User Manual

UbiQ

Scenario Manager

User Manual V1.03

Trusted ePlatform Services

ADVANTECH

Copyright Notice

Acknowledgements

IBM and PC are trademarks of International Business Machines Corporation.

Intel is a trademark of Intel Corporation.

MS-DOS and Windows are trademarks of Microsoft Corporation.

ActiveX, Visual Basic, Excel, Access and Visual C++ are trademarks of Microsoft Corporation.

All other product names or trademarks are the properties of their respective owners.

Advantech Customer Services

Each and every Advantech product is built to the most exacting specifications to ensure reliable performance in the harsh and demanding conditions typical of industrial environments. Whether your new Advantech equipment is destined for the laboratory or the factory floor, you can be assured that your product will provide the reliability and ease of operation for which the name Advantech has come to be known. Your satisfaction is our primary concern. Here is a guide to Advantech's customer services. To ensure you get the full benefit of our services, please follow the instructions below carefully.

Technical Support

We want you to get the maximum performance from your products. So if you run into technical difficulties, we are here to help. For the most frequently asked questions, you can easily find answers in your product documentation. These answers are normally a lot more detailed than the ones we can give over the phone.

So please consult this manual first. If you still cannot find the answer, gather all the information or questions that apply to your problem, and with the product close at hand, call your dealer. Our dealers are well trained and ready to give you the support you need to get the most from your Advantech products. In fact, most problems reported are minor and are able to be easily solved over the phone.

In addition, free technical support is available from Advantech engineers every business day. We are always ready to give advice on application requirements or specific information on the installation and operation of any of our products.

Limited Warranty

Advantech Corporation does not warrant that the UbiQ Scenario Manager software utility package will function properly in every hardware/software environment. Advantech Corporation makes no representation or warranties of any kinds whatsoever with respect to the contents of this manual and specifically disclaims any implied warranties or fitness for any particular purpose. Advantech Corporation shall not be held liable for errors in this manual or for incidental or consequential damages in connection with the use of this manual or its contents. Advantech Corporation reserves the right to revise this manual at any time without prior notice.

About This Manual

Advantech UbiQ Scenario Manager software is a comprehensive, flexible human machine interface application environment, that supports the functions and utilities to develop all types of automation applications in the Windows XP, and Windows CE environment. UbiQ Scenario Manager software provides an windows-based, mouse driven system for designing Web-Enabled Graphic-Interactive System.

Organization of this Manual

- Chapter 1, Introduction gives a general background to the UbiQ platform. The system architecture is explained, and the product's main features are introduced. Installation of the software is explained.
- Chapter 2, Getting Start explains how to use UbiQ platform and complete some of the most common tasks within UbiQ Scenario Manager software package.
- Chapter 3, Tutorial explains how to complete the basic skills that you need to use the UbiQ platform. In addition, there are step-by-step tutorials that explain how to complete simple web-enabled system within the UbiQ platform.
- Chapter 4, Basics of Smart-C Script Language explains the basics of the Smart-C script language. In this chapter, you could learn the statements and syntax of C languages.
- Chapter 5, Functions Reference explains the UbiQ platform supporting functions list, you can call these functions in your programs or CGI scripts.

Contents

Chapter 1

Introduction

1.1 Overviews

Congratulations on your purchase of Advantech's UbiQ-230 product for developing scenario control for web-enabled solutions of home automation. Advantech UbiQ-230 product is a comprehensive, flexible web-enabled controlling platform that supports the functions and utilities to develop all types of automation applications on e-home in the Windows XP, and Windows CE environment. UbiQ-230 products provide scenario web-based systems including Web server, remote controlling & accessing functions and CGI-script to let your devices to be connected to Internet/Intranet.

UbiQ-230 is extremely flexible and easy to use. The client could use Internet Browser to get/set the information of devices. A list of UbiQ devices will be automatically probed and shown on your UbiQ Scenario Manager utility screen. Controllers°Ø and devices' functions and arguments are provided to you through the controller°Øs list. You can simply add/edit/delete functions for each controller to perform specific user defined functions; you can arrange those functions as a strategy, to manipulate the control programs for each device. You can also modify each controller's network configurations and port specifications via UbiQ Scenario Manager utility which runs on any control units. In addition to be easy to use, UbiQ-230 is built-in C interpreter to strengthen the ability to design complex calculation or analysis.

The UbiQ-230 kernel is a multi-threaded engine for optimal performance. It provides you plug-and-play connectivity with your devices, including lighting controllers, digital in/out controllers, and other automated devices. It saves a lot of effort and time when developing your applications. The UbiQ platform ensures that you can integrate your process data into existing e-Home information systems throughout your home.

In addition, UbiQ Scenario Manager software utility leverages 32-bit Windows°Ø pre-emptive multi-tasking capability to support Windows XP environments.

1.1.1 Contents

- System architecture
- Installation
- How does UbiQ-230 platform work?

1.2 System Architecture

We designed UbiQ-230 platform with a scenario Web-based

and open integrated architectures. The open platform is simple to configure and allows you to easily integrate automation system with other controllers and devices in your environment.

1.2.1 Module Description

1.2.1.1 Compact Embedded Web Server

Web Server is built-in on UbiQ-230 products. You could connect UbiQ-230 products by Internet Browser, and also get/set the connected devices' information.

1.2.1.2 C Script Engine

C Script Engine is the interpreter of C-language. The UbiQ-230 runs scenarios controlling programs after power-up. Controlling programs are interpreted as C by C Script Engine. In addition, CGI(Common Gateway Interface) scripts, are the interface of the web-server and connected devices, also are interpreted as C by C Script Engine.

1.2.1.3 UbiQ Scenario Manager Software Utility

UbiQ Scenario Manager utility is used remotely to configure and control UbiQ platform. When using UbiQ-230 in the first time, you must configure the name and IP address for the UbiQ device. And you need to add the controllers/devices connected UbiQ, if these controllers/devices are not in the default list. Then you could add these controllers/devices to UbiQ and test them by the UbiQ Scenario Manager utility.

1.3 Installation

Web server and C script engine is built-in on UbiQ platform. So you don't install anything in the UbiQ product.

UbiQ Remote Scenario utility is the client for configuring and controlling UbiQ devices remotely, you must install it into a PC system.

1.3.1 PC System Requirements

- OS : Microsoft Windows XP
- RAM : at least 128 MB memory
- Disk space: at least 4 MB space
- CPU: Intel Pentium II processor 400 MHz or higher
- Display: VGA resolution or higher
- Microsoft-compatible mouse
- Ethernet port

1.3.2 Installing UbiQ Scenario Manager Utility

UbiQ Scenario Manager utility ships with an installation program that helps you install the program to your computer.

Installation can normally be completed within two minutes.

1. Run the UbiQ Scenario Manager utility installation program at:
   d:\Scenario\setup.exe
   where "d" is the drive letter of your CD-ROM drive.
2. The *Welcome* screen loads. Click the **Next** button.

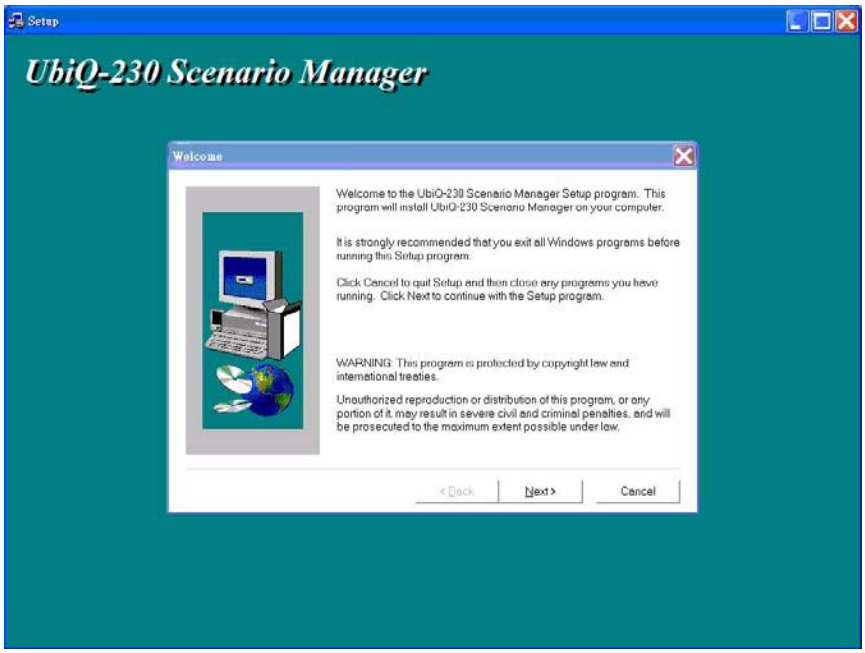


Figure 1.1 UbiQ Scenario Manager Installation Welcome Screen

3. Select a location where you want to install the UbiQ Scenario Manager utility in the Choose Destination Location window. The default location is:
   C:\Program Files\Advantech\UbiQ-230 Scenario manager

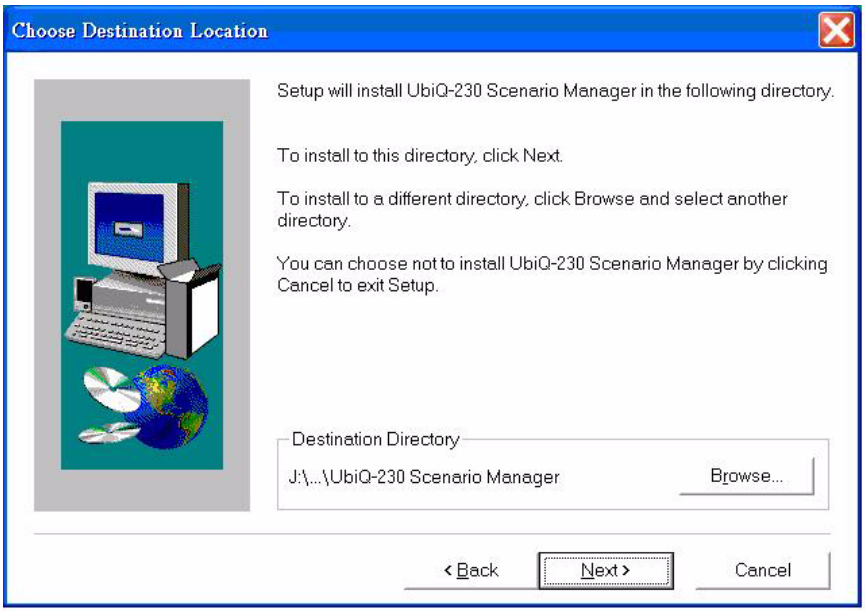


Figure 1.2 Choose Destination Location Screen

4. The installation program will create program shortcuts on your Windows Start menu so that you can easily launch the program. The default program folder is:
   UbiQ-230 Scenario Manager
   If you want to have the shortcuts made in a different folder, type it or select it in the *Select Program Folder dialog box*.

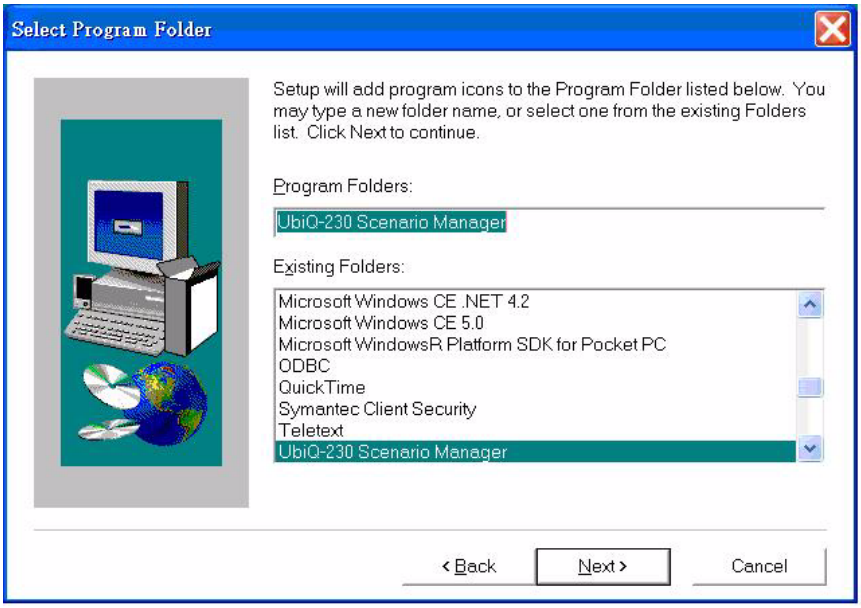


Figure 1.3 Select Program Folder Screen

5. UbiQ Scenario Manager utility is now installed on the computer; you can start using the program.

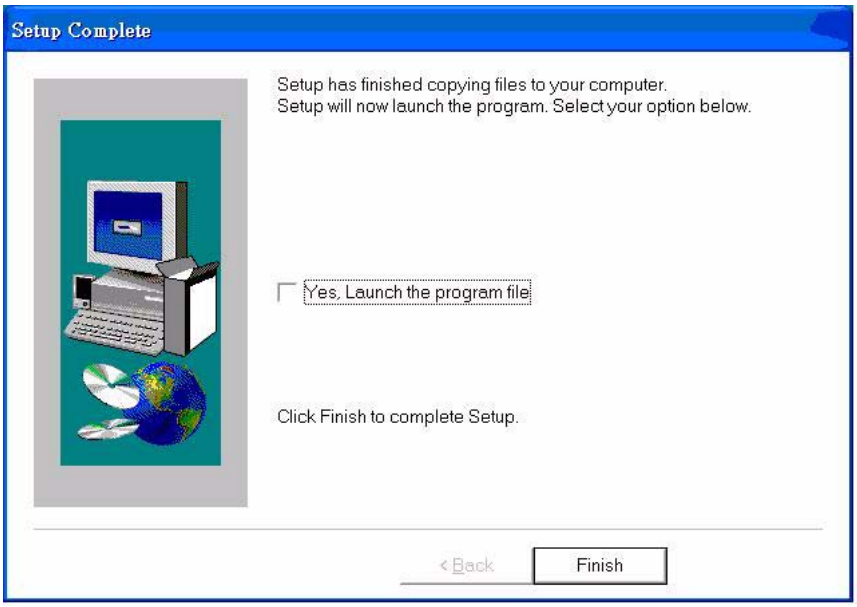


Figure 1.4 Setup Complete Screen

Uninstalling UbiQ Scenario Manager

1. Select Settings | Control Panel from the Windows Start menu and then double-click the the *Add/Remove Programs icon.*
2. Select the item *UbiQ Scenario Manager* utility and then click the Add/Remove... button.
3. Click the Yes button in the *Confirm File Deletion dialog box*.
4. The un-installation program removes the program files and registry entries from your computer. Click the OK button when the un-installation program finishes.

1.3.3 Start Menu Shortcuts

The UbiQ Scenario Manager utility installation program creates the following program shortcuts on the computer's Start menu.

The links are the following:

- *UbiQ-230 Scenario Manager* : The program folder.
- *Scenario* : The UbiQ Scenario Manager utility shortcut.

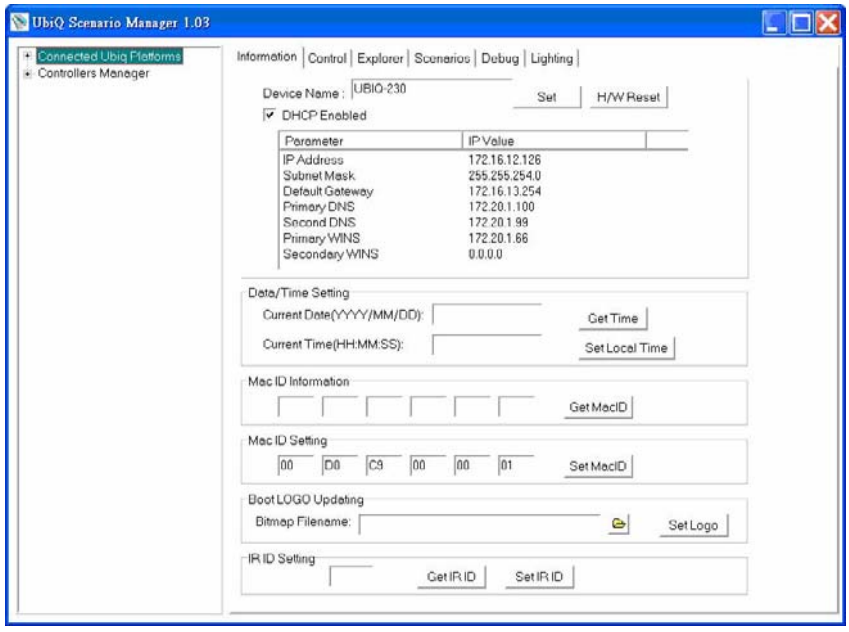
1.4 How Does UbiQ Work?

UbiQ platform includes two parts: controlling server and remote utility. The controlling server is the hardware body plus web server, C Script Engine, and scenarios program softwares. UbiQ device has one Ethernet port and 1 RS485 port. You can connect PC-based controllers/devices, and lighting controllers to UbiQ device by these ports. If data of the connected devices are periodic, you could put the controlling programs on UbiQ device to collect these data. You also could get these data or control these devices by Internet Browser anywhere anytime. Via CGI programs, you even setup connected devices to control their behaviors remotely.

UbiQ platform is designed to fit the following purposes:

- Web-based automation system
- Data acquisition provider and access controllers through Internet Browser
- Enable legacy controllers to Internet
- Provide remote accessing control
- Easy to configure connected controllers
- Simple to develop homepages combined with controller°Øs data
- Offer data to other applications

The UbiQ Scenario Manager utility is the client to configure and control the UbiQ platform. It could configure multi-UbiQ devices and run controlling programs to control connected devices.



Chapter 2

Getting Started

2.1 Quick Start to UbiQ-230 platform

As a quick introduction to using UbiQ-230 platform, complete the following procedure to run UbiQ-230 and scenario utility that was copied to your computer°Øs hard disk drive during the software installation.

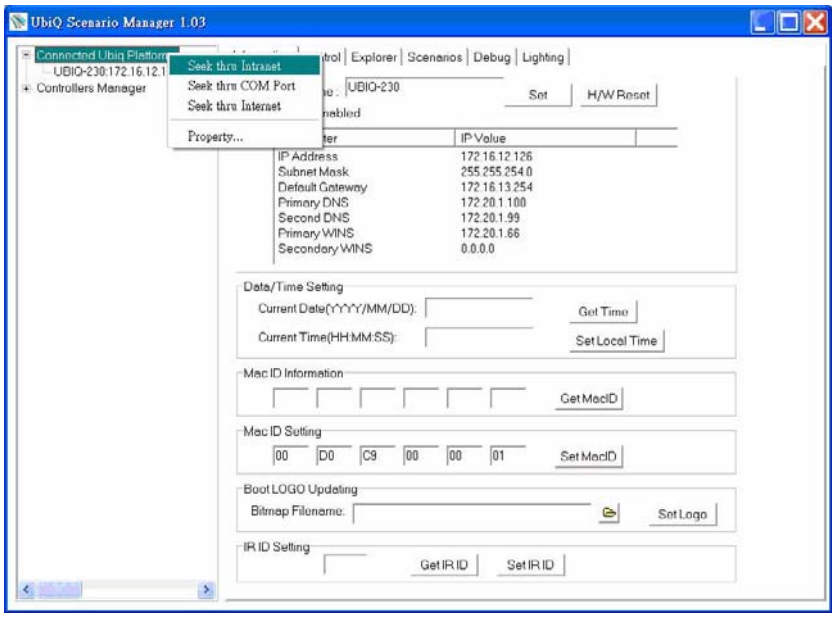
1. Power on the *UbiQ-230* , plug-in RJ45 ethernet connector to it and be sure that UbiQ-230 is on the local network.
2. Make sure that your computer is on local network.
3. Launch the UbiQ *scenario utility*.
4. Click on the menu item ”*Seek thru Intranet*” key and the UbiQ devices on your local network will be probed and linked automatically.
5. Select the UbiQ device, the *UbiQ information* will be displayed. This window includes five pages: *Information, Control, Explorer, Scenario, and Debug*.
6. Select the *Information* page, modify your UbiQ-230 name, IP address or DHCP enabled. Then click the Set button. Now the UbiQ-230 will save your setting values.
7. See the next sessions to learn how to control the UbiQ-230.

