C-Toolkit User's Manual

The Visual Logic Controller®

RESTRICTED RIGHTS LEGEND

TRADEMARKS

Document version 1.1

Document written and produced by Mahler Associates, Inc., Ann Arbor, MI.

WARRANTY/REMEDY

Steeplechase Software warrants goods of its manufacture as being free of defective materials and faulty workmanship. Commencing with the date of shipment, Steeplechase's warranty runs for 90 days. If warranted goods are returned to Steeplechase during that period of coverage, Steeplechase will replace without charge those items it finds defective. The foregoing is Buyer's sole remedy and is IN LIEU OF ALL OTHER WARRRANTIES, EXPRESS OR IMPLIED, INCLUDING THOSE OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The Buyer acknowledges that using programs improperly developed with the C-Toolkit can lead to significant failure risks in a manufacturing environment. Specifically, the Buyer acknowledges his responsibility to ensure that programs developed with the C-Toolkit utilize good design practice, are bug free, integrate properly with VLC developed programs, and are fully tested and operate properly in the ultimate customer's manufacturing environment.

Contents

1. BEFORE YOU BEGIN

This *VLC C-Toolkit User's Manual* provides information for programmers who want to take advantage of powerful customization and programming features available using the Visual Logic Controller (VLC) C-Toolkit. Before using this manual, you should be familiar with control project creation in the VLC Control Designer. Be sure to read the *Control Designer User's Manual* — especially Chapter 8, *Special Functions* and Chapter 16, *Theory of Operation*.

NOTE: *You can also find these topics in the VLC Control Designer online help system.*

To use C programs within a VLC control project, you must create driver functions that project developers access through the Control Designer special function tool. You can use the C-Toolkit to:

- Implement existing C code.
- Create and protect proprietary code that you don't want users or competitors to see.
- Implement special algorithms that would be difficult or impossible with current VLC flow chart or RLL tools and functions.

Driver functions you develop with the C-Toolkit are similar to the Utility driver provided by Steeplechase. Control program developers can access your special drivers and their functions in both flow chart and RLL programs.

WARNING: *Before attempting to use VLC C-Toolkit, you should be familiar with C programming, and* ***must understand programming in the Windows NT environment****. Because the C-Toolkit lets you perform functions directly on the hard realtime system, you can easily crash both Windows NT and the realtime system. It is also possible to perform illegal operations on the UIOT that will cause the system to crash.* ***It is your responsibility to exercise caution and thoroughly test your applications with your C-Toolkit-developed functions.***

You must use Microsoft Developer Studio (MS DevStudio) to create your driver functions with C-Toolkit because the library functions we supply are tailored to this development environment. Your compiled module is linked with the INtime realtime system and can access all INtime system API functions. For information on INtime system API functions, you can access the INtime online help from the **Start** menu. Select **Start/Programs/Visual Logic Controller/Development/INtime**.

1.1 About This Manual

This manual defines the VLC C-Toolkit for the Visual Logic Controller running under Windows NT. You can purchase the C-Toolkit with or without the optional VLC Realtime Debugger.

NOTE: *We recommend that you purchase the VLC Realtime Debugger for all but the simplest implementations, or for implementing C code that you have already debugged and used in other environments.*

This manual assumes that you are using Microsoft Visual C++ and MS DevStudio environment. It provides information on how to effectively use the C-Toolkit with or without the debugger. However, we make no attempt to provide instruction on the use of the VLC Realtime Debugger. See the online documentation that comes with this program for information on how to use the debugger.

NOTE: *For information on the VLC Realtime Debugger, see the help file:* **`\INtime\Sswin32\Sswin32.hlp`**

In this manual you will see how to use the VLC C-Toolkit. You'll find:

- Installation instructions.
- C code development instructions and guidelines.
- User interface requirements.

This introduction continues with a description of conventions used in this manual, and how to contact Steeplechase Customer Support. The remaining chapters are:

Chapter 2, *Installing VLC C-Toolkit*, provides step-by-step instructions for installing VLC C-Toolkit.

Chapter 3, Creating User-Defined Driver Functions, includes a description of programming requirements for the VLC C-Toolkit, and how to create VLC driver functions with C-Toolkit.

Chapter 4, *Function Reference*, provides reference information for all the functions required to implement drivers you develop using C-Toolkit.

