XOFFChar (0, 1);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldProtoCall AS INTEGER
...
OldProtoCall:= K_R8XOFFChar (0, 1)
```

K_R8XONChar

Purpose

Specifies the character that enables the XON operation.

Prototype

C/C++

unsigned int pascal far K_R8XONChar (int *ComNum*, int *XONChar*);

Turbo Pascal

function K_R8XONChar (*ComNum* : integer; *XONChar*: integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8XONChar Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *XONChar* As Integer) As Integer

Parameters

ComNum	Communication port. Valid values: **0** to **31**
XONChar	XON character.

Return Value

K_R8XONChar returns the previously programmed protocol.

See Also

K_R8XOFFChar

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldProtoCall;
...
OldProtoCall = K_R8XONChar  (0, 0);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldProtoCall : word;
...
OldProtoCall:= K_R8XONChar  (0, 0);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldProtoCall AS INTEGER
...
OldProtoCall:= K_R8XONChar (0, 0)
```

K_R8XONProt

Purpose

Specifies whether XON is enable or disabled for receive operation.

Prototype

C/C++

unsigned int pascal far K_R8XONProt (int *ComNum*, int *ProtoCall*);

Turbo Pascal

function K_R8XONProt (*ComNum* : integer; *ProtoCall*: integer) : word; far; external 'riscom8';

Visual Basic

Declare Function K_R8XONProt Lib "RISCOM8.DLL"
(ByVal *ComNum* As Integer, ByVal *ProtoCall* As Integer) As Integer

Parameters

ComNum — Communication port.
Valid values: **0** to **31**

ProtoCall — Status of XON.
Valid values:

Bit	Value	Description
0	0	Tx disabled
	1	Tx enabled
1	0	Rx disabled
	1	Rx enabled

Return Value

K_R8XONProt returns the previously programmed protocol. See the description of *ProtoCall* above for more information.

See Also

K_R8HandShakeProt

Usage

C/C++

```
#include "RISCOM8.H"   // Use "RISCOM8.HPP for C++
unsigned int OldProtoCall;
...
OldProtoCall = K_R8XONProt (0, 2);
```

Turbo Pascal

```
{$I RISCOM8.INC}
OldProtoCall : word;
...
OldProtoCall:= K_R8XONProt (0, 2);
```

Visual Basic

(Include VBWDECL.BAS in your program make file)

```
DIM OldProtoCall AS INTEGER
...
OldProtoCall:= K_R8XONProt (0, 2)
```

Index

F

H

I

K

L

M

O

P

Q

R

S

T

V

W

X