The build-in functions shipped with UbiQ-230 can help you to accomplish some basic controlling scheme. You can also code your own functions in Controller menu to enhance the control over such controllers, which will be described in Chapter 4, “*Bascics of Smart-C Script Language*” and Chapter 5, “*Functions Reference*”. The control program is a collection of functions to perform specific controlling strategy toward each I/O. Before using the newly designed functions, the CGI Syntax check is provided for syntax verification. Those functions are linked and called within the most popular programming language - C. For easier implementations, a C interpreter environment is provided. In this way, users can change properties according to their needs. From small applications interfacing only a few lines of codes, through full-scale industrial control systems running many I/O Devices simultaneously, UbiQ-230 provide you with the quickest and most efficient HMI solutions.

The following sections overview the basic functions for developing your solutions with UbiQ-230.

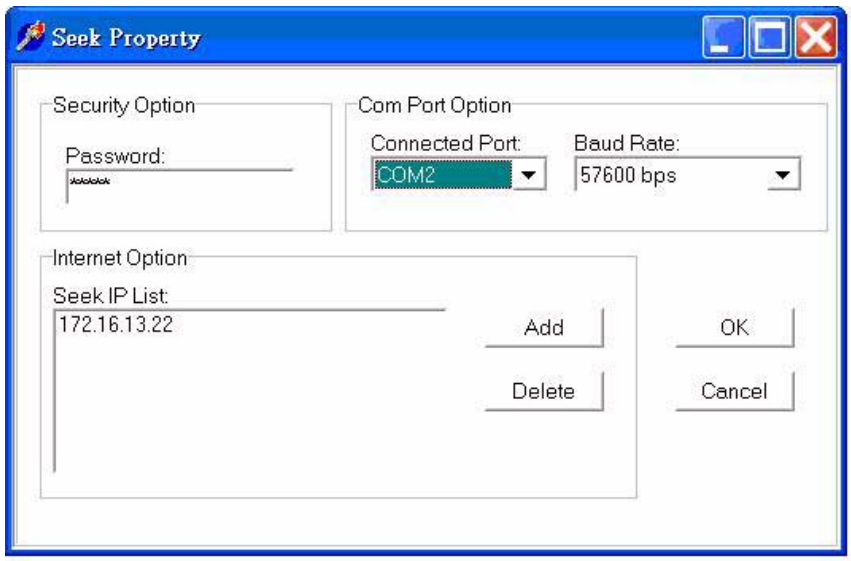
2.2 Seek Ubiq-230 Devices

While you move the mouse cursor on the item “Connected Ubiq Platforms” and press right-button of mouse, the popup menu will be displayed on the screen.

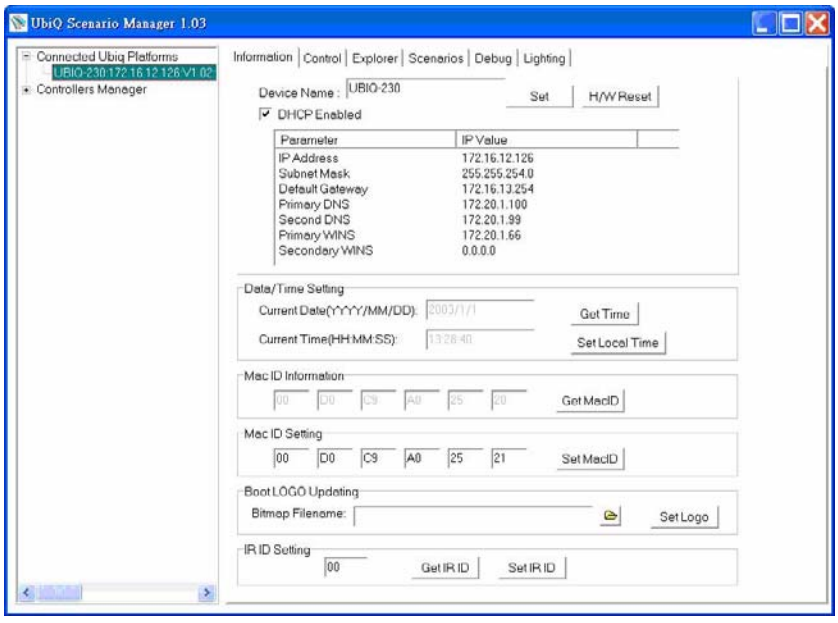


There are 4 items on the popup menu: “Seek thru Intranet”, “Seek thru COM Port”, “Seek thru Internet”, and “Property...”.

- “Seek thru Intranet” : Search the UbiQ devices on the local network. Application uses the broadcast UDP packet to get the UbiQ device’s response on the network.
- “Seek thru COM Port” : Search the UbiQ device on the host serial port. The port setting is defined on “Property...” Window.
- “Seek thru Internet” : Search the UbiQ device on the internet. Users need to specify the IP address for the UbiQ devices. These IP addresses are added on “Property...” Window.
- “Property...” : The ”Seek Property” window will be popup. There are several fields and buttons need be specified by users.
  - “Password” : Control Access Password for UbiQ device.
  - “Connected Port” : Two ports “COM1:” and “COM2:” can be chosen.
  - “Baud Rate” : Define baud rate for the connected port.
  - “Seek IP List” : Internet IP of UbiQ devices. Double-Click the IP will enter edit-mode for IP address.
  - “Add” : Add a Internet IP address to IP List.
  - “Delete” : Delete the current IP item on the IP List.

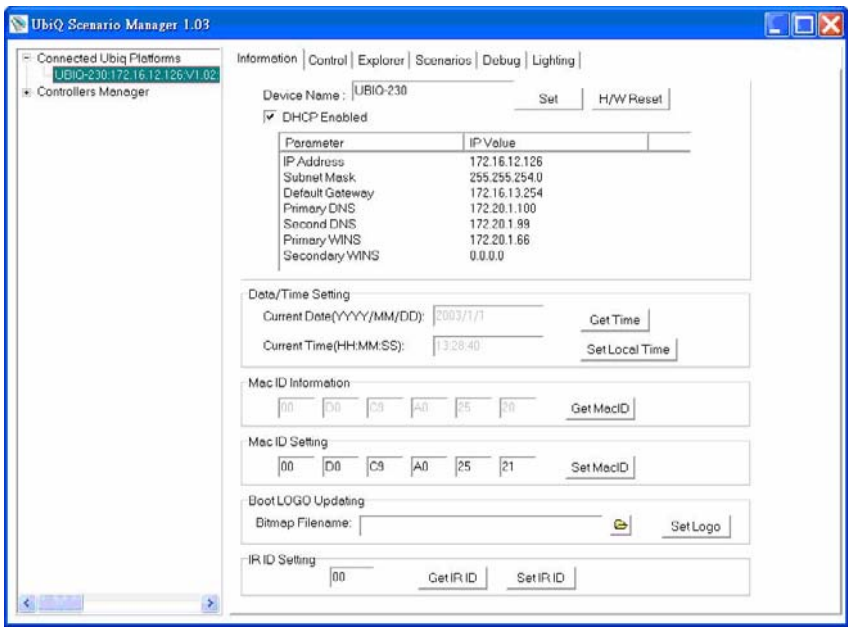


For example, there are 2 UbiQ devices on the local network and select “Seek thru Intranet” item. Then the UbiQ devices will be added to the list.

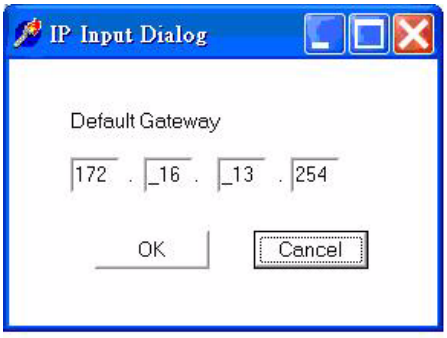


2.3 Information of Connected UbiQ

Click one item of Connected Ubiq Platform, and the information of the UbiQ device will be shown on the window:



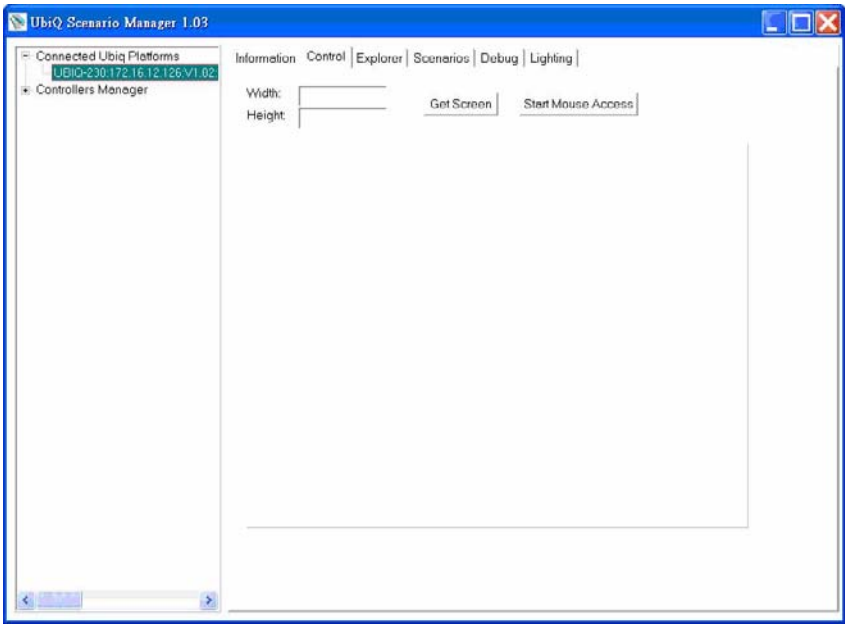
- "Device Name" field can be edited for the name of UbiQ device.
- "DHCP Enablde" checkbox is switch on/off for DCHCP server exist or not on the LAN. If "DHCP Enabled" is set to off, then the items of IP parameters can be edited by users.



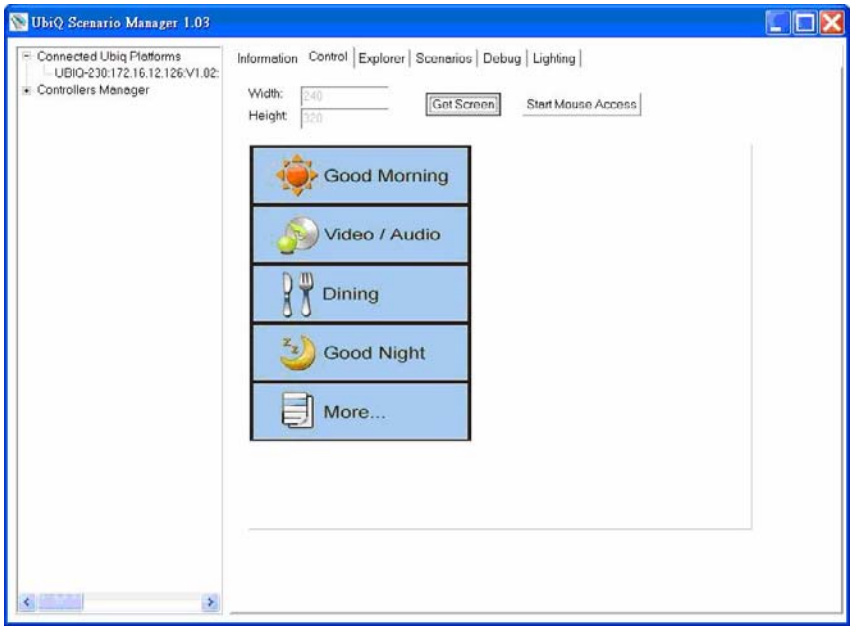
- "Set" button : notify the modified messages to UbiQ device.
- "Get Time" button : get the current time of UbiQ device.
- "Set Local Time" button : set the host PC time to UbiQ device.
- "H/W Rest" button : launch the H/W reset command to UbiQ device.
- "Get MacID" button : get the MacID address of UbiQ device.
- "Set MacID" button : set the MacID address to UbiQ device.
- "Bitmap Filename" field : specify the boot logo bitmap filename.
- "Set Logo" button : set the boot logo bitmap to UbiQ device.

2.4 Control of Connected UbiQ

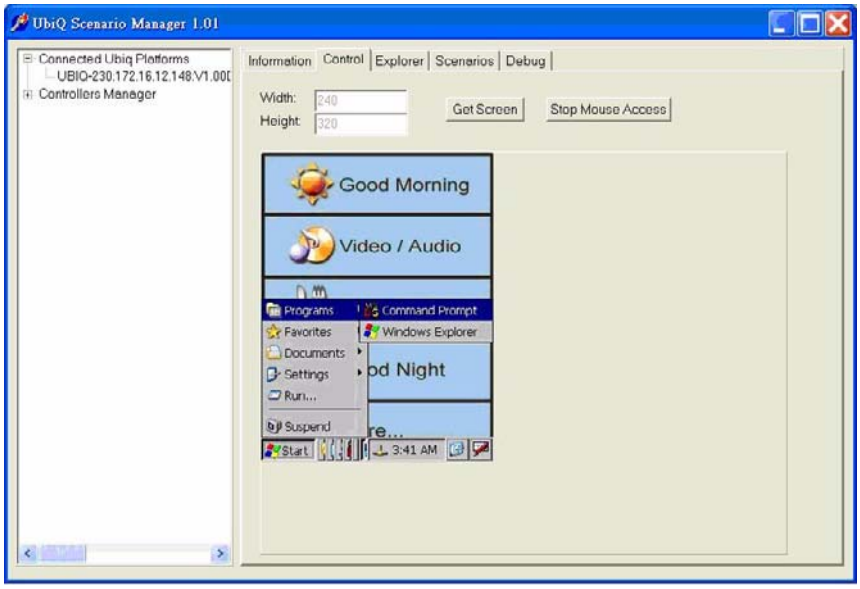
The “Control” page will enables you to access the UbiQ device remotely by mouse input.



- “Get Screen” button : Get the current screen from the UbiQ device.



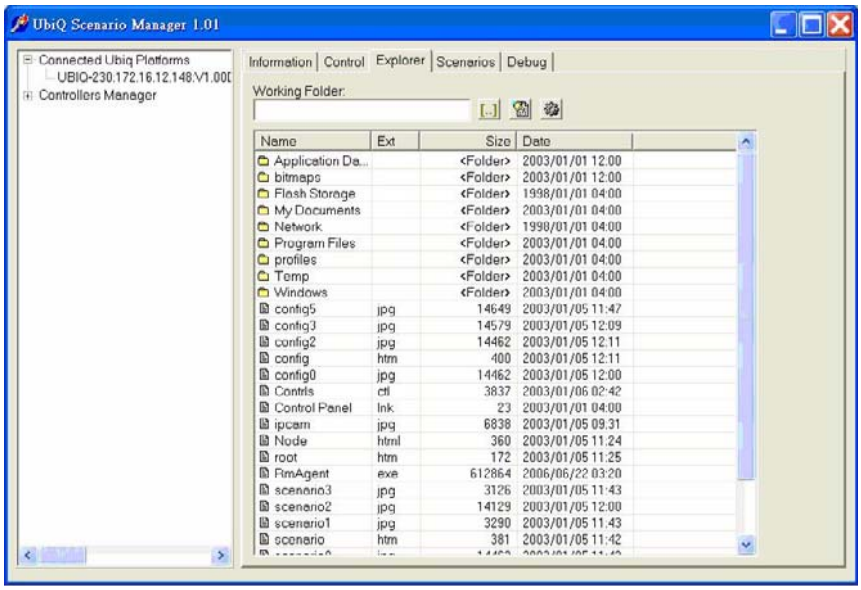
- “Start Mouse Access” button : Control the UbiQ device remotely by this page. All mouse move or clicks will pass to UbiQ device when the mouse location is on the display area. When the function is enabled, the caption of the button will be changed to “Stop Mouse Access”.



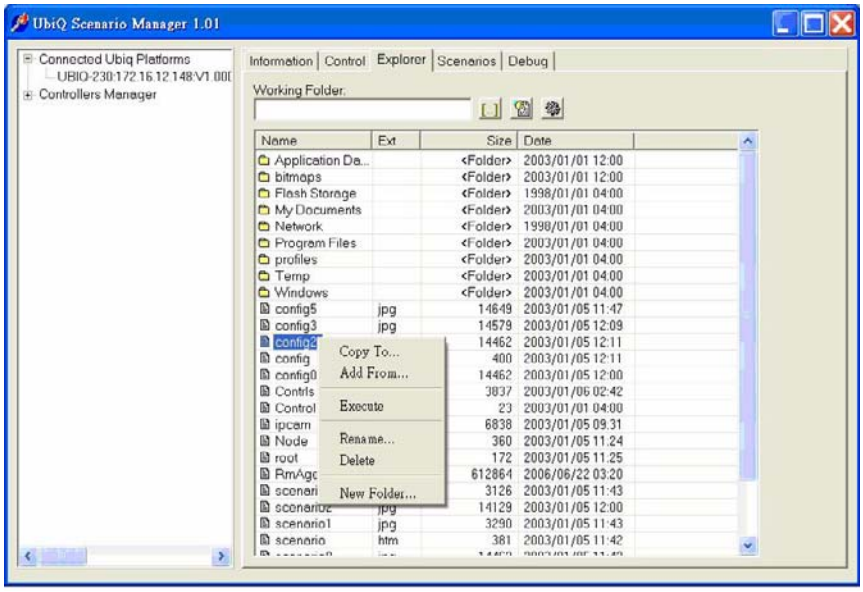
- “Stop Mouse Access” button : Stop to control the UbiQ device remotely.

2.5 Explorer of Connected UbiQ

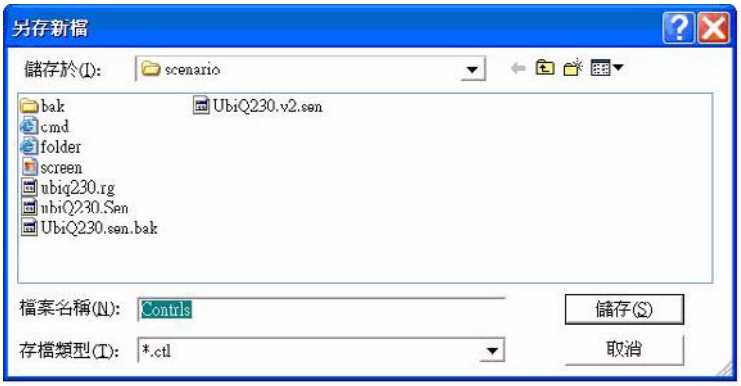
The “Explorer” Page enables you to access file system of the UbiQ device remotely.



When you move the cursor on the file view window and press the right-button of mouse, a popup menu is shown:

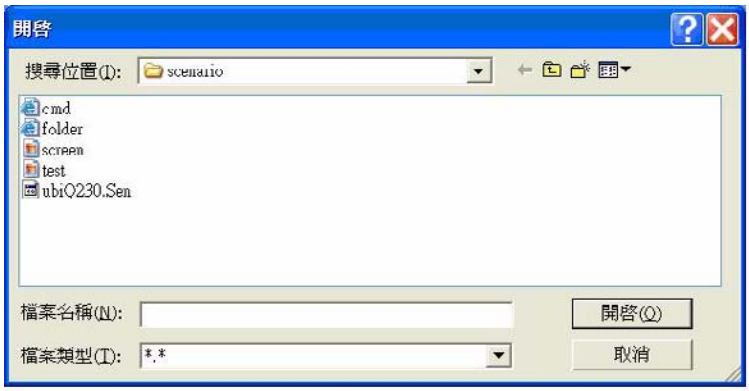


- “Copy To...” menu item: Select this item and a dialog will be prompt on screen:



Choose the folder and specify your new filename you want to save. This function is just copy a file from UbiQ device to your PC side.

- “Add From...” menu item: Select this item and a dialog will be prompt on screen:



Choose the folder and a file you want to add to UbiQ device. This function is just copy a file from PC side to UbiQ device.

- “Execute” menu item: Select this item and this file will be executed on UbiQ device.