A sample application that uses a C Tookit-developed driver appears in the **`c:\Program Files\VLC\Development\Ctoolkit\Samples`** directory. This directory includes a sample VLC project and source code to illustrate use of a driver function created with C-Toolkit.

1.2 Manual Conventions

This manual uses Windows NT style terminology and long filenames.

1.2.1 Text Conventions

This manual uses a few special symbols and conventions. Words and characters shown in **`courier`** font are directory or file names. For example, **`\VLC\Project`** is a directory name; **`myprog.rt3`** is a filename. Specific programming language statements and examples also appear in courier font in mixed case. Sample code or filenames that include a name you must type appears in angle brackets with a descriptive name in italics, for example **`<`*`module_name`*`>.rt3`**. This name indicates that you should provide the name of your module (without the brackets) followed by a period and the **`rt3`** extension.

Words and characters in **bold** in a san-serif font indicate anything you must type exactly as it appears. For example, **COM1** specifies that you should type the bold characters exactly as shown. Bold is also used to identify command buttons or key words and phrases found in dialog

boxes. For example, "click the **Settings...** button." In some cases, words and characters in bold also indicate **emphasis**.

In a few places, you'll see mnemonic command letters used in menus and dialog boxes (e.g., **F** for **File** menu). These appear in bold with an underline to be consistent with the screen displays.

Words and characters in *italics* indicate a new term. An explanation generally follows the italicized term. Italics also indicate references to specific chapter or section titles.

In Chapter 3, *Creating User-Defined Driver Functions*, names of VLC functions or variables appear in bold in a san-serif font. For example, **rtOpen()** is the name of a function. Chapter 4, *Function Reference*, uses this same conventions, plus arguments appear in italic. For example, *TagName* is the name of an argument.

1.3 Steeplechase Software Customer Support

If you have a question about the C-Toolkit and can't find the answer in this manual or in the Help system, contact our customer support using phone, fax, or e-mail. Our Customer Support staff will give you the advice you need to get the most from the Visual Logic Controller.

We suggest that you try to duplicate the problem, and as you do so, write down each step as well as any error messages you see. So that we can provide you with the best possible customer support, we recommend that you be at your computer when you call, and have the following available:

- *Visual Logic Controller C-Toolkit User's Manual* (this manual).
- The version on your VLC C-Toolkit disk label.
- Windows NT version number, you can display this by double-clicking the System application in the Control Panel.
- Computer information including type and model of computer, video card, and type of I/O boards installed in the system. Amount and type of installed memory are also important.

- Any error messages you received.

Steeplechase Software, Inc.
1330 Eisenhower Place
Ann Arbor, MI 48106-3049

Phone:	(734) 975-8100 (main office)
	(734) 975-8120 (Customer Service)
Fax:	(734) 975-8123
e-mail:	**help@steeplechase.com**
	http://www.steeplechase.com

2. INSTALLING VLC C-TOOLKIT

This chapter provides installation procedures for VLC C-Toolkit for Windows NT.

2.1 What You Need

Before installing VLC C-Toolkit, we recommend that you have the following:

- A personal computer with an Intel microprocessor. Windows NT systems require a Pentium processor (or higher) with at least 32 MB of RAM and Service Pack 3 installed.
- A 3.5-inch floppy disk drive.
- A hard disk for installing all components and sample programs with:
  - 8.5 MB for the C-Toolkit.
  - 2.5 MB for the VLC Realtime Debugger.

2.2 Licenses

You can use VLC C-Toolkit without special hardware or software licenses. However, to execute programs that use driver functions you create with the C-Toolkit, you must execute on a system with a valid VLC Runtime license.

2.3 Installing VLC C-Toolkit

Installing VLC C-Toolkit is easy because the install program takes care of most details for you. Don't try to copy the files from the distribution disk(s).

To install VLC C-Toolkit follow these steps:

[1] Start Windows NT. If necessary, login as an administrator.

[2] Insert the disk labeled **VLC C-Toolkit, Disk 1** into a floppy drive (usually drive A), or insert the CD-ROM in your CD-ROM drive.

[3] From the **Start** menu, select **Run...** to display the **Run** dialog.

[4] If you're installing VLC C-Toolkit from drive A, type: **`A:\setup.exe`**, and press e .