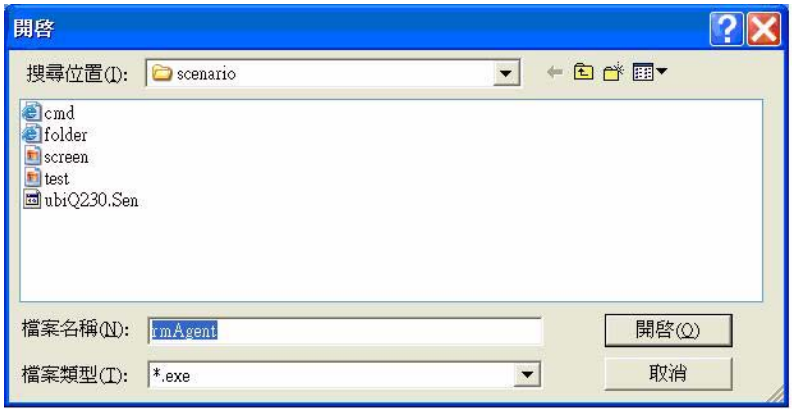
- “Rename...” menu item: Select this item and a dialog will be prompt on the screen:

Input the new file name you want to rename and press OK button to confirm or Cancel button to disable this action.

- “Delete” menu item: Select this item and this file on UbiQ device will be deleted.
- “New Folder...” menu item: Select this item and a dialog will be prompt on the screen:

Input the new folder name you want to create and press OK button to confirm or Cancel button to disable this action.

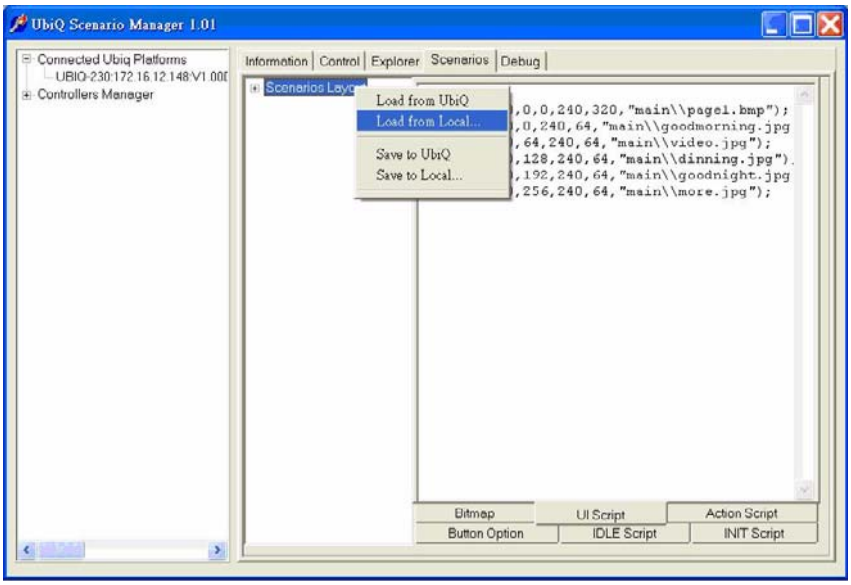
- “Go To Parent Folder” button: Click this button, then UbiQ device will go to parent folder and file explorer area will be refreshed.
- “Refresh” button: Click this button, then file explorer area will be refreshed with current folder file list got from UbiQ device.
- “Upgrade” button: Click this button, then a dialog will be prompt on the screen:



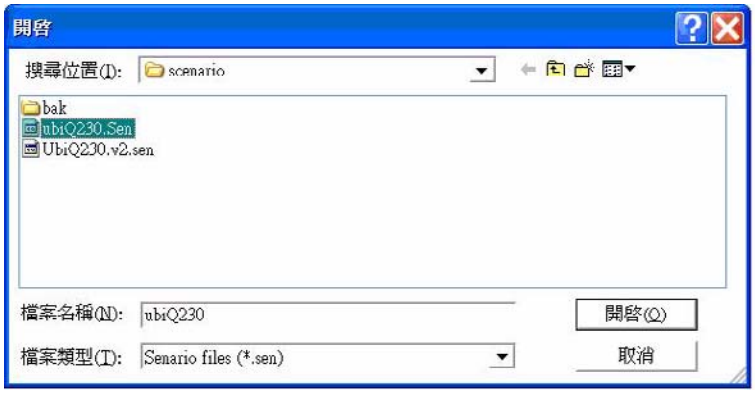
Choose the correct platform engine utility and press OK button to confirm or press Cancel button to cancel this action. Please take care to use this function. It could destroy the UbiQ device program if you choose the wrong file to upgrade.

2.6 Scenarios of Connected UbiQ

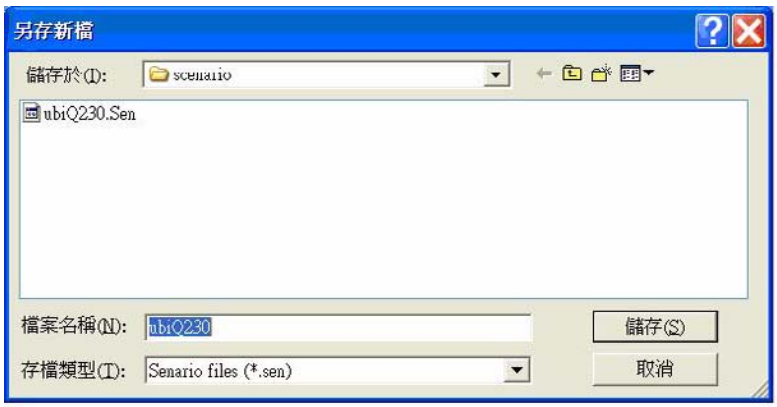
The Scenario page will let you configure the user interface and control flow to UbiQ device. You need to prepare the bitmaps for display and write the controlling codes for relative connected controllers. We provide a sample for your reference. Please move the mouse cursor to the Scenario area and press right-button. A popup menu will be prompt:



- “Load from UbiQ” menu item: Select this menu item, then the Scenario configuration file will be got and refreshed on scenario display area.
- “Load from Local...” menu item: Select this menu item, then the Scenario configuration file will be loaded from local disk through below dialog:



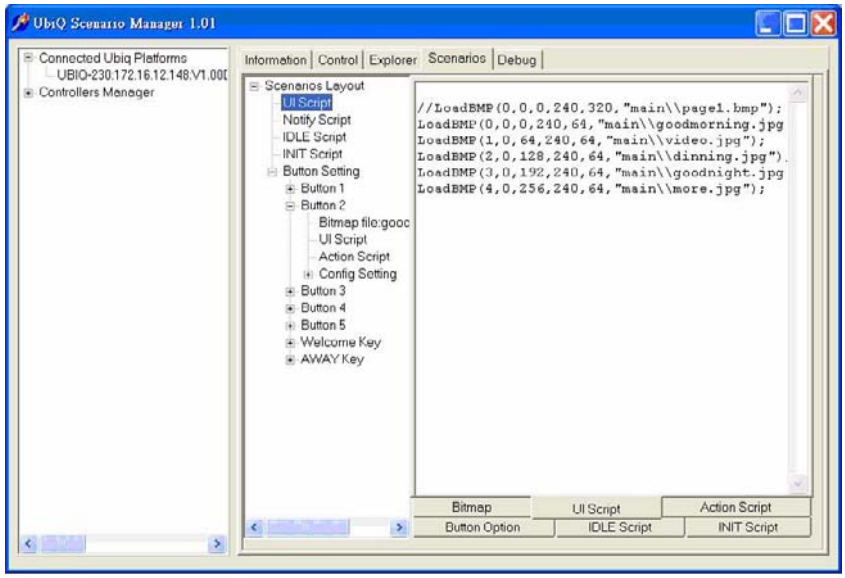
- “Save to UbiQ” menu item: Select this menu item, then Scenario configuration file will be transferred to UbiQ device.
- “Save to Local...” menu item: Select this menu item, then a dialog window will be shown as follows:



Choose the folder and specify the filename for scenario configuration file.

On the scenario configuration area, there are two basic elements needed to explain: one is button and the other is script. There are seven buttons physically on the UbiQ-230, so we need to define display interface and control flow after press these buttons. Control flow is set according to button style and action script. Display interface is defined by UI script.

- "Scenario Layout" is the root node for scenario configuration. It has several scripts and seven sub-button needed to be configured.
- "UI Script" page is used to define display area. For example, in the "Scenario Layout" root node will add following scripts on "UI Script" page:



//LoadBMP(0,0,0,240,320,"main\\page1.bmp");

LoadBMP(0,0,0,240,64,"main\\goodmorning.jpg");

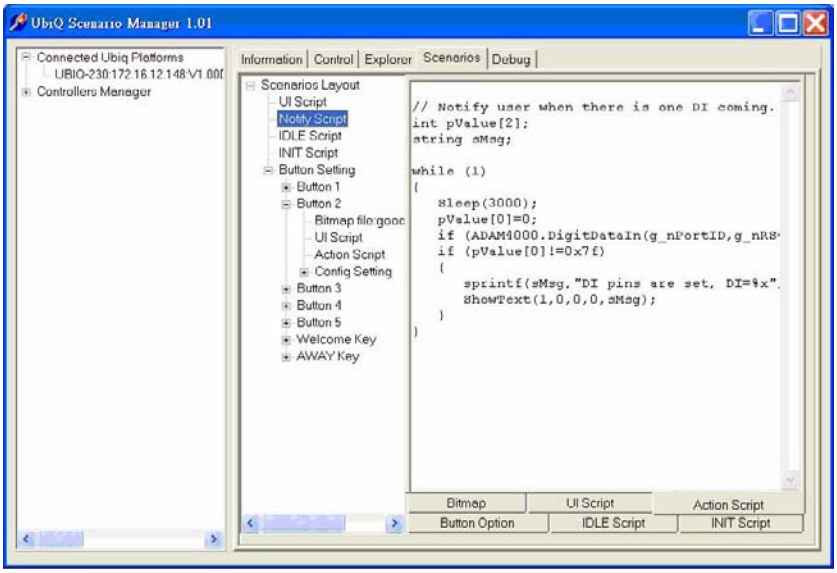
LoadBMP(1,0,64,240,64,"main\\video.jpg");

LoadBMP(2,0,128,240,64,"main\\dinning.jpg");

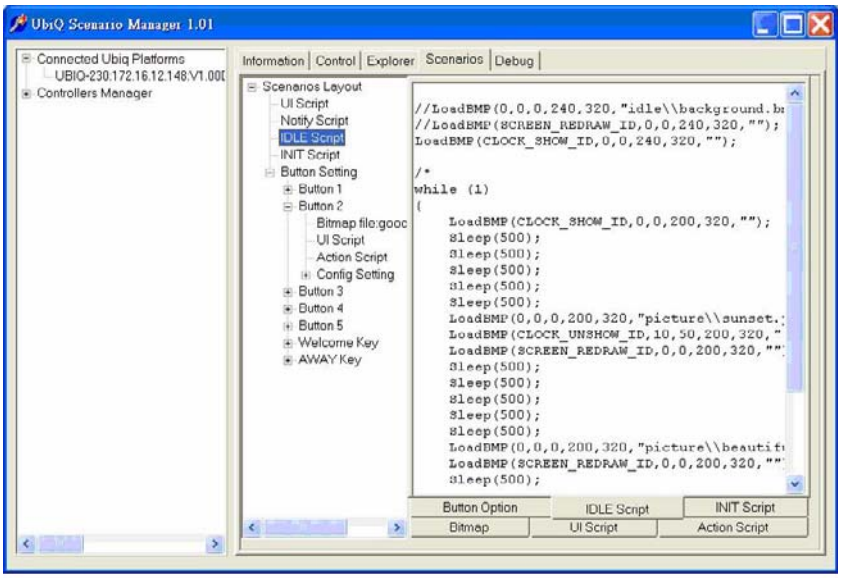
LoadBMP(3,0,192,240,64,"main\\goodnight.jpg");

LoadBMP(4,0,256,240,64,"main\\more.jpg");

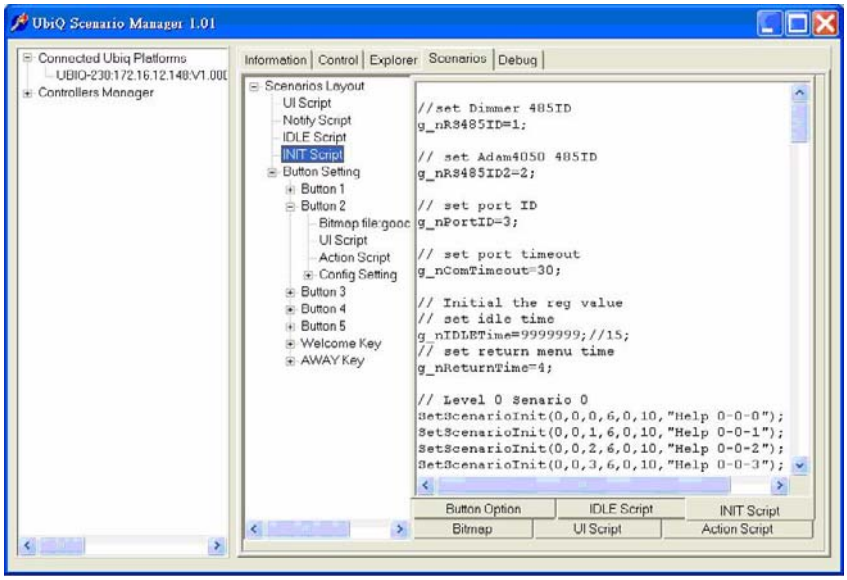
"Notify Script" page is used to detect some events happened and notify some actions by script.



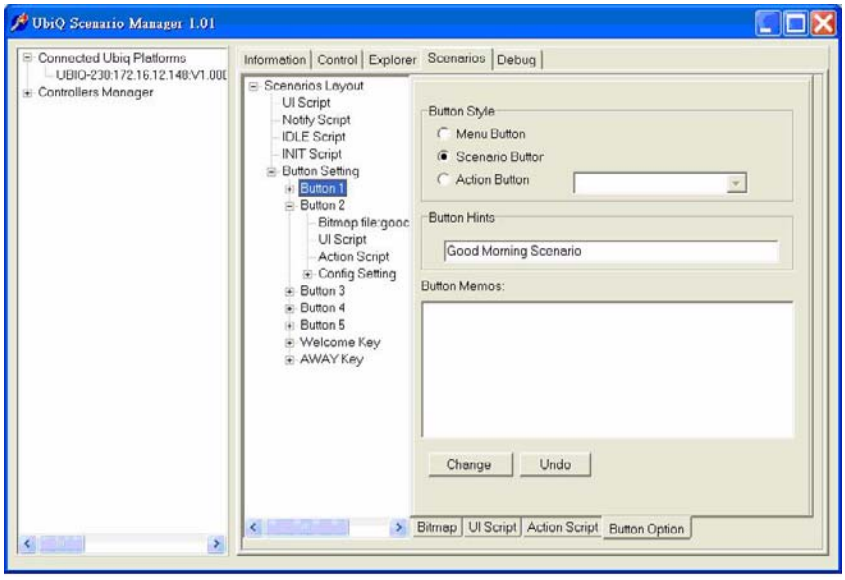
“IDLE Script” page is used to execute some actions while idle mode timer is active.



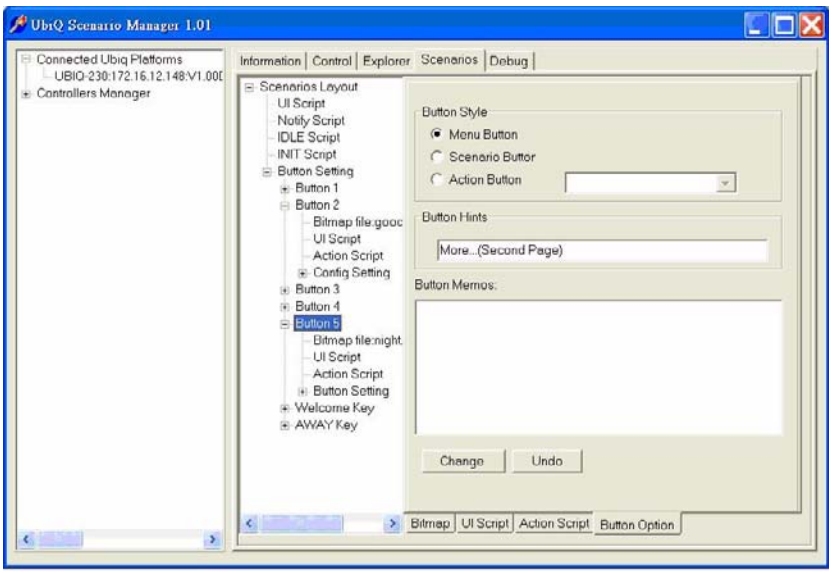
“INIT Script” page is used to execute some actions while scenario configuration file is reset.



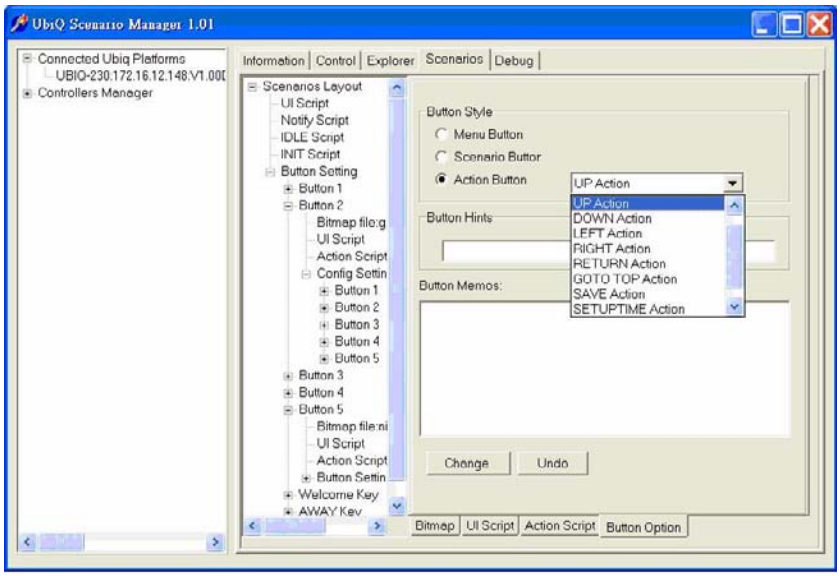
"Button Option" page is used to define the button style, button hint and button memo.



- "Menu Button" radio checkbox: Click this checkbox, and button style will be changed to MENU button. MENU button means that it has child-buttons for menu, scenario, or action buttons. For example, we define button 5 as Menu button, so button 5 still has the "Button Setting" sub-node to define user interface and control flow for its child button nodes.

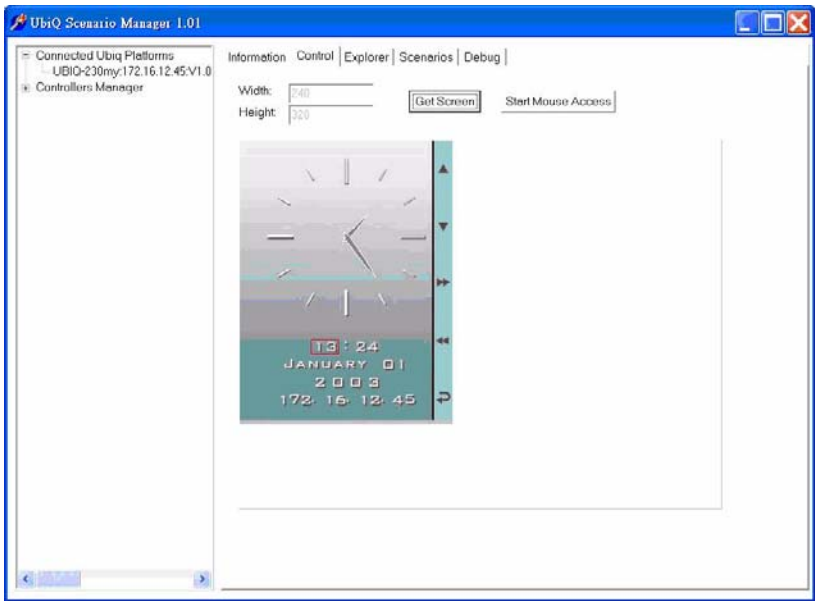


- “Scenario Button” radio checkbox: Click this checkbox, and button style will be changed to Scenario button. Scenario button means that it is real button for scenario control and no child-buttons, so we need provide setup methods for configuration mode. For example, we define button 2 as Menu button, so button 2 only has the “Config Setting” sub-node to define user interface and control flow for its scenario configuration behaviors.
- “Action Button” radio checkbox: Click this checkbox, and button style will be changed to Action button. Action button means that it is real button for some actions. There are several built-in actions on button as follows:

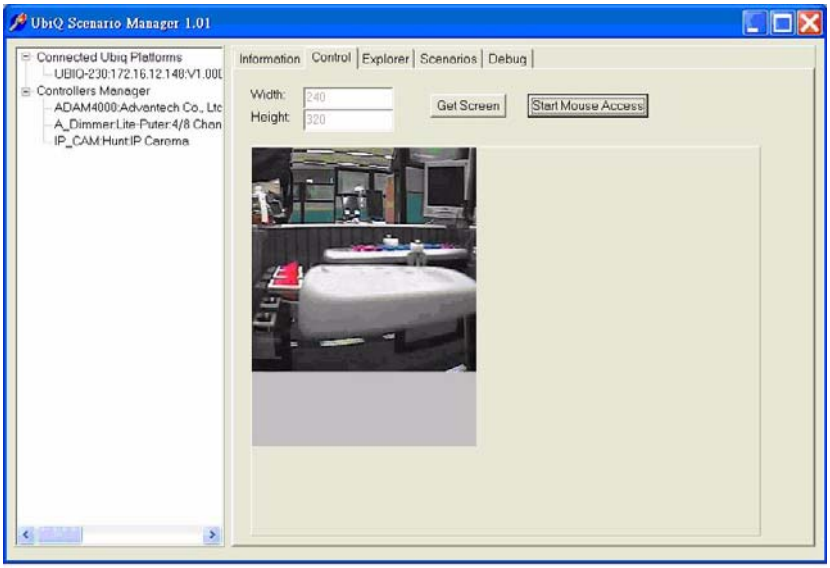


- “UP Action”: Add one degree if the current item is numeral (i.e., the maximum is bigger than 1) or set to ON flag if the current is switch (i.e., the maximum is equal 1). Use only on Configuration mode.
- “DOWN Action”: Decease one degree if the current item is numeral (i.e., the maximum is bigger than 1) or set to OFF flag if the current is switch (i.e., the maximum is equal 1). Use only on Configuration mode.

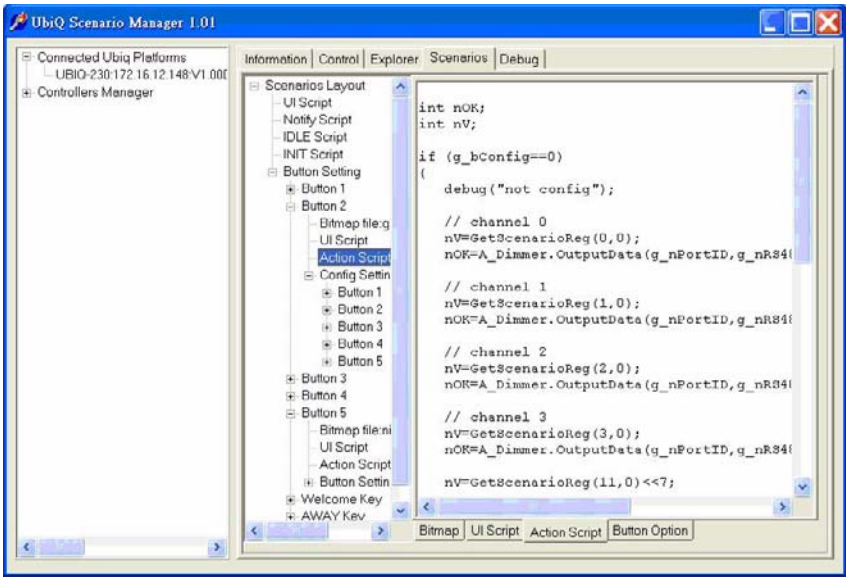
- “LEFT Action”: Move current index to the previous item. When the current item is first, the current index will move to the last item. Use only on Configuration mode.
- “RIGHT Action”: Move current index to the next item. When the current item is last, the current index will move to the first item. Use only on Configuration mode.
- “RETURN Action”: Change the control flow to parent button node. The user interface (UI script) of parent button will be executed.
- “GOTO TOP Action”: Change the control flow to root button node. The user interface (UI script) of root button will be executed.
- “SAVE Action”: Save current configuration values to the file system. Use only on Configuration mode.
- “SETUPTIME Action”: Change the control flow to the Setup Clock display. In this display, you can modify current date and time on UbiQ device.



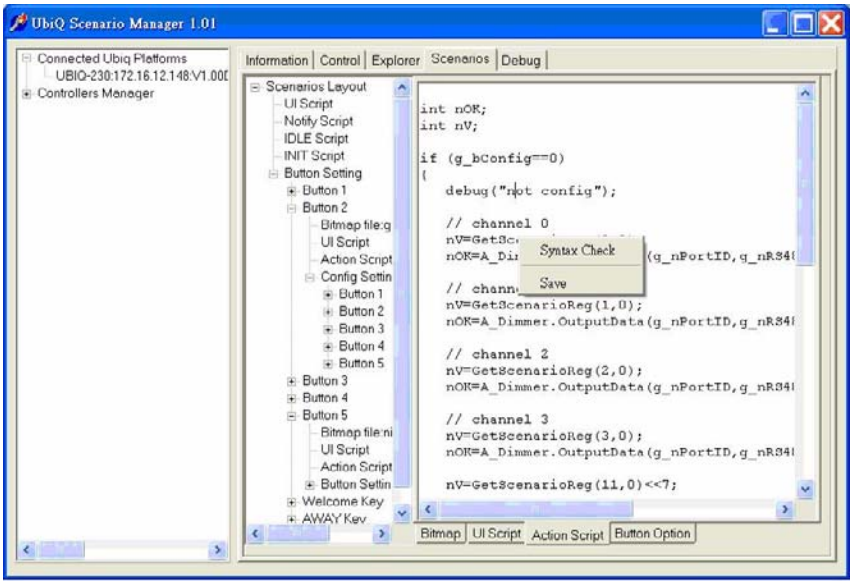
- “IP CAM Action” : Change the control flow to the IP CAM display. You should specify the picture file from IP CAM on °×UI script°±. In this display, you can see dynamic picture on UbiQ device.



- “Button Hint” edit field: Input the help message for this button. It will be shown with a item on homepage in Text mode when user views UbiQ device with Internet Browser.
- “Button Memos” edit list: Input the memo text for this button.
- “Action Script” page is used to access I/O with scripts.



In Script area, you could press the right-button of mouse, and a popup menu will appear:

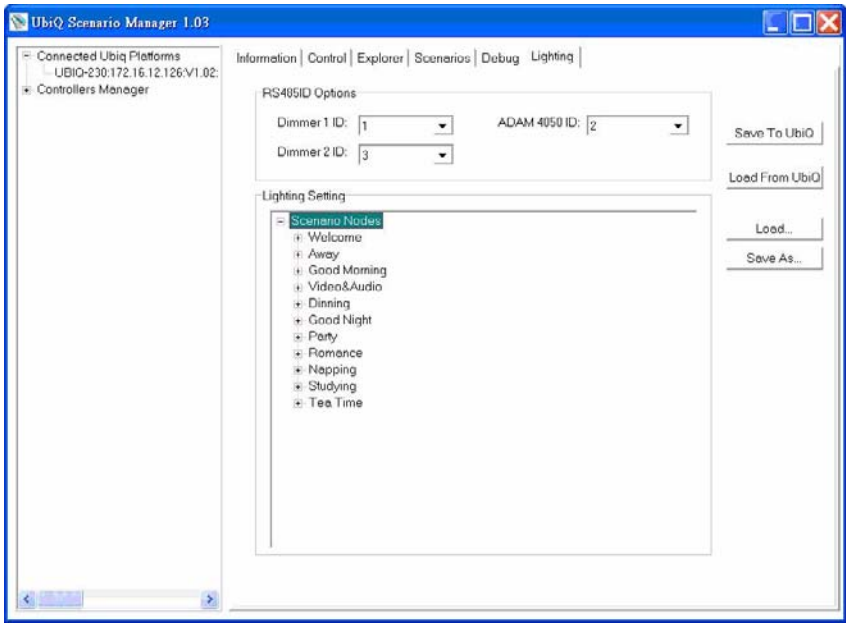


- “Syntax Check” menu item: Select this menu, then the script engine will check the script syntax correct or not. If there are some errors on syntax check, a popup window will be shown:

- “Save” menu item: Select this item, then this script will be saved to internal memory.

2.7 Lighting of Connected UbiQ

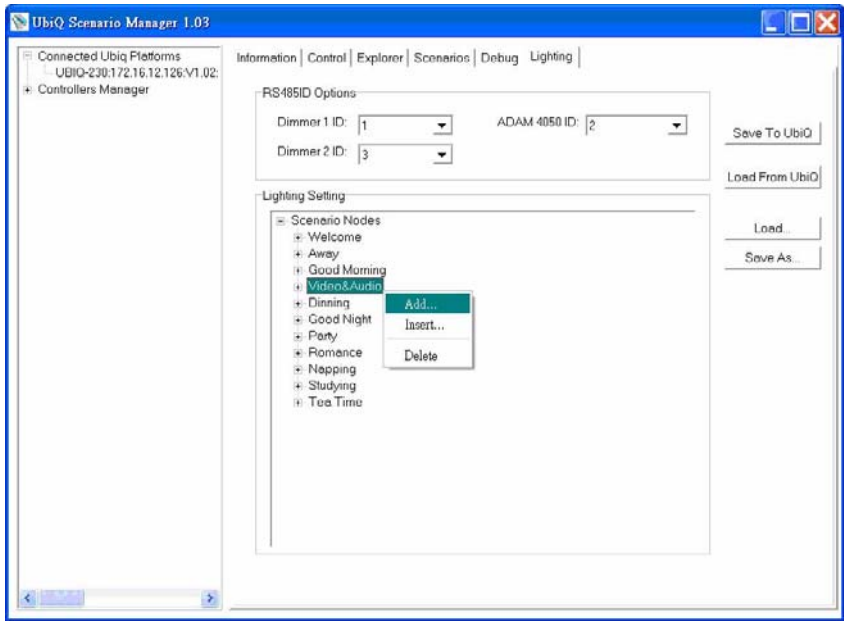
The Lighting page will let you configure the parameters for dimmers and DO controllers connected with UbiQ device. The page is shown as follows:



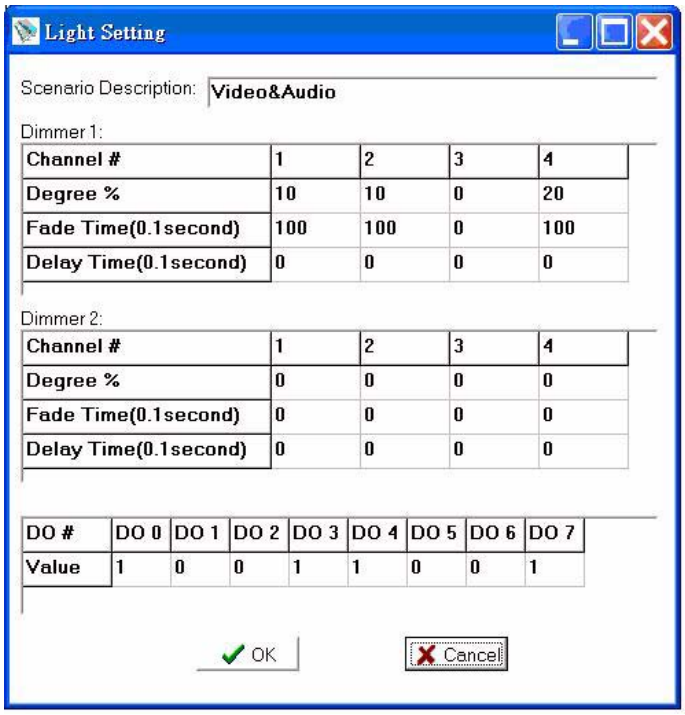
There are 3 RS485 ID specified:

1. Dimmer 1 ID: Specified the value and is stored to the global variable g_nRS485ID.
2. Dimmer 2 ID: Specified the value and is stored to the global variable g_nRS485ID3.
3. ADAM 4050 ID: Specified the value and is stored to the global variable g_nRS485ID2.

Move the mouse cursor on “Lighting Setting” tree view, a popup menu will be shown:

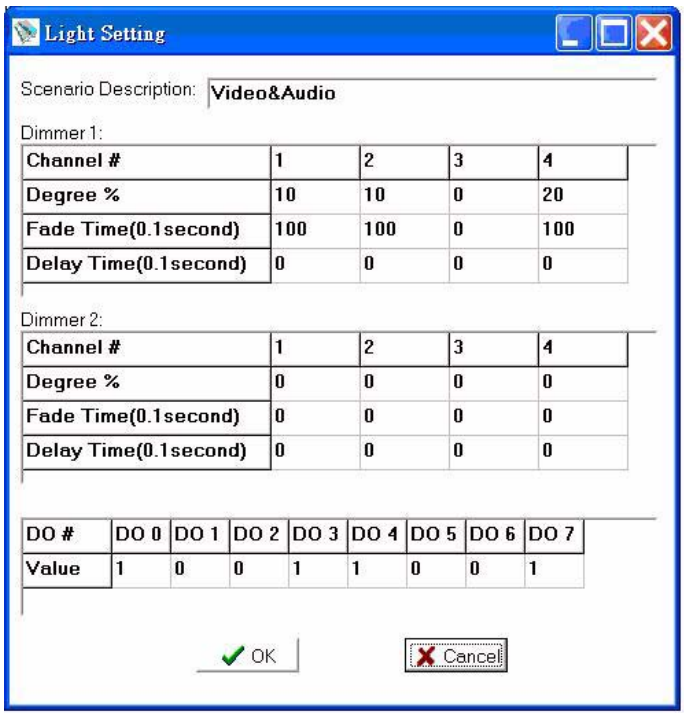


- “Add...” menu item : Click the item and a dialog will shown as follows:



Input the scenario description and the table for dimmer 1, dimmer2 and DO controllers. Press OK to confirm the modification or press Cancel button to cancel the input. The added scenario item will be a last item on the tree view.

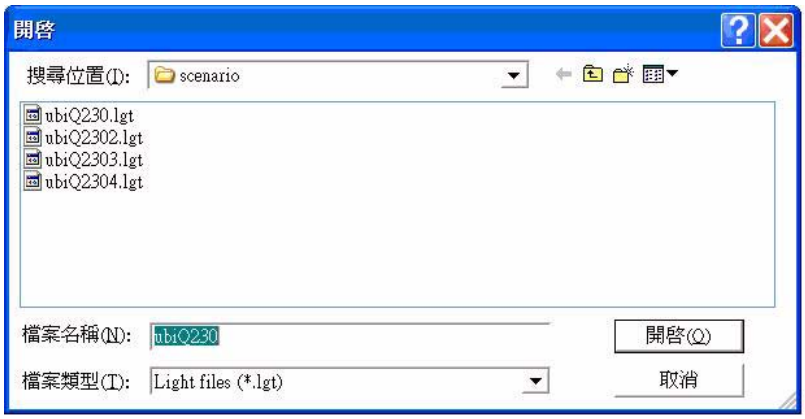
- “Insert...” menu item : Click the item and a dialog will shown as follows:



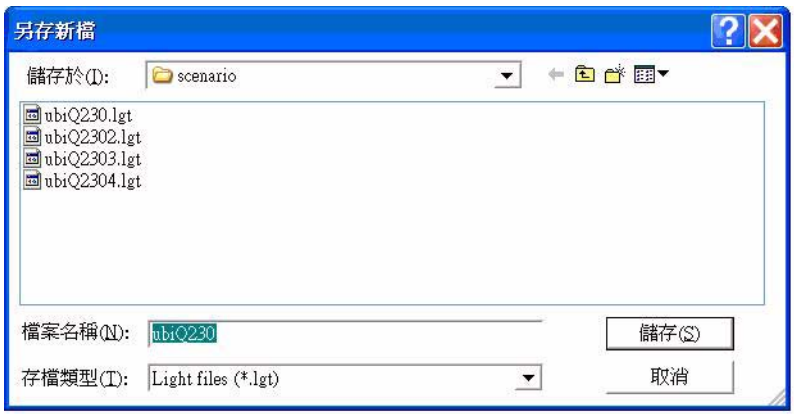
Input the scenario description and the table for dimmer 1, dimmer2 and DO controllers. Press OK to confirm the modification or press Cancel button to cancel the input. The added scenario item will be a preceding item above the selected item on the tree view.

To modify the item, you could double-click the mouse left button.

- "Save to UbiQ" button: Click the button, then the Scenario Lighting file will be got and refreshed on scenario display area.
- "Load from UbiQ" button: Click the button, the lighting setting file will transfer to UbiQ device.
- "Load..." button: Click the button, a open dialog will be shown:

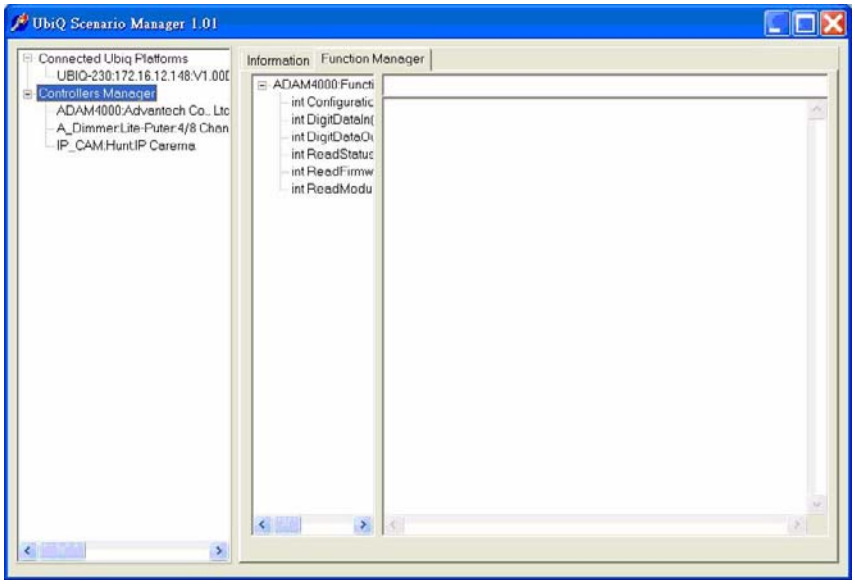


- "Save As..." button: Click the button, a save as dialog will be shown:

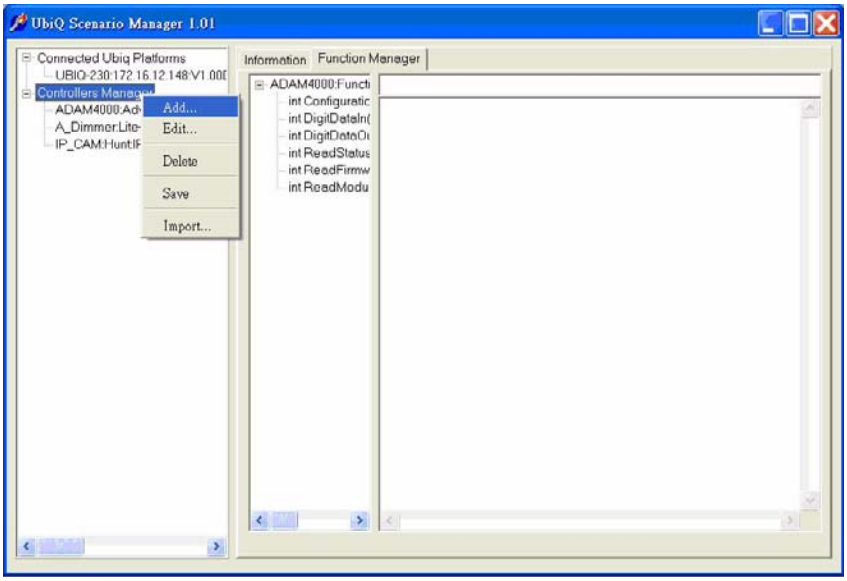


2.8 Controller Manager

The controller manager enables the user adding and modifying connected controller I/O access functions.