Figure 2-1. The **Run** dialog box with the VLC C-Toolkit setup command.

NOTE: *You should install the standard C-Toolkit files before installing the debugger.*

[5] When the installation program begins to execute, it displays an installation wizard. The first page of the wizard has a message similar to the one shown in Figure 2-2. After reading the message, click the **Start** button to continue.

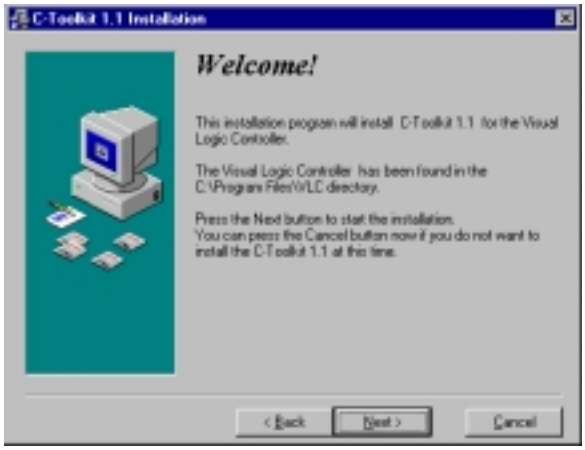

Figure 2-2. Initial install message.

[6] The next page, shown in Figure 2-3, lets you choose a different drive and directory for installing the C-Toolkit. Use the Browse button select another drive or directory. The installation program will create this directory if necessary. Make sure that the **Free Disk Space after Install** is a positive number before continuing. If it is not, select another drive or free-up space on the selected drive. Click the **Next** button to continue.

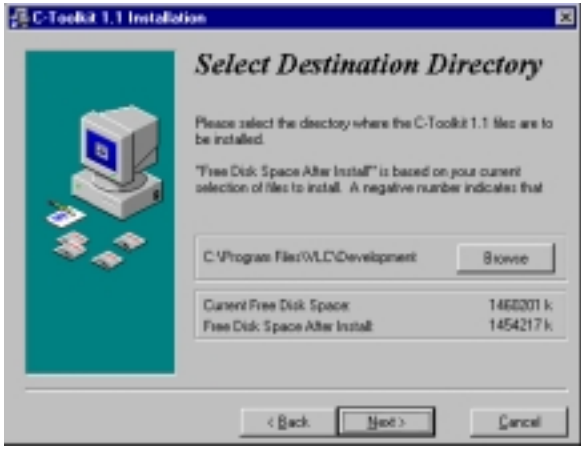

Figure 2-3. The installation **Select Destination Drive** page lets you pick the destination for the INtime development system installation.

[7] The next page, shown in Figure 2-4, indicates that the setup program has all the information it needs and is ready to begin copying programs to your hard disk. Click the **Next** button to continue.

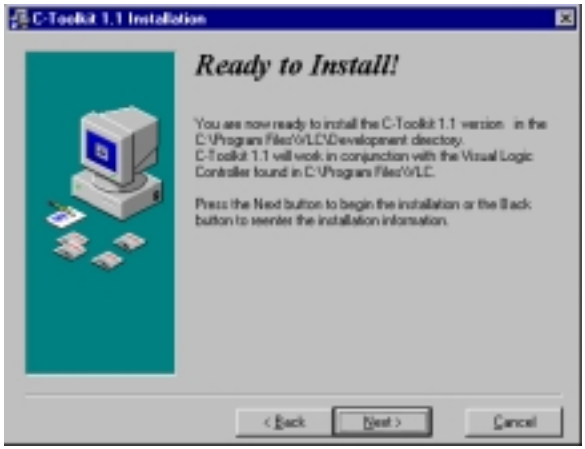

Figure 2-4.
Select **Next** to let the installation program begin copying files to your system.

[8] The installation program begins copying files to your hard disk. It displays a dialog that shows the progress of the installation for each file (see Figure 2-5).

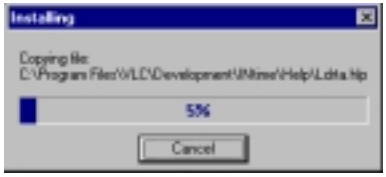

Figure 2-5.
This dialog displays a progress bar for each file installed.

[9] When the installation program completes copying all required files, it displays the wizard page shown in Figure 2-6. Click the **Finish** button to complete the installation.