Move mouse cursor to the Controller Manager area and press right-button of mouse, then a popup menu will be shown:



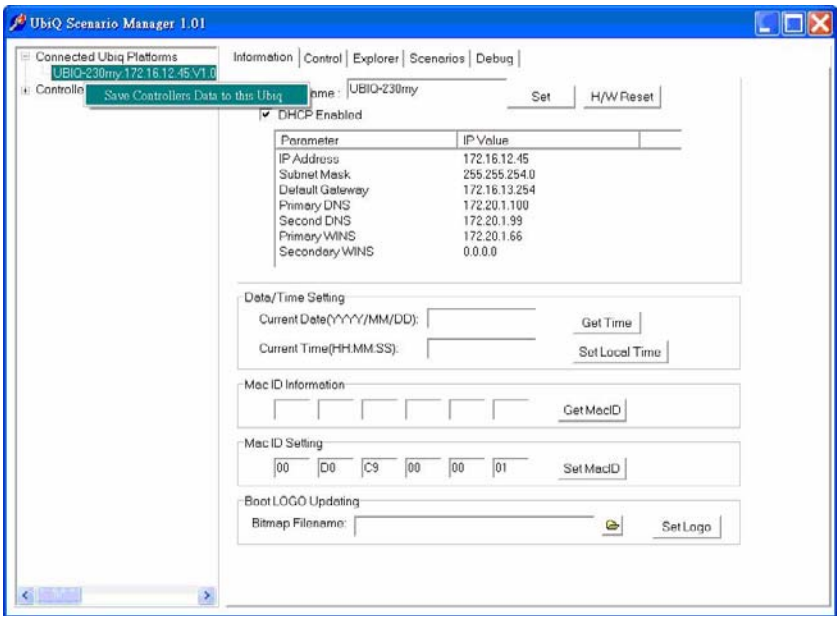
- "Add..." menu item: Select this item, then a dialog window will be shown on the screen:

Input the new controller name, vendor name, and description on the edit fields, and then press OK button to confirm or press Cancel button to cancel.

- "Edit..." menu item: Select this item and a dialog window with current controller information will be shown:

- “Delete” menu item: Select this item, then the current controller will be deleted.
- “Save” menu item: Select this item, then all controllers information will save to local file “contrls.ctl”.

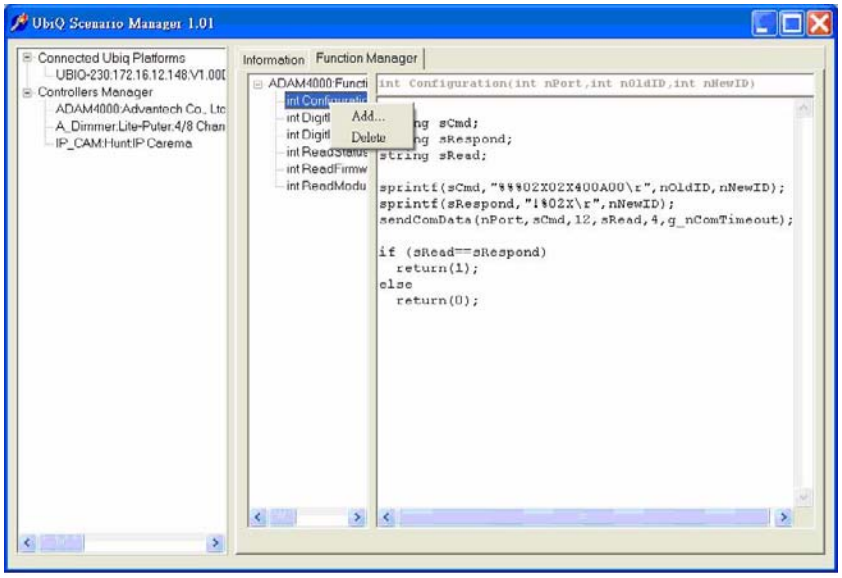
After you have modified the information of controllers, you could transfer controllers data to UbiQ device. Please select one UbiQ device and press right-button of mouse, then a popup menu is prompt:



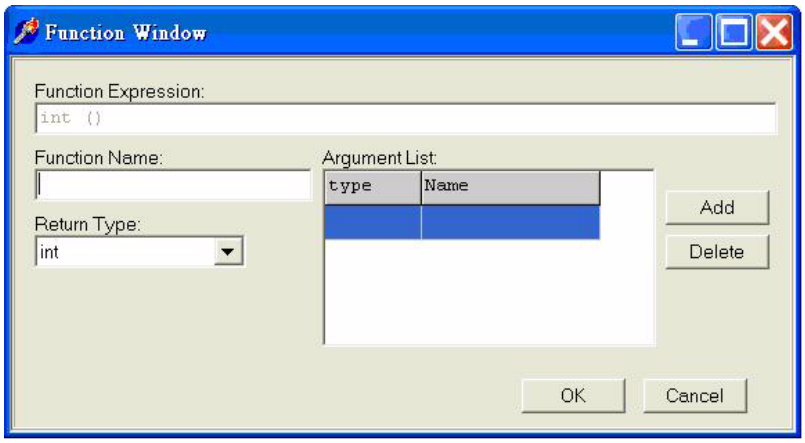
- “Save Controllers Data to this Ubiq” menu item: Select this item, then the controllers data will be transferred to UbiQ device.

2.9 Function Manager Page

The function manager page of controller manager enables the user adding and modifying I/O access functions for specified controller. Move the mouse cursor and press right-button of mouse, then a popup menu will be shown:

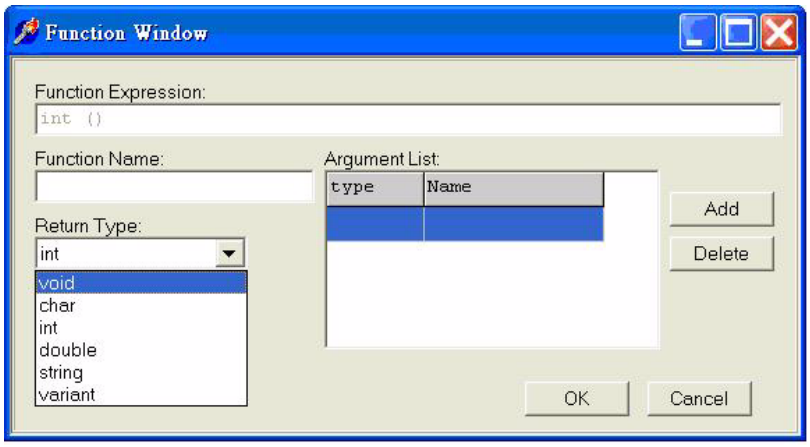


- “Add...” menu item: Select this item, then a dialog will appear on screen:



This dialog has several fields and buttons to let you input the function name, type, and arguments.

- "Function Name" field : Input the function name.
- "Return Type" combo-box : Choose one return type of function. The list of function type are the following:

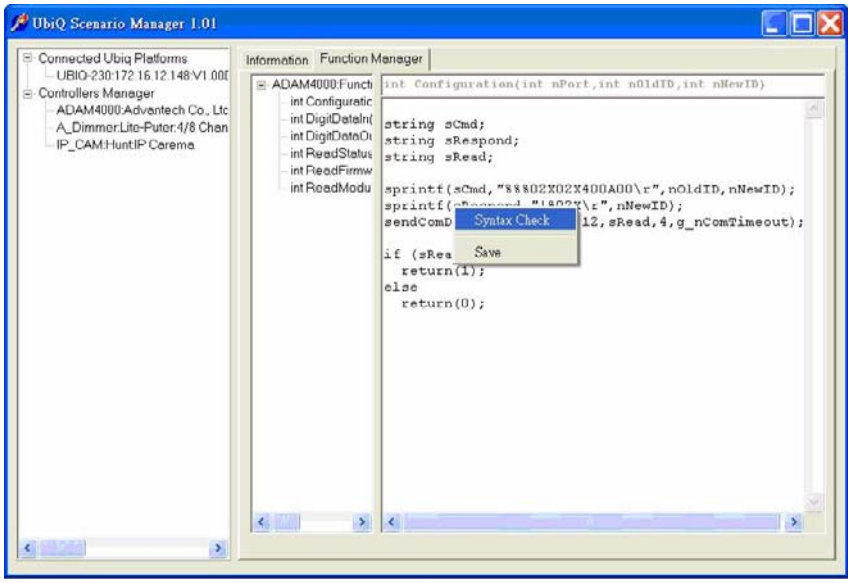


  - void : The function returns none.
  - char : The function returns a character.
  - int : The function returns a integer.
  - double : The function returns a real number.
  - string : The function returns a string.
  - variant : The function returns a variant variable.
- "Add" button : Click this button, then a argument window will be shown:

Input the new name and type for the argument.

- “Delete” button : Click this button, then the current argument will be deleted from the argument list.
- “Delete” menu item: Select this item, then the current function will be deleted from the function list.

Move the mouse cursor to function editor and press right-button of mouse, then a popup menu will be prompt:



- “Syntax Check” menu item: Select this menu, then the script engine will check the script syntax correct or not. If there are some errors on syntax check, a popup window will be shown:

- “Save” menu item: Select this item, then this function will be saved to internal memory.

Chapter 3

Tutorials

3.1 Tutorials

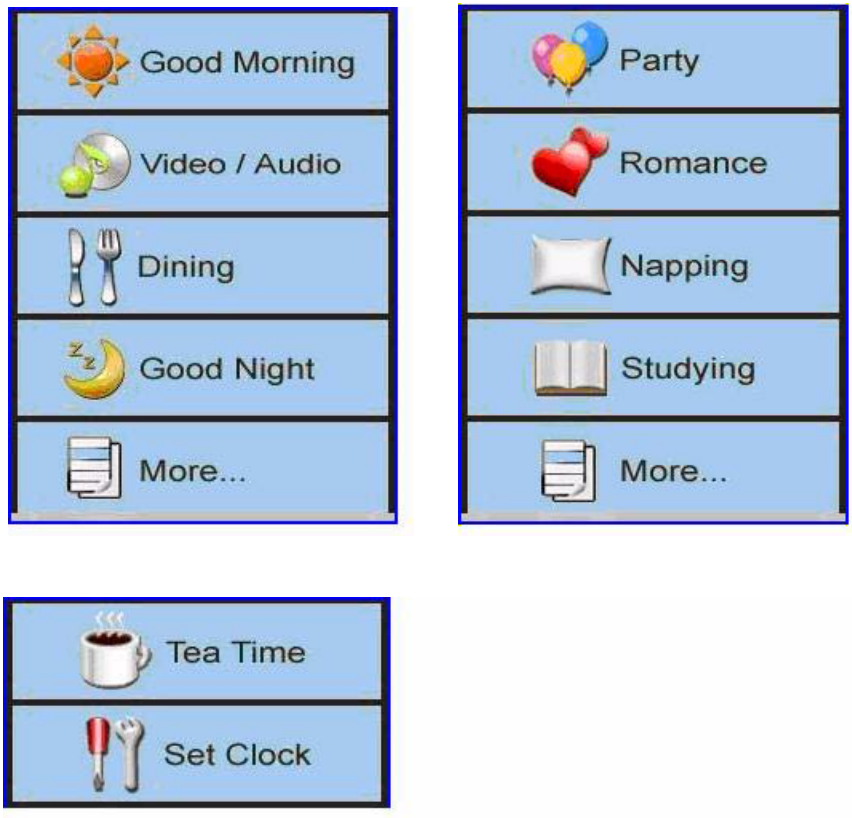
This chapter teaches you how to design a web-enabled system by using Advantech's UbiQ-230 device. Firstly, scratch your system charts, decide what controllers used, collect their communicating protocols, and program what data you want. Secondly, build a simply but workable web system by Scenario Manager utility and UbiQ-230 device. Finally, improve the web system by adding interesting graphs and associating controllers' data with your CGI scripts.

3.2 Define your user interface on UbiQ-230

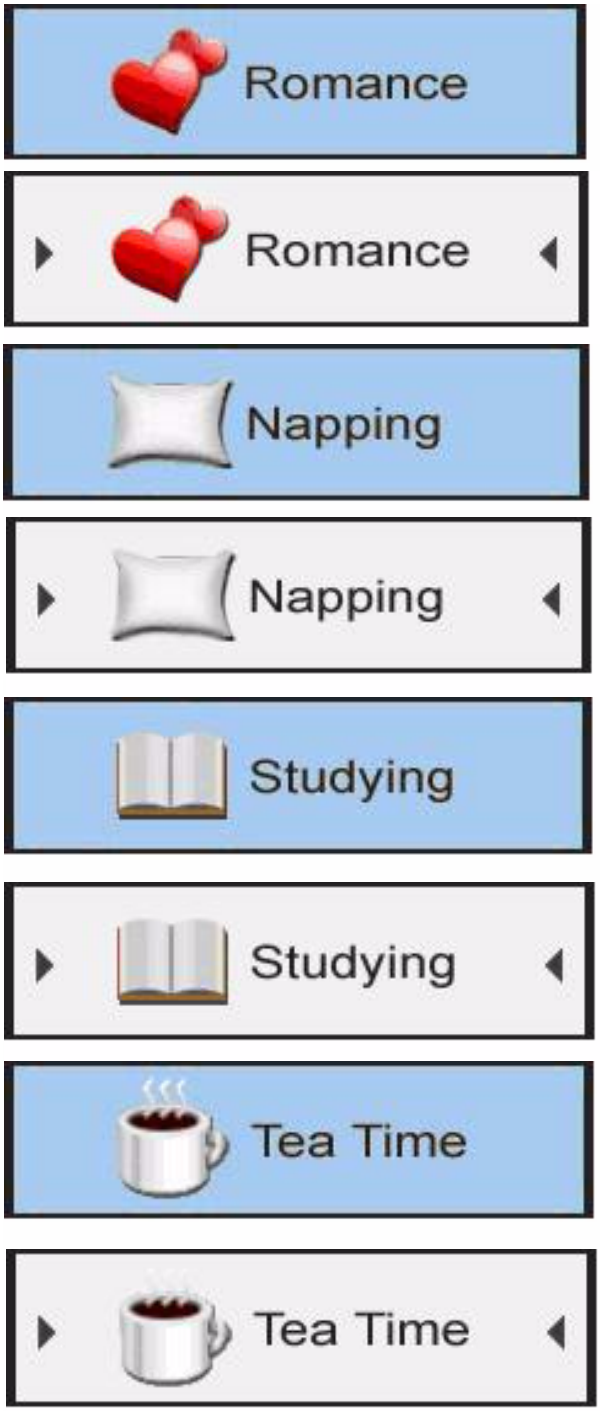
The product UbiQ-230 is defined as a scenario controller. So the user interface on the first level should be scenario menu items. The user interface on second level should show individual volumes for dimmers and DIO points. Suppose that we would like to implement 9 scenarios (not including welcome and away modes) as follows:

- Good Morning
- Video / Audio
- Dining
- Good Night
- Party
- Romance
- Napping
- Studying
- Tea Time

Because there are only five buttons on UbiQ-230 device, we need to separate scenario modes to more pages. For example, we define 3 pages items UI as follows:



For each scenario mode, we could define 2 pictures to show. One is normal, and the other is selected.



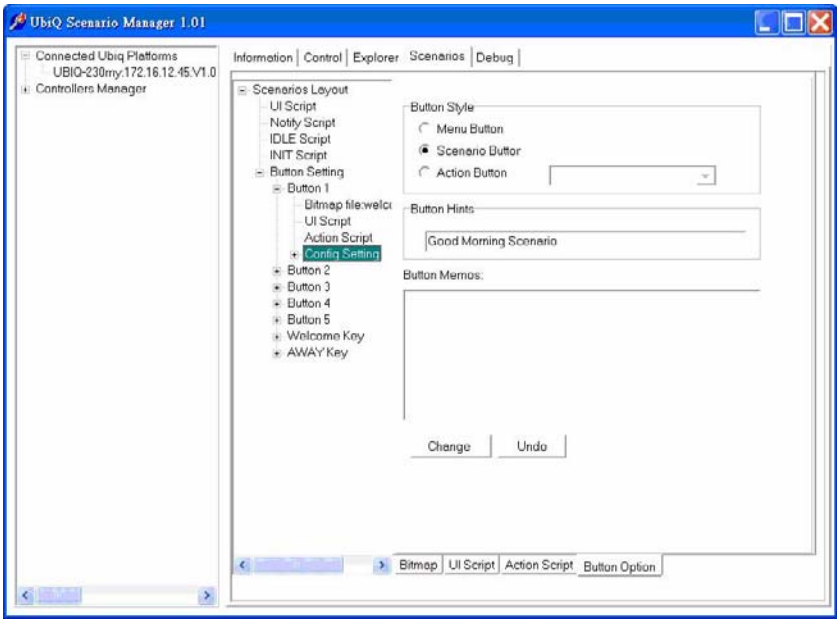
We need the “more...” item to indicate more pages on the following.

In the UI Script of °×Scenario Layout°± node, we define UI as follows:
LoadBMP(0,0,0,240,64,"main\\goodmorning.jpg");
LoadBMP(1,0,64,240,64,"main\\video.jpg");
LoadBMP(2,0,128,240,64,"main\\dinning.jpg");

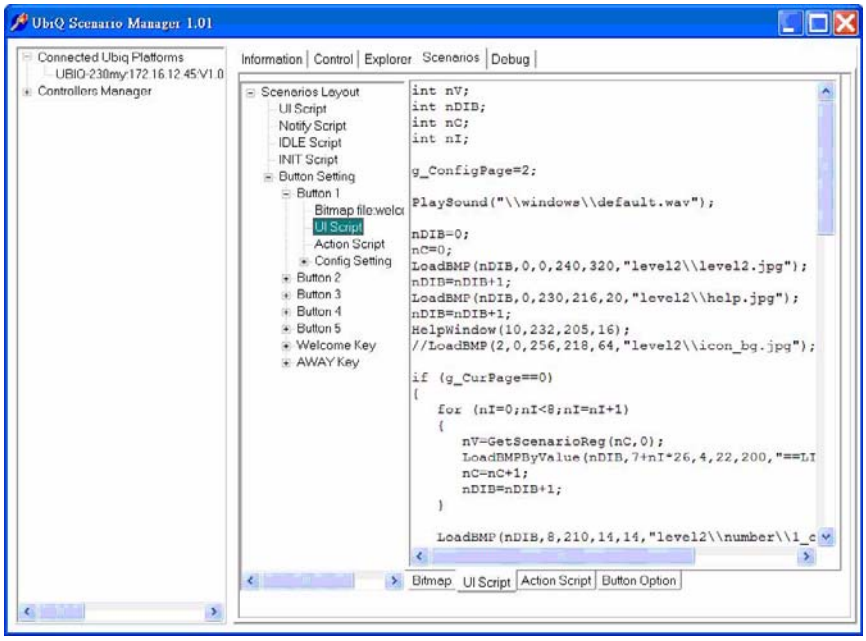
LoadBMP(3,0,192,240,64,"main\\goodnight.jpg");
LoadBMP(4,0,256,240,64,"main\\more0.jpg");
In the UI Script of °×Button 5°± node, we define UI as follows:
LoadBMP(0,0,0,240,64,"main\\party.jpg");
LoadBMP(1,0,64,240,64,"main\\romance.jpg");
LoadBMP(2,0,128,240,64,"main\\napping.jpg");
LoadBMP(3,0,192,240,64,"main\\studying.jpg");
LoadBMP(4,0,256,240,64,"main\\more1.jpg");

After we defined the UI for first level, we should think what the second user interfaces are. What number circuits of dimmers are used and digit out signals are needed to trigger? How to display the detailed information user could be understand? One example is the following:

We defined the style of “Button 1” as *Scenario Button*. The button hint is set to “Good Morning Scenario”. This hint will be shown on the Internet Browser.



Then we begin to write the UI script for “Button 1”.



The code sample is the following:

```
int nV;
int nDIB;
int nC;
int nI;

g_ConfigPage=2;

PlaySound("\\windows\\default.wav");

nDIB=0;
nC=0;
LoadBMP(nDIB,0,0,240,320,"level2\\level2.jpg");
nDIB=nDIB+1;
LoadBMP(nDIB,0,230,216,20,"level2\\help.jpg");
nDIB=nDIB+1;
HelpWindow(10,232,205,16);

if (g_CurPage==0)
{
  for (nI=0;nI<8;nI=nI+1)
  {
    nV=GetScenarioReg(nC,0);
    LoadBMPByValue(nDIB,7+nI*26,4,22,200,"==LIGHT_BAR4",nV);
    nC=nC+1;
    nDIB=nDIB+1;
  }
```

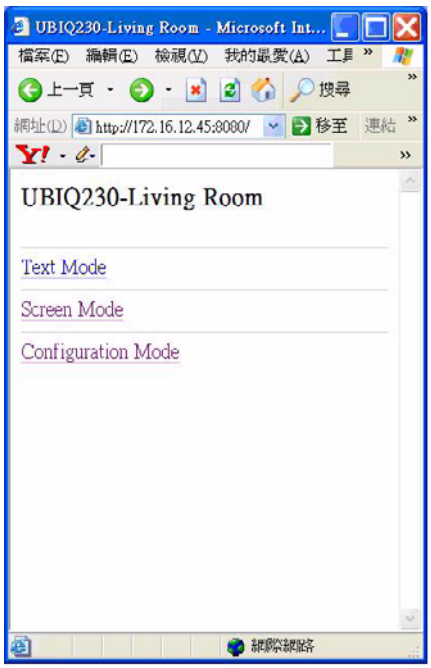
```
  LoadBMP(nDIB,8,210,14,14,"level2\\number\\1_on.bmp");
  nDIB=nDIB+1;
  LoadBMP(nDIB,35,210,14,14,"level2\\number\\2_on.bmp");
  nDIB=nDIB+1;
  LoadBMP(nDIB,62,210,14,14,"level2\\number\\3_on.bmp");
  nDIB=nDIB+1;
  LoadBMP(nDIB,89,210,14,14,"level2\\number\\4_on.bmp");
  nDIB=nDIB+1;
  LoadBMP(nDIB,115,210,14,14,"level2\\number\\5_on.bmp");
  nDIB=nDIB+1;
  LoadBMP(nDIB,141,210,14,14,"level2\\number\\6_on.bmp");
  nDIB=nDIB+1;
  LoadBMP(nDIB,167,210,14,14,"level2\\number\\7_on.bmp");
  nDIB=nDIB+1;
  LoadBMP(nDIB,193,210,14,14,"level2\\number\\8_on.bmp");
  nDIB=nDIB+1;

  nV=GetScenarioReg(nC,0);
  LoadBMPByValue(nDIB,4,250,52,65,"level2\\icons\\curtain",nV);
  nC=nC+1;
  nDIB=nDIB+1;

  nV=GetScenarioReg(nC,0);
  LoadBMPByValue(nDIB,57,250,52,65,"level2\\icons\\door",nV);
  nC=nC+1;
  nDIB=nDIB+1;

  nV=GetScenarioReg(nC,0);
  LoadBMPByValue(nDIB,110,250,52,65,"level2\\icons\\dvd",nV);
  nC=nC+1;
  nDIB=nDIB+1;

  nV=GetScenarioReg(nC,0);
  LoadBMPByValue(nDIB,163,250,52,65,"level2\\icons\\window",nV);
  nC=nC+1;
  nDIB=nDIB+1;
  LoadBMP(nDIB,205,306,10,8,"level2\\right.bmp");
  nDIB=nDIB+1;
}
else if (g_CurPage==1)
{
  nV=GetScenarioReg(nC,1);
  LoadBMPByValue(nDIB,8,260,46,50,"level2\\icons\\projector",nV);
  nC=nC+1;
  nDIB=nDIB+1;
```

```
  nV=GetScenarioReg(nC,1);
  LoadBMPByValue(nDIB,59,260,46,50,"level2\\icons\\smoke",nV);
  nC=nC+1;
  nDIB=nDIB+1;

  nV=GetScenarioReg(nC,1);
  LoadBMPByValue(nDIB,110,260,46,50,"level2\\icons\\speaker",nV);
  nC=nC+1;
  nDIB=nDIB+1;

  nV=GetScenarioReg(nC,1);
  LoadBMPByValue(nDIB,161,260,46,50,"level2\\icons\\TV",nV);
  nC=nC+1;
  nDIB=nDIB+1;
  LoadBMP(nDIB,1,306,10,8,"level2\\left.bmp");
  nDIB=nDIB+1;
}
```

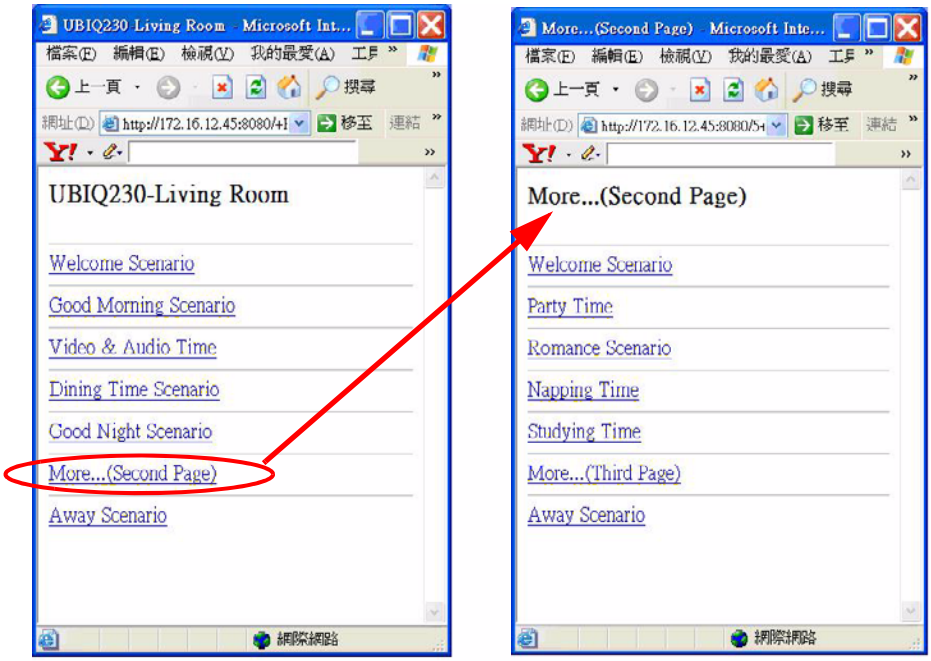
3.3 Web-enabled UI Access

There is one default home page present while you connect UbiQ-230 by Internet Browser. UbiQ-230 device provides three-modes to be accessed by remote client IE. These modes are “Text Mode”, “Screen Mode” and “Configuration Mode”. If you would like to access UbiQ-230 device by Internet Browser, please specify the IP address of UbiQ-230 device and assign the socket port as 8080. For example, If the IP address of UbiQ-230 device is 172.16.12.45, then you can launch IE for the following address:

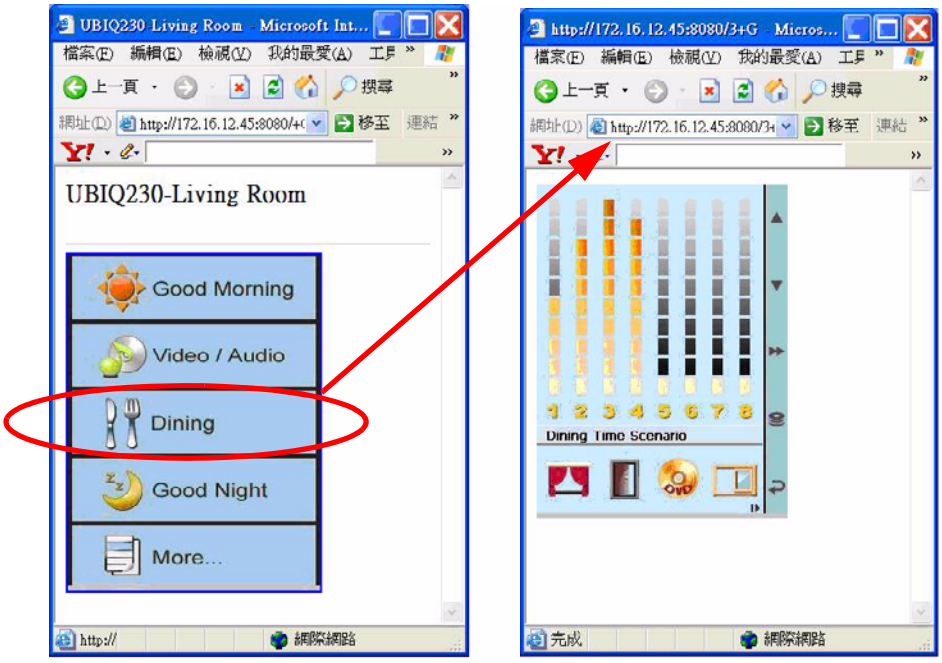
http://172.16.12.45:8080/



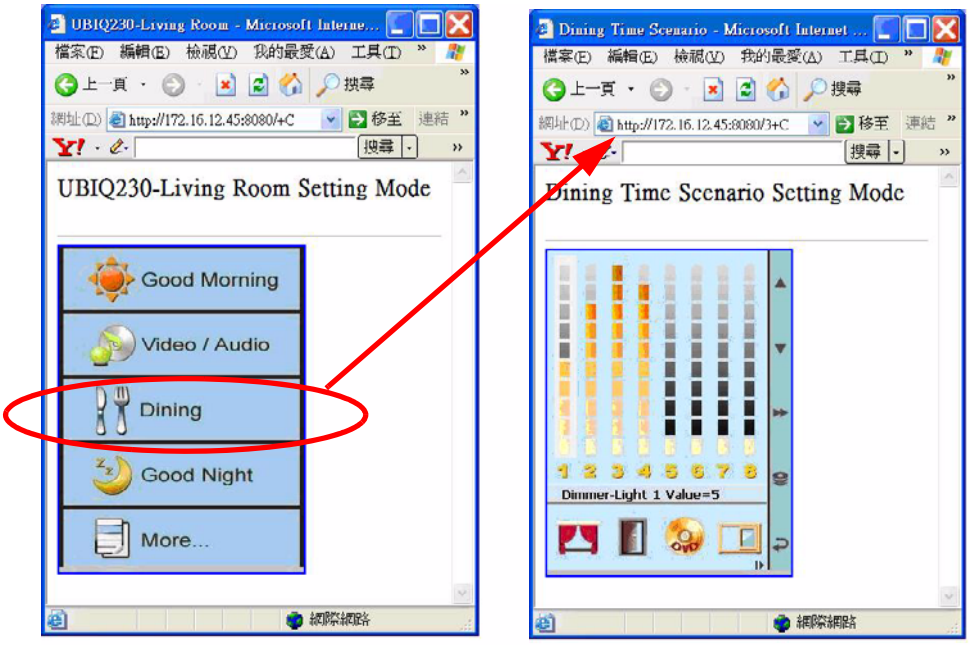
“Text Mode” will display all scenarios with text in the IE window.



“Screen Mode” will show graphic display of UbiQ-230 in the IE window.



“Configuration Mode” will show graphic display of UbiQ-230 and allow user adjusting the values of each circuit and DO signals.



Chapter 4

Basic of Smart-C Script Language

4.1 Elements of C

This section describes the organizations of the Smart-C script programming language, including the names, numbers, and characters used to construct a C program. The ANSI C syntax labels these components “tokens.” This section explains how to define tokens and how the interpreter evaluates them.

The following topics are discussed:

- Tokens
- Comments
- Keywords
- Identifiers
- Constants
- String literals
- Punctuation and special characters

4.1.1 Tokens

In a C source program, the basic element recognized by the interpreter is the “token.”. A token is source-program text that the interpreter does not break down into component elements.

Syntax

token:

- keyword
- identifier
- constant
- string-literal
- operator
- punctuator

The keywords, identifiers, constants, string literals, and operators described in this section are examples of tokens. Punctuation characters such as brackets ([ ]), braces ({ }), parentheses (( )), and commas (,) are also tokens.

4.1.2 Comments

A “comment” is a sequence of characters beginning with a forward slash/asterisk combination (/*) that is treated as a single white-space character by the interpreter and is otherwise ignored. A comment can include any combination of characters from the representable character set, including newline characters, but excluding the “end comment” delimiter (*/). Comments can occupy more than one line but cannot be nested.

Comments can appear anywhere a white-space character is allowed. Since the interpreter treats a comment as a single white-space character, you cannot include comments within tokens. The interpreter ignores the characters in the comment. Use comments to document your code. This example is a comment accepted by the interpreter:

```
/* Comments can contain keywords such as
   for and while without generating errors. */
```

Comments can appear on the same line as a code statement:

```
printf( "Hello\n" );  /* Comments can go here */
```

The interpreter also supports single-line comments preceded by two forward slashes (//). Comments beginning with two forward slashes (//) are terminated by the next newline character that is not preceded by an escape character.

```
printf( "Hello\n" );  // Comments can go here
```

4.1.3 Keywords

“Keywords” are words that have special meaning to the C interpreter. An identifier cannot have the same spelling and case as a C keyword. The Smart-C language uses the following keywords:

break	char	continue	do	double
else	if	for	int	return
string	variant	while	void	

You cannot redefine keywords.

4.1.4 Constants

A “constant” is a number, character, or character string that can be used as a value in a program. Use constants to represent floating-point, integer, enumeration, or character values that cannot be modified.

Syntax

constant :

- floating-point-constant
- integer-constant
- character-constant

Constants are characterized by having a value and a type.

4.1.5 Hex-decimal Integer Constant

Syntax

Hex-decimal-constant:

- 0xHex-Integer Opt or
- 0XHex-Integer Opt

4.1.6 String literals

A “string litera” is a sequence of characters from the source character set enclosed in double quotation marks (" ") or single quotation marks (°Æ°Ø). String literals are used to represent a sequence of characters which, taken together, form a null-terminated string.

Syntax

string-literal :

- "char-sequence opt" or
- ‘char-sequence opt’

The backslash (\) must be followed with a second backslash (\\) when it appears within a string.

4.1.7 Punctuation and special characters

The punctuation and special characters in the C character set have various uses, from organizing program text to defining the tasks that program carries out. They do not specify an operation to be performed.

Syntax

punctuator : one of

```
[] () {} = ;
```

These characters have special meanings in C.

4.2 Program Structure

This section gives an overview of C programs and program execution. Terms and features important to understanding C programs and components are also introduced. Topics discussed include:

- The main function and program execution
- Name spaces

4.2.1 The main function and program execution

Every C program has a primary (main) function that must be named main. The main function serves as the starting point for program execution. It usually controls program execution by directing the calls to other functions in the program. A program usually stops executing at the end of main, although it can terminate at other points in the program for a variety of reasons. At times, perhaps when a certain error is detected, you may want to force the termination of a program. To do so, use the return keyword.

Functions within the source program perform one or more specific tasks. The main function can call these functions to perform their respective tasks. When main calls another function, it passes execution control to the function, so that execution begins at the first statement in the function. A function returns control to main when a return statement is executed or when the end of the function is reached.

You can declare any function, except for main(), to have parameters. The term “parameter” or “formal parameter” refers to the identifier that receives a value passed to a function. When one function calls another, the called function receives values for its parameters from the calling function. These values are called “arguments”.

4.2.2 Name spaces

The interpreter sets up “name spaces” to distinguish between the identifiers used for different kinds of items. The names within each name space must be unique to avoid conflict, but an identical name can appear in more than one name space. This means that you can use the same identifier for two or more different items, provided that the items are in different name spaces. The interpreter can resolve references based on the syntactic context of the identifier in the program.

This list describes the name spaces used in C.

4.2.2.1 Functions of controllers

Function names are allocated in name spaces associated with each controller. That is, the same identifier can be a component name in any number of controllers at the same time. Definitions of component names always occur within controller’s name. Uses of component names always immediately follow the member-selection operators (.). The name of a function must be unique within the controller, but it does not have to be distinct from other names in the program.

4.2.2.2 Ordinary identifiers

All other names fall into a name space that includes variables, functions (including formal parameters and local variables), and enumeration constants. Identifier names have nested visibility, so you can redefine them within blocks.

4.3 Declarations and Types

This section describes the declaration and initialization of variables, functions, and types. The C language includes a standard set of basic data types. The following topics are discussed:

- Overview of declarations
- Type specifiers

4.3.1 Overview of declarations

A “declaration” specifies the interpretation and attributes of a identifier. A declaration that also causes storage to be reserved for the object or function named by the identifier is called a “definition.” C declarations for variables, functions, and types have this syntax:

Syntax

declaration :

type_specifiers declarator ;

Declarations are made up of some combination of type specifiers and declarators.

In the general form of a variable declaration, type-specifier gives the data type of the variable. The declarator gives the name of the variable, possibly modified to declare an array. For example,

int fp[20];

declares a variable named fp as a 20-index array to int value.

A declaration must have one declarator. Declarators provide any remaining information about an identifier. A declarator is an identifier that can be modified with brackets ([ ]), or parentheses (( )) to declare an array, or function type, respectively. When you declare simple variables (such as character, integer, and floating-point items), the declarator is just an identifier.

4.3.2 Type specifiers

Type specifiers in declarations define the type of a variable or function declaration.

Syntax

type-specifier :

- void
- char
- int
- double
- string
- variant

The keyword void specify a function return type. You can use the void type to declare functions that return no value.

4.4 Expressions and Assignments

This section describes how to form expressions and to assign values in the C language. Constants, identifiers, strings, and function calls are all operands that are manipulated in expressions. The C language has all the usual language operators. This section covers those operators as well as operators that are unique to C. The topics discussed include:

- Operators
- Operator precedence

4.4.1 Operators

There are three types of operators. A unary expression consists of a unary operator prepended to an operand. The expression can be either the name of a variable or a cast expression. If the expression is a cast expression, it must be enclosed in parentheses. A binary expression consists of two operands joined by a binary operator. A ternary expression consists of three operands joined by the conditional-expression operator.

C includes the following unary operators:

Symbol	Name
-	Negation operators
+	Unary plus operator

Binary operators associate from left to right. C provides the following binary operators:

Symbol	Name
* / %	Multiplicative operators
+ -	Additive operators
< > <= >= == !=	Relational operators
& |	Bitwise operators
&& ||	Logical operators
<< >>	Bit-Shift operators

The conditional-expression operator has lower precedence than binary expressions and differs from them in being right associative.

Expressions with operators also include assignment expressions, which use unary or binary assignment operators. The binary assignment operators are the simple-assignment operator (=) and the compound-assignment operators. Each compound-assignment operator is a combination of another binary operator with the simple-assignment operator.

4.4.2 Operator precedence

The precedence and associativity of C operators affect the grouping and evaluation of operands in expressions. An operator’s precedence is meaningful only if other operators with higher or lower precedence are present. Expressions with higher-precedence operators are evaluated first.

Following table summarizes the precedence and associativity (the order in which the operands are evaluated) of C operators, listing them in order of precedence from highest to lowest. Where several operators appear together, they have equal precedence and are evaluated according to their associativity. The operators in the table are described in the sections beginning with Postfix Operators. The rest of this section gives general information about precedence and associativity.

Symbol
Type of Operation
Associativity
[ ] ( ) .
Expression
Left to right
+ ®C
Unary
Right to left
* / %
Multiplicative
Left to right
+ ®C
Additive
Left to right
< > <= >= == !=
Relational
Left to right
& | && || << >>
Logical and Bitwise
Left to right

An expression can contain several operators with equal precedence. When several such operators appear at the same level in an expression, evaluation proceeds according to the associativity of the operator, either from right to left or from left to right. The direction of evaluation does not affect the results of expressions that include more than one multiplication (*), addition (+), or binary-bitwise (& |) operator at the same level.

Statements

The statements of a C program control the flow of program execution. In C, as in other programming languages, several kinds of statements are available to perform loops, to select other statements to be executed, and to transfer control. Following a brief overview of statement syntax, this section describes the C statements in alphabetical order:

* block statement
* break statement
* continue statement
* expression statement
* for statement
* if statement
* return statement
* while statement

block statement

A block statement typically appears as the body of another statement, such as the if statement.

Syntax

block-statement :

{ declaration-list statement-list }

declaration-list :
declaration or
declaration-list declaration

statement-list :
°°°°°°°°°°statement or
°°°°°°°°°°statement-list statement

If there are declarations, they must come before any statements. The scope of each identifier declared at the function including the block-statement.
This example illustrates a compound statement:

```
if ( i > 0 )
{
   line[i] = x;
   x=x+1;
   i=i-1;
}
```

In this example, if i is greater than 0, all statements inside the compound statement are executed in order.

break statement
The break statement terminates the execution of the nearest enclosing for, or while statement in which it appears. Control passes to the statement that follows the terminated statement.

Syntax

break-statement :

break;

Within nested statements, the break statement terminates only the for, or while statement that immediately encloses it. You can use a return statement to transfer control elsewhere out of the nested structure.