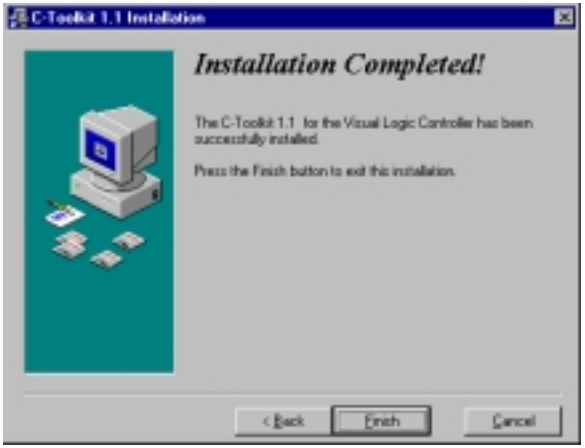

Figure 2-6. This dialog appears at the end of the installation process.

[10] If you purchased the optional VLC Realtime Debugger, insert the disk labeled **VLC Realtime Debugger, Disk 1** into a floppy drive (usually drive A), and repeats steps [3] through [9]. Although, this time Step [6] will only let you modify the disk drive and require installation in the **`\INtime\Sswin32`** directory.

2.4 Installed Files

The VLC C-Toolkit installs the files shown in Table 2--7 in sub-folders under the install path. By default, the install path is: **`c:\Program Files\VLC\Development`**.

Table 2--7. VLC C-Toolkit Installed Files

Component	Default Location
INtime System include files	**`\INtime\Inc\*.h`**
INtime System Library files	**`\INtime\Lib\*.lib`**
INtime documentation	**`\INtime\Help`**
VLC-specific include files	**`\Inc\*.h`**
VLC-specific library files	**`\Lib\*.lib`**
VLC sample C project	**`\Ctoolkit\Samples`**
VLC Realtime Debugger	Installs additional files in **`\INtime\Bin`** and **`\INtime\Sswin32`**

3. CREATING USER-DEFINED DRIVER FUNCTIONS

This chapter provides an overview of important programming considerations for creating user-defined driver functions with the VLC C-Toolkit. We assume that you have an understanding of programming concepts for Windows NT, and that you've created other programs using the C language and the MS DevStudio.

Although you can create driver-based functions for VLC version 3.63 (after loading the VLC C-Toolkit), this chapter assumes version 4.1 (or greater) when discussing the Control Designer and how the **Edit Special Function** dialog appears.

3.1 Driver Development Concepts

You must use MS DevStudio and Visual C++ to create driver-based functions for use within a VLC Control Designer project. In DevStudio, you create two projects. Together, these two projects comprise a module that produces your driver. (In this manual, we use the term driver and module interchangeably.) The DevStudio projects are:

GUI	Defines the parameter page of the **Edit Special Function** dialog or the configuration dialog used for RLL special functions. The techniques defined for creating this project provide customizing special function configuration without requiring additional programming.
Runtime	Defines the actual C code and other project data required for the driver you are developing.

To facilitate your development efforts, the C-Toolkit includes a directory with sample MS DevStudio projects that define a module. This sample module produces a driver with three functions. The sample folders also include a Control Designer project that uses the C-Toolkit driver functions. When you begin developing a custom

driver, you should copy the sample module — follow the steps in *Creating a New Driver* on page 58 — and use it as a prototype for your module development.

When compiled and linked, these two MS DevStudio projects generate two files:

`<module_name>.io3` Used by the VLC Control Designer to display the configuration page of the **Edit Special Function** dialog or the configuration dialog used for RLL special functions.

`<module_name>.rt3` Provides the INtime executable code. This code:

- Executes at a higher priority than the NT operating system.
- Can directly access any hardware resource.
- Its entry points are deterministically called at very precise moments in time.
- It can directly call any INtime system API, and can dynamically create new threads. The driver is responsible for deleting any threads it creates.
- It can directly access the VLC UIOT for the running project.

WARNING: *You are responsible for preserving the integrity of all data. Corrupting the UIOT or accessing null pointers will lead to a system hang.*

WARNING: *The VLC is a deterministic realtime system. To keep cycle times to a minimum, functions you create for access by flow chart or RLL programs must execute as fast as possible. If your function requires significant processing or waiting time, you should execute these time-intensive operations in a lower priority thread.*

3.1.1 Control Designer Interface

Driver-based functions you create must interface with the VLC Runtime using event-driven routines. The VLC Runtime calls specific entry points for each loaded driver (or module) at specific points in the life of a project. The Runtime calls some entry points only once on project initiation or shutdown, and it calls other entry points as required during control program execution.

A module may appear many times in a VLC control project. Each instance has its own data area (see *DRIVER_INST Structure* on page 24), and the VLC Runtime passes a pointer to the appropriate data structure as a parameter to the entry point that is instance-specific.

Many of the functions called by the VLC Runtime identify when a control program changes state, so your functions can execute appropriate operations for the new state. Your module must define all Runtime event entry points, however you can simply return a **SUCCESS** value without performing any other operations.

The VLC Runtime has three principal states, and events identify transitions between these states:

- Unloaded
- Loaded/stopped
- Running

The following list briefly defines all the event functions your module must have available, and Figure 3-1 shows a graphical representation of the state transitions and the functions called at each transition.

rtIdentify() — Called only once when the module is launched. This entry point provides identification information for the module including the module ID, name, and version. The ID value returned must be unique across all modules. The version of the **<*module_name*>.io3** file must match the version of **<*module_name*>.rt3**, as reported to the Runtime engine by the **rtIdentify()** function.