This example illustrates the break statement:

```
for ( i = 0; i < LENGTH; i=i+1 )  /* Execution returns here when */
{
  if ( lines[i] == 0 )
  {
    nLen = i;
    break;              /* break statement is executed */
  }
}
```

The example processes an array of variable-length strings stored in lines. The break statement causes an exit from the interior for loop after the terminating null character (0) of each string is found and its position is stored in nLen.

continue statement

The continue statement passes control to the next iteration of the for, or while statement in which it appears, bypassing any remaining statements in the for, or while statement body. A typical use of the continue statement is to return to the start of a loop from within a deeply nested loop.

Syntax

continue-statement :

continue;

The next iteration of a for, or while statement is determined as follows:

* Within a while statement, the next iteration starts by reevaluating the expression of the while statement.
* A continue statement in a for statement causes the first expression of the for statement to be evaluated. Then the interpreter reevaluates the conditional expression and, depending on the result, either terminates or iterates the statement body.

This is an example of the continue statement:

```
while ( i > 0 )
{
   x = sum( i );
   if ( x == 1 )
      continue;
   y= y+x * x;
   i=i-1;
}
```

In this example, the statement body is executed while i is greater than 0. First sum(i) is assigned to x; then, if x is equal to 1, the continue statement is executed. The rest

of the statements in the body are ignored, and execution resumes at the top of the loop with the evaluation of the loop°Øs test.

expression statement

When an expression statement is executed, the expression is evaluated according to the rules outlined in section: Expressions and Assignments.

Syntax

expression-statement :

    expressions ;

An empty expression statement is called a null statement.

These examples demonstrate expression statements.

```
x = ( y + 3 );          /* x is assigned the value of y + 3  */
x=°±abcdefg°±;          /* x is assigned to °×abcdefg°±     */
y=sum(x );       /* Function call returning value    */
z=0xA8;              /* z is assigned to the hex value 0xA8 */
```

for statement

The for statement lets you repeat a statement or compound statement a specified number of times. The body of a for statement is executed zero or more times until an optional condition becomes false. You can use optional expressions within the for statement to initialize and change values during the for statement°Øs execution.

Syntax

for-statement :

for (init-expression ; cond-expression ; loop-expression)
  block-statement

Execution of a for statement proceeds as follows:

* The init-expression, if any, is evaluated. This specifies the initialization for the loop. There is no restriction on the type of init-expression.

* The cond-expression, if any, is evaluated. This expression must have arithmetic. It is evaluated before each iteration. Three results are possible:

(1) If cond-expression is true (nonzero), statement is executed; then loop-expression, if any, is evaluated. The loop-expression is evaluated after each iteration. There is no restriction on its type. The process then begins again with the evaluation of cond-expression.

(2) If cond-expression is omitted, cond-expression is considered true, and execution proceeds exactly as described in the previous paragraph.

(3) If cond-expression is false (0), execution of the for statement terminates and control passes to the next statement in the program.

A for statement also terminates when a break or return statement within the statement body is executed. A continue statement in a for loop causes loop-expression to be evaluated. When a break statement is executed inside a for loop, loop-expression is not evaluated or executed.

This example illustrates the for statement:

```
for ( i = 0; i < MAX; i=i+1 )
{
   if ( line[i] == '\n' )
      new_line=new_lines+1;
}
```

First i is initialized to 0. Then i is compared with the constant MAX; if i is less than MAX, the statement body is executed. Depending on the value of line[i], the body of one or neither of the if statements is executed. Then i is incremented and tested against MAX; the statement body is executed repeatedly as long as i is less than MAX.

if statement

The if statement controls conditional branching. The body of an if statement is executed if the value of the expression is nonzero. The syntax for the if statement has two forms.

Syntax

if-statement :

if (expression) block-statement

if (expression) block-statement else block-statement

In the first form of the syntax, if expression is true (nonzero), block-statement is executed. If expression is false, block-statement is ignored. In the second form of syntax, which uses else, the second block-statement is executed if expression is false. With both forms, control then passes from the if statement to the next statement in the program unless one of the statements contains a break, continue.

The following are examples of the if statement:

```
if ( i > 0 ) {
   y = x / i;
}
else
{
   x = i;
```

```
    y = f( x );
}
```

In this example, the statement y = x/i; is executed if i is greater than 0. If i is less than or equal to 0, i is assigned to x and f(x) is assigned to y. Note that the statement forming the if clause ends with a semicolon.

return statement

The return statement terminates the execution of a function and returns control to the calling function. Execution resumes in the calling function at the point immediately following the call. A return statement can also return a value to the calling function.

Syntax

return-statement :

```
    return(expression) ;
```

The value of expression, if present, is returned to the calling function. If expression is omitted, the return value of the function is undefined. The expression, if present, is converted to the type returned by the function. If the function was declared with return type void, a return statement containing an expression generates a warning and the expression is not evaluated.

If no return statement appears in a function definition, control automatically returns to the calling function after the last statement of the called function is executed. In this case, the return value of the called function is undefined. If a return value is not required, declare the function to have void return type; otherwise, the default return type is int.

This example demonstrates the return statement:

```
int sum( int num );
int main()
{
    int nSum;
    nSum=sum(100);
    printf(°×The sum(100)=%d°±,nSum);
}

/* Sum the values between 0 and num. */
sum(int num)
{
  int running_sum;
  running_sum = 0;
  while(num>0) {
    running_sum = running_sum + num;
    num = num - 1;
```

```
  }
  return(running_sum);
}
```

In this example, the main function calls one function: sum. The sum function returns the sum from 1 to num, where the return value is assigned to nSum.

while statement

The while statement lets you repeat a statement until a specified expression becomes false.

Syntax

while-statement :

    while (expression) block-statement

The expression must have arithmetic type. Execution proceeds as follows:

* The expression is evaluated.
* If expression is initially false, the body of the while statement is never executed, and control passes from the while statement to the next statement in the program. If expression is true (nonzero), the body of the statement is executed and the process is repeated beginning at step 1.

The while statement can also terminate when a break, or return within the statement body is executed. Use the continue statement to terminate an iteration without exiting the while loop. The continue statement passes control to the next iteration of the while statement.

This is an example of the while statement:

```
while(num>0) {
  running_sum = running_sum + num;
  num = num - 1;
}
```

This example adds running_sum from 1 to num. If num is greater than 0, running_sum added by num. When num reaches 0, execution of the while statement terminates.

Functions

The function is the fundamental modular unit in C. A function is usually designed to perform a specific task, and its name often reflects that task. A function contains declarations and statements. This section describes how to declare, define, and call C functions. Other topics discussed are:

* Overview of functions

* Return type
* Arguments

Overview of functions

Functions must have a definition and should have a declaration, although a definition can serve as a declaration if the declaration appears before the function is called. The function definition includes the function body °™ the code that executes when the function is called.

A function call passes execution control from the calling function to the called function. The arguments, if any, are passed by value to the called function. Execution of a return statement in the called function returns control and possibly a value to the calling function.

Return type

The return type of a function establishes the size and type of the value returned by the function and corresponds to the type-specifier. The type-specifier can specify any fundamental type. If you do not include type-specifier, the return type int is assumed.

The return type given in the function definition must match the return type in declarations of the function elsewhere in the program. A function returns a value when a return statement containing an expression is executed. The expression is evaluated, converted to the return value type if necessary, and returned to the point at which the function was called. If a function is declared with return type void, a return statement containing an expression generates a warning and the expression is not evaluated.

The following examples illustrate function return values.

```
/* Sum the values between 0 and num. */
sum(int num)
{
  int running_sum;
  running_sum = 0;
  while(num>0) {
    running_sum = running_sum + num;
    num = num - 1;
  }
  return(running_sum);
}
```

You need not declare functions with int return type before you call them, although prototypes are recommended so that correct type checking for arguments and return values is enabled.

Arguments

The arguments in a function call have this form:

```
expression ( expression-list ) /* Function call */
```

In a function call, expression-list is a list of expressions (separated by commas). The values of these latter expressions are the arguments passed to the function. If the function takes no arguments, expression-list should contain the keyword void.

An argument can be any value with fundamental type. All arguments are passed by value. This means a copy of the argument is assigned to the corresponding parameter. The function does not know the actual memory location of the argument passed. The function uses this copy without affecting the variable from which it was originally derived.

The expression-list in a function call is evaluated and the usual arithmetic conversions are performed on each argument in the function call. If a prototype is available, the resulting argument type is compared to the prototype°Øs corresponding parameter. If they do not match, either a conversion is performed, or a diagnostic message is issued.

The number of expressions in expression-list must match the number of parameters, unless the function°Øs prototype or definition explicitly specifies a variable number of arguments. In this case, the interpreter checks as many arguments as there are type names in the list of parameters and converts them, if necessary, as described above.

If the prototype°Øs parameter list contains only the keyword void, the interpreter expects zero arguments in the function call and zero parameters in the definition. A diagnostic message is issued if it finds any arguments.

The following examples illustrate the function argument.

```
int main()
{
    int nSum;
    nSum=sum(100);
    printf(°×The sum(100)=%d°±,nSum);
}

/* Sum the values between 0 and num. */
sum(int num)
{
  int running_sum;
  running_sum = 0;
  while(num>0) {
    running_sum = running_sum + num;
    num = num - 1;
  }
  return(running_sum);
}
```

In this example, the sum function is declared in main to have one argument, represented by the identifier num, is an int values.

Chapter 5

Functions Reference

5.1 Summary Tables

The following table summaries the global variables and functions that belong to the Smart-C Script language. Global variables and functions are grouped by tasks you might wish to perform.

Global Variables:

Name	Description
TRUE	Its value is 1
FALSE	Its value is 0
FILE_READ	Be used by the open function
FILE_RDWR	Be used by the open function
FILE_CREATE_NEW	Be used by the open function
FILE_CREATE_RW	Be used by the open function
SEEK_BEGIN	Be used by the seek function
SEEK_END	Be used by the seek function
SEEK_CURRENT	Be used by the seek function
g_nVar1	Global variable, its type is int, initialized to 0, can be used in your programs
g_nVar2	Global variable, its type is int, initialized to 0, can be used in your programs
g_nVar3	Global variable, its type is int, initialized to 0, can be used in your programs
g_nVar4	Global variable, its type is int, initialized to 0, can be used in your programs
g_nVar5	Global variable, its type is int, initialized to 0, can be used in your programs
g_fVar1	Global variable, its type is double, initialized to 0, can be used in your programs
g_fVar2	Global variable, its type is double, initialized to 0, can be used in your programs
g_sVar1	Global variable, its type is string, initialized to °×°±, can be used in your programs
g_sVar2	Global variable, its type is string, initialized to °×°±, can be used in your programs
g_bConfig	Global variable, its type is integer, and set to 1 when entering the configuration mode
g_ConfigPage	Global variable, its type is integer, and set to total configuration pages on the configuration mode
g_CurPage	Global variable, its type is integer, and set to current configuration page on the configuration mode
CLOCK_SHOW_ID	Be used by the LoadBMP function
CLOCK_UNSHOW_ID	Be used by the LoadBMP function
SCREEN_REDRAW_ID	Be used by the LoadBMP function
g_nComTimeout	Global variable, its type is integer.
g_nCurIndex	Global variable, its type is integer. It specifies the current index on the configuration mode
g_nPortID	Global variable, its type is integer.
g_nRS485ID	Global variable, its type is integer.
g_nRS485ID2	Global variable, its type is integer.
g_nRS485ID3	Global variable, its type is integer.
g_nRS485ID4	Global variable, its type is integer.

g_nIDLETime	Global variable, its type is integer. It specifies the IDLE time to enter the idle mode
g_nReturnTime	Global variable, its type is integer. It specifies the return time to main menu.

FILE I/O Functions:

Name	Description
open	Open a file
close	Close the open file
read	Read data from a file
write	Write data to a file
seek	Move file pointer
readln	Read a line from a file
writeln	Write a line to a file
eof	Test whether or not end-of-file of a file
filecopy	Copy a file to another file

Data acquisition functions:

Name	Description
OpenPort	Open the specified COM port
sendComData	Send data to the specified COM port

String functions:

Name	Description
strlen	Return the length of a string
strcpy	Copy a string to another string
sprintf	Format the output to the string
itoa	Convert a integer to a string
ftoa	Convert a double to a string
atoi	Convert a string to integer by decimal type
atoh	Convert a string to integer by hexadecimal type
atof	Convert a string to a double

Debug functions:

Name	Description
debug	Show the output to the debug window
setdebug	Switch the debug mode
printf	Format the output to the debug window or web-client

Homepage client functions:

Name	Description
printf	Format the output to the debug window or web-client
getenv	Get the value of the environment variable of web-client

Date/Time functions:

Name	Description
date	Get the current date string
time	Get the current time string

Display functions:

Name	Description
LoadBMP	Display the specified file on screen
LoadBMPByValue	Display the specified file by assigned-value on screen
ShowText	Display the notified text on screen
HelpWindow	Assign the help caption to the specified coordinate on screen
SetupTime	Display and provide the interface to setup the time on screen

Scenario Data functions:

Name	Description
GetScenarioReg	Get item value on the current scenario registry.
SetScenarioInit	Set the initialization value of the specified scenario index

Voice functions:

Name	Description
PlaySound	Play the specified sound wave file

Ethernet functions:

Name	Description
GetFileFromHttp	Get one file from specified homepage

5.2 Support Functions

5.2.1 atof

Syntax

double atof(char* pDest) or
double atof(string sDest)

Parameters

pDest: Specifies the Null-ended character array
sDest: Specifies a string variable.

Description

This function returns the converted double value of the string.

Example

```
// The example will convert a string to a double value and show them
char pChar[20];  // declare pDest as char[20]
double fValue;

setdebug(TRUE);
pChar="23.4567";
fValue=atof(pChar);
debug("atof(",pChar,)=",fValue); // It will show "atof(23.4567)=23.4567"
```

See Also

itoa, ftoa, atoi, atoh

5.2.2 atoh

Syntax

int atoh(char* pDest) or
int atoh(string sDest)

Parameters

pDest: Specifies the Null-ended character array
sDest: Specifies a string variable.

Description

This function returns the converted value of the string based on hexadecimal.

Example

```
 // The example will convert a string to a integer and show them
char pChar[20];  // declare pDest as char[20]
int nI;

setdebug(TRUE);
pChar="20A";
nI=atoh(pChar);
debug("atoh(",pChar,)=",nI); // It will show "atoh(20A)=522"
```

See Also

itoa, ftoa, atof, atoi

5.2.3 atoi

Syntax

int atoi(char* pDest) or

int atoi(string sDest)

Parameters

pDest: Specifies the Null-ended character array

sDest: Specifies a string variable.

Description

This function returns the converted value of the string.

Example

```
// The example will convert a string to a integer and show them
char pChar[20];  // declare pDest as char[20]
int nI;

setdebug(TRUE);
pChar="2310";
nI=atoi(pChar);
debug("atoi(",pChar,)=",nI); // It will show "atoi(2310)=2310"
```

See Also

itoa, ftoa, atof, atoh

5.2.4 close

Syntax

void close(int hFile)

Parameters

hFile: Specifies the handle referring to open file

Description

This function closes the file associated with handle.

Example

```
int hFile;
hFile=open("\\Flash Storage\\template.txt", FILE_CREATE_RW);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");

}
else {
// process the file

close(hFile); // close the file
}
```

See Also

open, read, write, seek, readln, writeln, filecopy, eof

5.2.5 date

Syntax

string date()

Parameters

none

Description

This function returns the current date, its format is fixed as “yyyy/mm/dd”.

Example

```
string sDate;
setdebug(TRUE);
sDate=date();
debug(“The current date is”,sDate);
```

See Also

time

5.2.6 debug

Syntax

void debug(variant msg,...)

Parameters

msg: the messages will be shown on the debug window.

... : more variant variables to be shown.

Description

This function will show the values of the argument in the debug window, if the mode of the debug is TRUE.

Comments

The arguments are variant on type and numbers. You can add any type or numbers of variables to this function, but at least one argument for it.

Example

```
// The example will show the pChar and f1 in the debug window
char pChar[20];  // declare pChar as char[20]
double f1; // declare f1 as double

f1=2.789;
pChar=”The value of f1 is “;
setdebug(TRUE);
debug(pChar,f1); // It will show “The value of f1 is 2.789”
```

See Also

setdebug, printf

5.2.7 eof

Syntax

int eof(int hFile)

Parameters

hFile: Specifies the handle referring to open file

Description

This function returns whether or not the current file position is the end-of-file. If the handle is invalid, or the file is not open for reading, or the file is locked, the function returns -1.

Example

```
int hFile;
hFile=open("\\flash storage\\template.txt", FILE_CREATE_RW);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");

}
else {
// process the file
readln(hFile, data);
while (!eof(hFile)) {
readln(hFile,data);
}
close(hFile); // close the file
}
```

See Also

open, close, read, write, seek, readln, writeln, filecopy

5.2.8 filecopy

Syntax

int filcopy(char* pSrc,char *pDest) or

int filcopy(string sSrc, string sDest)

Parameters

pSrc: Specifies the source filename array

pDest: Specifies the destination filename array

sSrc: Specifies the source filename string

sDest: Specifies the destination filename string

Description

This function returns the result of copying source file to destination file. If it successes, returns TRUE; otherwise returns FALSE.

Example

```
ilecopy("\\flash storage\\program.c1","\\flash storage\\program.cgi");
```

See Also

open, close, read, write, seek, readln, writeln, eof

5.2.9 ftoa

Syntax

int ftoa(double fValue, char* pDest, int nPrecision)

Parameters

FValue: the double value will be converted

pDest: Specifies the Destination array

nPrecision: Specifies the number of digits to be stored after the decimal point.

Description

This function returns the actually length of the converted string.

Example

```
// The example will convert a double to string and show them
char pDest[20];  // declare pDest as char[20]
dobuble fValue;
int nLen;

setdebug(TRUE);
fValue=23.4567
nLen=ftoa(fValue,pDest,3);
debug("ftoa(",fValue,"pDest,3)=",pDest);
```

It will show "ftoa(23.4567, pDest, 3)=23.457

See Also

itoa, atoi, atof, atoh

5.2.10 GetScenarioReg

Syntax

int GetScenarioReg (int nChanneNol, int nPageNo)

Parameters

nChannelNo: Specifies the index of current scenario items, the value is from 0 to 20.

nPageNo: Specifies a given index of the display page, the value is from 0 to 2.

Description

This function returns the value of the current scenario registry by the given item and page index.

Example

```
Int nV;
// Get the current scenario registry for item 1 and page 0
nV=GetScenarioReg(1,0);
```

See Also

SetScenarioInit

5.2.11 GetFileFromHttp

Syntax

void GetFileFromHttp(string sHttp, string sDesFile)

Parameters

sHttp: Specifies the homepage file address.

sDestFile: Specifies the destination file on the device side. It should include the path name.

Description

This function will get the file from the specified homepage address if the homepage exist.

Example

```
string sHttp;
string sDestFile;
sHttp="http://taiwan.advantech.com.tw/unzipfunc/Unzip/EH-7105_ds.pdf";
sDestFile="\\EH7105.pdf";
GetFileFromHttp(sHttp,sDestFile);
```

See Also

none

5.2.12 getenv

Syntax

string getenv(char* pEnv) or

string getenv(string sEnv)

Parameters

pEnv: Specifies the Null-ended environment variable.

sEnv: Specifies the environment variable string.

Description

This function returns the string value to the given environment variable.

Comments

This function just is used for writing CGI script on the WebCON kernel.

Example

```
string sButton;
sButton=getenv("BUTTON");
printf("BUTTON=%s",sButton);
```

See Also

printf

5.2.13 HelpWindow

Syntax

void HelpWindow(int nStartX, int nStartY, int nWidth, int nHeight)

Parameters

nStartX: Specifies the beginning x-coordinate of display area.

nStartY: Specifies the beginning y-coordinate of display area.

nWidth: Specifies the width of display area.

nHeight: Specifies the height of display area.

Description

This function set the help window on the specified coordinate when entering configuration mode.

Example

```
HelpWindow(10,238,200,17);
```

See Also

LoadBMP, LoadBMPByValue, ShowText

5.2.14 itoa

Syntax

int itoa(int nValue, char* pDest, int nBase)

Parameters

nValue: the value will be converted

pDest: Specifies the Destination array

nBase: Specifies the base number, it will be 10 or 16.

Description

This function returns the actually length of the converted string.