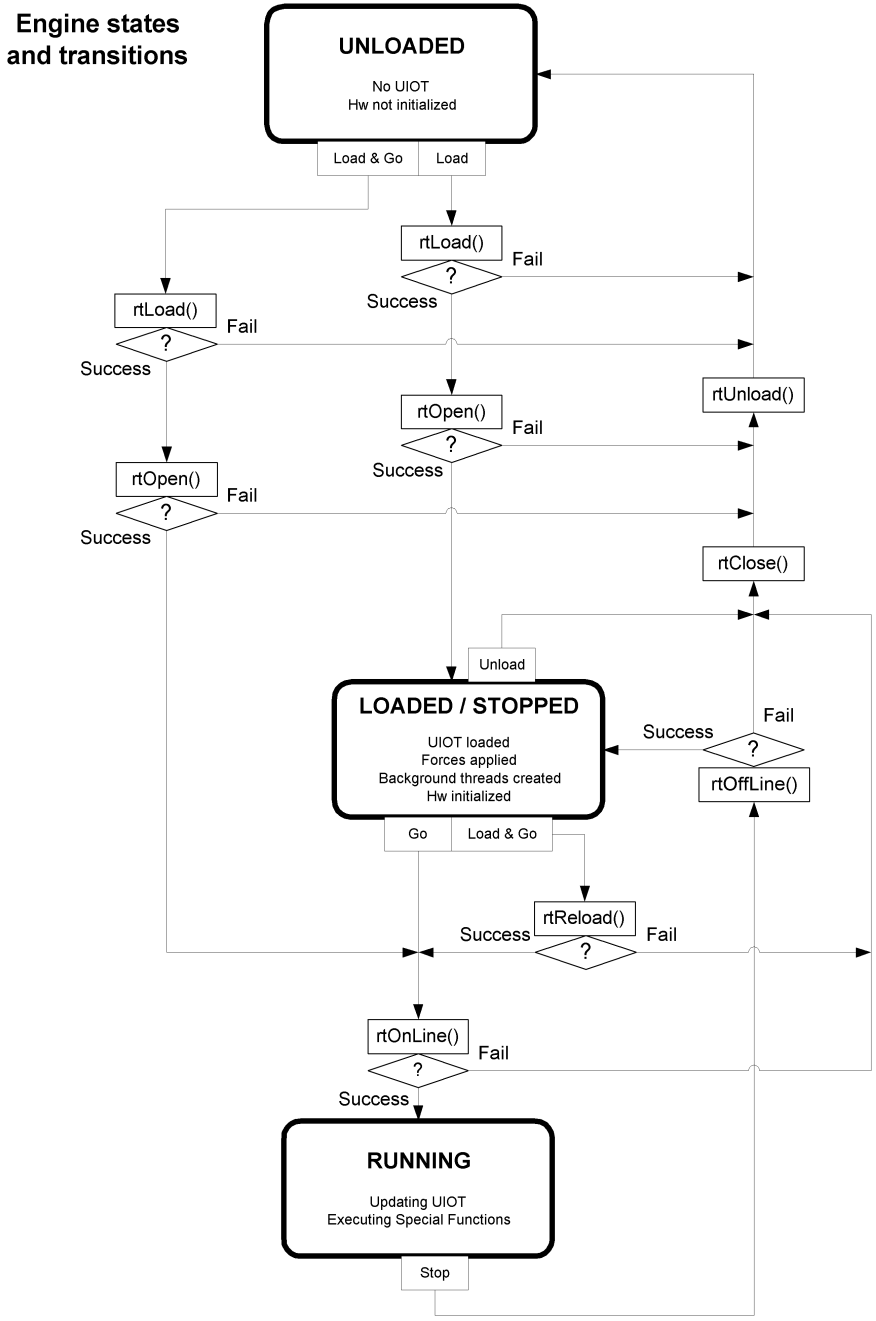

Figure 3-1. VLC Runtime engine state-transition flow chart.

rtLoad()	Called once for every module type. This function provides the module with the specified VLC scan rate in microseconds.
rtOpen()	Called once for every instance of a module. Use this function to initialize your module.
rtReload()	Called once for every instance of a module if the Control Designer user requests a Load and Go from the Loaded/Stopped state.
rtOnLine()	Called once for every instance of a module when the Control Designer user initiates execution of the Runtime project.
rtOffLine()	Called once for every instance of a module when stopping.
rtClose()	Called once for every instance of a module after the VLC Runtime engine stops a project This function executes regardless of whether the stop was due to operator command or an error.
rtUnload()	Called once for every module type to unload a project.
rtInput()	Called in the Running state at the beginning of every scan cycle.
rtOutput()	Called in the Running state at the end of every scan cycle.
rtSpecial()	Called in the Running state whenever the execution flow reaches a special function element.

For details on the parameters required for each of these functions, see Chapter 4, *Function Reference*.

3.1.2 Defining Module Parameters

Developers of VLC control projects place special function elements in either flow chart (see Figure 3-2) or RLL programs. Each instance of a special function defines the driver (your module), function desired, and parameters for the function. The VLC Runtime calls your module **rtSpecial()** entry point as a void function whenever it encounters the special function element in the program flow. Because it's called as a void function, your module must pass returned values via parameters.

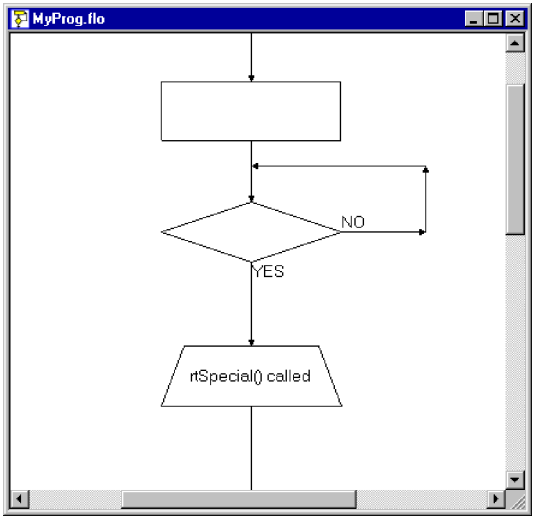

Figure 3-2. Flow chart program segment showing special function element. Each instance of one of these elements causes the VLC Runtime to call **rtSpecial()**.

You use the GUI MS DevStudio project to define the configuration page of the **Edit Special Function** dialog or the configuration dialog used for RLL special functions. In either case, you define the static text, edit fields, and drop-downs in the **`<module_name>.rcd`** file. This file looks like a standard C language resource file.

Because the VLC Control Designer opens, closes, and reads the data on the configuration page or dialog, the GUI portion of your module doesn't need any C language code. To see an example of a **`<module_name>.rcd`** file, open the **`\Program Files\VLC\Development\Ctoolkit\Gui\Ctoolkit.rcd`**

file. For a description of the sections of this file, see *RCD File Definition* on page 30.