Example

```
// The example will convert a integer to string and show them
char pDest[20];  // declare pDest as char[20]
int nI;
int nLen;

setdebug(TRUE);
nI=1024;
nLen=itoa(nI,pDest,10);
debug("itoa(",nI,"pDest,10)=",pDest); // It will show "itoa(1024,pDest,10)=1024"
itoa(nI,pDest,16);
debug("itoa("nI,"pDest,16)=",pDest); // It will show "itoa(1024,pDest,16)=400"
```

See Also

ftoa, atoi, atof, atoh

5.2.15 LoadBMP

Syntax

void LoadBMP(int nDIBID, int nStartX, int nStartY, int nWidth, int nHeight, string sFilename)

Parameters

nDIBID: Specifies the display ID. Its value is from 0 to 32. There are some values for default function described as follows:

CLOCK_SHOW_ID: show the clock on the screen.

CLOCK_UNSHOW_ID: hide the clock on the screen.

SCREEN_REDRAW_ID: force the display redrawing.

nStartX: Specifies the beginning x-coordinate of display area.

nStartY: Specifies the beginning y-coordinate of display area.

nWidth: Specifies the width of display area.

nHeight: Specifies the height of display area.

sFilename: Specifies the bitmap filename. This file needs to be a bitmap or jpeg file format. If the filename is beginning with °×\\°±, then the file will be search from root path. Otherwise, the file will be search from bitmap folder of current application path.

Description

This function will put the specified bitmap file stretched on the assigned display area.

Example

```
// Show the clock on the screen
LoadBMP(CLOCK_SHOW_ID,0,0,240,320,"");
...
// Load the background bitmap from current bitmap folder on the screen
LoadBMP(0,0,0,240,320,"level2\\level2.jpg");
// Load help background to the display area (0,238),(218, 238+18)
LoadBMP(1,0,238,218,18,"level2\\help.jpg");
// Load the icon background to display area (0,256),(218,256+64)
LoadBMP(2,0,256,218,64,"level2\\icon_bg.jpg");
```

See Also

LoadBMPByValue

5.2.16 LoadBMPByValue

Syntax

void LoadBMPByValue(int nDIBID, int nStartX, int nStartY, int nWidth, int nHeight, string sFilename, int nValue)

Parameters

nDIBID: Specifies the display ID. Its value is from 0 to 32.

nStartX: Specifies the beginning x-coordinate of display area.

nStartY: Specifies the beginning y-coordinate of display area.

nWidth: Specifies the width of display area.

nHeight: Specifies the height of display area.

sFilename: Specifies the bitmap filename. This file needs to be a bitmap or jpeg file format. The file will be search from bitmap folder of current application path. If the sFilename is equal to "==LIGHT_BAR2", then the screen will show the default pictures for lighting control.

nValue: Specifies the value associated with sFilename.

Description

This function will put the specified bitmap file by given value stretched on the assigned display area.

Example

```
int nV;
nV=GetScenarioReg(0,0);
LoadBMPByValue(3,25,4,18,206,"==LIGHT_BAR2",nV);
...
nV=GetScenarioReg(4,0);
LoadBMPByValue(11,8,260,46,50,"level2\\icons\\curtain",nV);
```

See Also

LoadBMPByValue, GetScenarioReg, SetScenarioInit

5.2.17 OpenPort

Syntax

void OpenPort(int nPort, int nBaudRate)

Parameters

nPort: Specifies the COM port ID. The UbiQ-230 device RS485 port is set to COM3:.

nBaudRate: Specifies the baud rate of COM Port. The UbiQ-230 device for RS485 port is set to 19200 bps.

Description

This function will open a COM port by specified baud rate.

Example

```
g_nPortID=3;
OpenPort(g_nPortID,19200);
```

See Also

sendComData

5.2.18 open

Syntax

int open(char* pFilename,int open_type) or

int open(string sFilename,int open_type)

Parameters

pFilename: Specifies the Null-ended character array filename. A array that is the path to the desired file. The path must be absolute.

sFilename: Specifies the filename string. A string that is the path to the desired file. The path must be absolute.

Description

This function returns a file handle for the opened file. A return value of 0 indicates an error.

Comments

The open function opens the file specified by filename and prepares the file for reading or writing, as specified by open_flag:

open_flag	description
FILE_READ	Opens the file for reading only
FILE_RDWR	Opens the file for reading and writing
FILE_CREATE_NEW	Create a new file. If the file exists already, it is truncated to 0 length.
FILE_CREATE_RW	Create a new file. If the file being created already exists, it is not truncated to 0 length. Thus the file is guaranteed to open, either as a newly created file or as an existing file. This might be useful, for example, when opening a settings file that may or may not exist already.

Example

```
int hFile;
hFile=open("\\flash storage\\template.txt", FILE_CREATE_RW);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");

}
else {

close(hFile);
}
```

See Also

read, write, close, seek, readln, writeln, filecopy, eof

5.2.19 PlaySound

Syntax

void PlaySound(string sWaveFile)

Parameters

sWaveFile : Specifies the wave filename. This file needs to be a wave format. If the filename is beginning with “\\”, then the file will be search from root path. Otherwise, the file will be search from current application path.

Description

This function will play a sound with specified wave file.

Example

PlaySound(“\\windows\\default.wav”);

See Also

none

5.2.20 printf

Syntax

void printf(char *pFormat,...) or

void printf(string sFormat,...)

Parameters

pFormat : Specifies the Null-end character array format to show on the debug window or the client of webpages

sFormat: Specifies thestring format to show on the debug window or the client of webpages

.. : more variant variables to be added into the format string

Description

The function formats and prints a series of characters and values to the debug window or the client of webpages. If arguments follow the format string, the format string must contain specifications that determine the output format for the arguments.

Comments

Format specifications always begin with a percent sign (%) and are read left to right. When printf encounters the first format specification (if any), it converts the value of the first argument after format and outputs it accordingly. The second format specification causes the second argument to be converted and output, and so on. If there are more arguments than there are format specifications, the extra arguments are ignored. The results are undefined if there are not enough arguments for all the format specifications.

A format specification, which consists of optional and required fields, has the following form:

%[flags] [width] [.precision] type

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a type character (for example, %s). If a percent sign is followed by a character that has no meaning as a format field, the character is copied to stdout. For example, to print a percent-sign character, use %%.

The optional fields, which appear before the type character, control other aspects of the formatting, as follows:

- type

  Required character that determines whether the associated argument is interpreted as a character, a string, or a number.

Character	Type	Output Format
c or C	int or char	Specifies a single-byte character.
d	int	Signed decimal integer.
i	int	Signed decimal integer.
o .	int	Unsigned octal integer
u	int	Unsigned decimal integer.
x	int	Unsigned hexadecimal integer, using “abcdef.”
X	int	Unsigned hexadecimal integer, using “ABCDEF.”
e	double	Signed value having the form [-]d.dddd e [sign]ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or -.
E	double	Identical to the e format except that E rather than e introduces the exponent.
f	double	Signed value having the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
g	double	Signed value printed in f or e format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than -4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
G	double	Identical to the g format, except that E, rather than e, introduces the exponent (where appropriate).
s or S	String	Specifies a single-byte - character string. Characters are printed up to the first null character or until the precision value is reached.

- flags

  Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.

Flag	Meaning	Default
-	Left align the result within the given field width.	Right align.
+	Prefix the output value with a sign (+ or -) if the output value is of a signed type	Sign appears only for negative signed values (-).
0	If width is prefixed with 0, zeros are added until the minimum width is reached. If 0 and ®C appear, the 0 is ignored. If 0 is specified with an integer format (i, u, x, X, o, d) the 0 is ignored.	No padding.

blank (' ')	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear.	No blank appears.
#	When used with the o, x, or X format, the # flag prefixes any nonzero output value with 0, 0x, or 0X, respectively	No blank appears.
	When used with the e, E, or f format, the # flag forces the output value to contain a decimal point in all cases.	Decimal point appears only if digits follow it.
	When used with the g or G format, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. Ignored when used with c, d, i, u, or s.	Decimal point appears only if digits follow it. Trailing zeros are truncated.

- width

  Optional number that specifies the minimum number of characters output.

  The second optional field of the format specification is the width specification. The width argument is a nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values -- depending on whether the - flag (for left alignment) is specified -- until the minimum width is reached. If width is prefixed with 0, zeros are added until the minimum width is reached (not useful for left-aligned numbers).

  The width specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if width is not given, all characters of the value are printed

- precision

  Optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.

  The third optional field of the format specification is the precision specification. It specifies a nonnegative decimal integer, preceded by a period (.), which specifies the number of characters to be printed, the number of decimal places, or the number of significant digits. Unlike the width specification, the precision specification can cause either truncation of the output value or rounding of a floating-point value. If precision is specified as 0 and the value to be converted is 0, the result is no characters output, as shown below:

```
printf( "%.0d", 0 );      /* No characters output */
```

  If the precision specification is an asterisk (*), an int argument from the argument list supplies the value. The precision argument must precede the value being formatted in the argument list.

Type	Meaning	Default
c, C	The precision has no effect.	Character is printed.
d, i, u, o, x, X	The precision specifies the minimum number of digits to be printed. If the number of digits in the argument is less than precision, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds precision.	Default precision is 1.

e, E	The precision specifies the number of digits to be printed after the decimal point. The last printed digit is rounded.	Default precision is 6; if precision is 0 or the period (.) appears without a number following it, no decimal point is printed.
f	The precision value specifies the number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.	Default precision is 6; if precision is 0, or if the period (.) appears without a number following it, no decimal point is printed.
g, G.	The precision specifies the maximum number of significant digits printed.	Six significant digits are printed, with any trailing zeros truncated
s, S	The precision specifies the maximum number of characters to be printed. Characters in excess of precision are not printed.	Characters are printed until a null character is encountered.

Example

```
// The example will show the messages in the debug window
char pChar[20];  // declare pChar as char[20]
double f1; // declare f1 as double
int nI;
string  aStr;
char pBuf[30];

f1=2.789;
nI=120;
pBuf=”Test Printf”;
aStr=”Test printf 2”;
setdebug(TRUE);
printf(“The f1=%f, nI=%d, pBuf=%s, aStr=%s”,f1,nI,pBuf,aStr);
```

See Also

sprintf, debug, setdebug, getenv

5.2.21 read

Syntax

int read(int hFile,char *buf, int nCount)

Parameters

hFile: Specifies the handle referring to open file

buf: Specifies the storage location for data

nCount: Specifies the maximum number of bytes

Description

This function returns the number of bytes read, which may be less than nCount if there are fewer than count bytes left in the file. If the function tries to read at end of file, it returns 0. If the handle is invalid, or the file is not open for reading, or the file is locked, the function returns -1.

Example

```
int hFile;
int nLen;
char data[512];
hFile=open("\\flash storage\\template.txt", FILE_CREATE_RW);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");

}
else {
// process the file
nLen=read(hFile, data, 512);
while (nLen!=512) {
nLen=read(hFile,data,512);
}
close(hFile); // close the file
}
```

See Also

open, close, write, seek, readln, writeln, filecopy, eof

5.2.22 readln

Syntax

int readln(int hFile,char *buf)

Parameters

hFile: Specifies the handle referring to open file

buf: Specifies the storage location for data

Description

This function returns the number of bytes read. This function reads a line of text and then skips to the next line of the file. The file must be a text file, in which case each carriage return®Clinefeed (CR-LF) pair or single linefeed character is replaced with a Null-ended character. If the function tries to read at end of file, it returns 0. If the handle is invalid, or the file is not open for reading, or the file is locked, the function returns -1.

Example

```
int hFile;
int nLen;
char data[512];
hFile=open("\\flash storage\\template.txt", FILE_READ);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");

}
else {
// process the file
readln(hFile, data);
while (!eof(hFile)) {
readln(hFile,data);
}
close(hFile); // close the file
}
```

See Also

open, close, read, write, seek, writeln, filecopy, eof

5.2.23 seek

Syntax

int seek(int hFile,int nFrom, int nOffset)

Parameters

hFile: Specifies the handle referring to open file

nFrom: Specifies the Pointer movement mode.

nOffset: Specifies the number of bytes to move the pointer

Description

This function returns the new byte offset from the beginning of the file. A return value of -1 indicates an error.

Comments

Pointer movement mode must be one of the following values:

SEEK_BEGIN	Move the file pointer nOffset bytes forward from the beginning of the file.
SEEK_END	Move the file pointer nOffset bytes from the end of the file. Note that nOffset must be negative to seek into the existing file; positive values will seek past the end of the file
SEEK_CURRENT	Move the file pointer lOff bytes from the current position in the file.

Example

```
int hFile;
int nLen;
char data[512];
hFile=open("\\flash storage\\template.txt", FILE_CREATE_RW);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");
}
else {
// process the file
seek(SEEK_END,0);
data="This is a test file.";
nLen=write(hFile, data, strlen(datta));
close(hFile); // close the file
}
```

See Also

open, close, read, write, readln, writeln, filecopy, eof

5.2.24 sendComData

Syntax

int sendComData (int nPort, char *OutBuf, int nOutLen, char *ReadBuf, int nReadLen, int nTimeout)

Parameters

nPort: Specifies the COM port

OutBuf: Specifies output data buffer address

nOutLen: Specifies the length of sending data

ReadBuf: Specifies receiving data buffer address

nReadLen: Specifies the length of receiving data

nTimeout: Specifies the timeout of receiving data

Description

This function sends the data to the COM nPort and waits until return data is received or timeout. The return value is the actual length of receiving data.

Example

```
char pOut[20];
char pRead[50];
char pTemp[5];
int nRet;
int i;

pOut="#011001\r";
setdebug(TRUE);
debug("Digital Out in", 0);
nRet = sendComData(2,pOut,8,pRead,2,1000); //send data to COM2
if (nRet==2) {
debug("Data is correct!",nRet);
return (1);
}
return (0);
```

See Also

OpenPort

5.2.25 setdebug

Syntax

void setdebug(int bEnable)

Parameters

bEnable ： Enable or disable the show message of the debug window.

Description

This function will enable or disable the show message of the debug window. If set to TRUE, the debug window is enabled. If set to FALSE, the debug window is disabled.

Example

```
// The example will show the pChar and f1 in the debug window
char pChar[20];  // declare pChar as char[20]
double f1; // declare f1 as double

f1=2.789;
pChar=”The value of f1 is “;
setdebug(TRUE);
debug(pChar,f1); // It will show “The value of f1 is 2.789”
setdebug(FALSE);
debug(pChar,f1); // It does not show anything
```

See Also

debug, printf

5.2.26 SetupTime

Syntax

void SetupTime()

Parameters

None

Description

This function will display the clock on the screen and provide user interface to setup time and clock.

Example

```
SetupTime();
```

See Also

HelpWindow, LoadBMP, LoadBMPByValue

5.2.27 ShowText

Syntax

void ShowText(int nType, int nX, int nY, int nStyle, string sText)

Parameters

nType: Specifies the type of text. The value 1 is for the notification text.

nX: Specifies the beginning x-coordinate of text.

nY: Specifies the beginning y-coordinate of text.

nStyle: Specifies the style of display text. Please set to 0.

sText: pecifies the display string of text.

Description

This function set the help window on the specified coordinate when entering configuration mode.

Example

```
string sMsg;
sMsg=”Please check the input pin is OK or not”;
ShowText(1,0,0,0,sMsg);
```

See Also

HelpWindow, LoadBMP, LoadBMPByValue

5.2.28 Sleep

Syntax

void Sleep(int nMilliseconds)

Parameters

nMilliseconds :Specifies the time, in milliseconds, for which to suspend execution. A value of zero causes the thread to relinquish the remainder of its time slice to any other thread of equal priority that is ready to run. If there are no other threads of equal priority ready to run, the function returns immediately, and the thread continues execution.

Description

This function suspends the execution of the current thread for a specified interval.

Example

```
int nI;

setdebug(TRUE);
for (nI=0;nI<500;nI=nI+1) {
Sleep(1000);
debug(“Sleep “,nI,” mini-second”);
}
```

See Also

5.2.29 sprintf

Syntax

void sprintf(char* pBuf, char* pFormat,... or

void sprintf(char* pBuf, string sFormat,...)

Parameters

pBuf: Specifies the destination array.

pFormat : Specifies the Null-end character array format

sFormat: Specifies the string format

...: Specifies more variant variables to be added into the format string

Description

This function returns the number of bytes stored in buffer, not counting the terminating null character.

Comments

The sprintf function formats and stores a series of characters and values in buffer. Each argument (if any) is converted and output according to the corresponding format specification in format. The format consists of ordinary characters and has the same form and function as the format argument for printf. A null character is appended after the last character written. If copying occurs between strings that overlap, the behavior is undefined.

Example

```
// The example will show the pChar and f1 in the debug window
char pChar[20];  // declare pChar as char[20]
double f1; // declare f1 as double
int nI;
string  aStr;
char pData[15];
char pBuf[50];

f1=2.789;
nI=120;
pData=”Test Printf”;
aStr=”Test printf 2”;
setdebug(TRUE);
sprintf(pBuf,”The f1=%f, nI=%d, pBuf=%s, aStr=%s”,f1,nI,pBuf,aStr);
```

See Also

printf, debug , setdebug

5.2.30 strcpy

Syntax

int strcpy(char* pDest, char* pSour, int nStart, int nLen) or

int strcpy(char* pDest, string sSour, int nStart, int nLen)

Parameters

pDest: Specifies the Destination array

pSour: Specifies the NULL ended character array.

sSour: Specified the source string

nStart: the begin position to be copied, is the 0-based index

nLen: the length from source string to destination array

Description

This function return the actually length from source string to destination array.

Example

```
// The example will show the pChar and f1 in the debug window
char pDest[20];  // declare pDest as char[20]
char pSour[20];  // declare pSour as char[20]
string aStr;      // declare aStr as string

aStr="0123456789";
pSour="abcdefghijk";
setdebug(TRUE);
strcpy(pDest,pSour,5,4);
debug("strcpy(pDest,",pSour,",5,4)=",pDest);  // It will show "strcpy(pDest,abc-defghijk,5,4)=fghi
strcpy(pDest,aStr,3,5);
debug("strcpy(pDest,",pSour,",3,5)=",pDest);  // It will show "(0123456789,3,5)=34567
```

See Also

strlen

5.2.31 strlen

Syntax

int strlen(char* pChar) or

int strlen(string aStr)

Parameters

pChar: Specifies a Null-ended character array

aStr: Specifies a string variable

Description

This function return the character number of NULL ended character array or a string variable.

Example

```
// The example will show the pChar and f1 in the debug window
char pChar[20];  // declare pChar as char[20]
string aStr;      // declare aStr as string

aStr="String";
pChar="pChar";
setdebug(TRUE);
debug("strlen(",aStr,")=",strlen(aStr));  // It will show "strlen(String)=6"
debug("strlen(",pChar,")=",strlen(pChar));  // It will show "strlen(pChar)=5"
```

See Also

strcpy

5.2.32 time

Syntax

string time()

Parameters

none

Description

This function returns the current date, its format is fixed as "hh:mm:ss".

Example

```
string sTime;
setdebug(TRUE);
sDate=time();
debug("The current time is",sTime);
```

See Also

date

5.2.33 write

Syntax

int write(int hFile,char *buf, int nCount)

Parameters

hFile: Specifies the handle referring to open file

buf: Specifies the storage location for data

nCount: Specifies the maximum number of bytes

Description

This function returns the number of bytes actually written. If the actual space remaining on the disk is less than the size of the buffer the function is trying to write to the disk, write fails and does not flush any of the buffer's contents to the disk. A return value of -1 indicates an error.

Example

```
int hFile;
int nLen;
char data[512];
hFile=open("\\flash storage\\template.txt", FILE_CREATE_RW);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");
}
else {
// process the file
seek(hFile,SEEK_END,0);
data="This is a test file";
nLen=write(hFile, data, strlen(datta));
 close(hFile); // close the file
}
```

See Also

open, close, read, seek, readln, writeln, filecopy, eof

5.2.34 writeln

Syntax

int writeln(int hFile,char *buf)

Parameters

hFile: Specifies the handle referring to open file

buf: Specifies the storage location for data

Description

This function returns the number of bytes writen. This function writes the buf data plus an end-of-line marker (carriage-return/linefeed) to the file. If the handle is invalid, or the file is not open for writing, or the file is locked, the function returns -1.

Example

```
int hFile;
int nLen;
char data[512];
hFile=open("\\flash storage\\template.txt", FILE_READ);
setdebug(TRUE);
if (hFile==0) {
debug("Open file failed!");

}
else {
// process the file
data = "This is a test";
writeln(hFile, data);
close(hFile); // close the file
 }
```

See Also

open, close, read, write, seek, readln, filecopy, eof

Trusted ePlatform Services

ADVANTECH

www.advantech.com