3.1.3 Passing Parameters

When a control program developer selects and configures a driver-based function, the configuration dialog specifies constant values or tag names that are input to the function. Functions in your module obtain these parameters passed by value. You can safely use or modify your local copy of these parameters without concern for how the tags are used in the control application. A sample code segment that defines a function with an input paramter is:

```
void foo( [in] int X)
{
        //use X
}
```

Since all driver-based functions are void type, you must pass return values in parameters provided by the calling control application. In this case, your module function gets a pointer to the return tag in a call by reference. A sample code segment that defines a function with an output parameter is:

```
void foo ( [out] int* pY)
{
        ...
        *pY = 1;   //return value of 1 to control program
}
```

Large data types, like arrays, are called by reference, regardless of whether they are **[in]**, **[out]**, or **[in, out]**. As in return tags, your module function gets a pointer to the tag in a call by reference. A sample code segment that defines a function with an input/output array parameter is:

```
void foo ( [in, out] int* pBuffer)
{
        ...
        *pBuffer = *(pBuffer+1) //copy 2nd element to 1st
}
```

As part of the definition of a module, you create a variable structure called **SPECIAL_INST** (for details, see *Driver.h File Definition* on

page 22). This structure includes runtime data private to an instance of a special function element. Each special function element that appears in a flow chart or RLL program has its own instance in the UIOT.

The Runtime engine copies parameters passed by value to the **SPECIAL_INST** structure. For parameters passed by reference, the Runtime engine stores a UIOT offset in the **SPECIAL_INST** structure. The actual parameter is stored in a different area of the UIOT that is independent of the instance's **SPECIAL_INST** structure.

Figure 3-3 illustrates two tags (called, **tag** and **array**) in the UIOT that are referenced by a special function parameter block. Later sections of this manual will provide additional information that will help you understand this diagram.

UIOT references

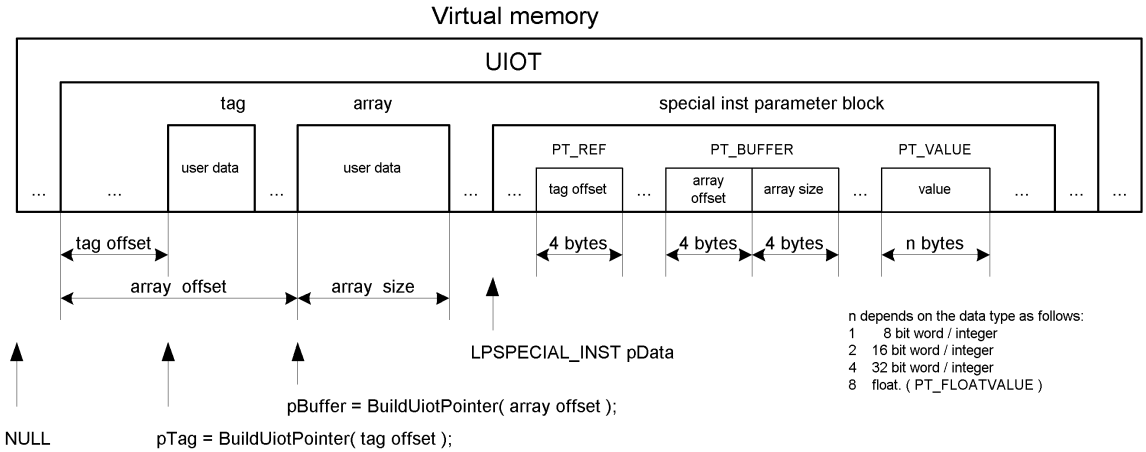

Figure 3-3. UIOT layout.

3.1.4 Module Return Codes

The state transition notification functions (**rtIdentify()**, **rtLoad()**, **rtOpen()**. **rtOnLine()**, **rtOffLine()**, **rtClose()**, and **rtUnload()** must return zero if they succed. Any non-zero return code indicates a failure, and the VLC Runtime engine performs the following operations:

[1] Switches to the Unloaded state following the path shown in Figure 3-1 on page 16.

[2] Calls **rtOffLine()** in running module instances, and **rtClose()** and **rtUnload()** in loaded module instances. The VLC Runtime ignores any new errors.

[3] Displays an error message dialog containing:

- Module identifier, as defined in **`driver.h`**. In the sample prototype, this is **DriverCTOOLKIT** (see *Module Identification* on page 23).
- Module type. This is the module short name from the **NETWORK_TYPE** paragraph in the **`<module_name>.rcd`** file (see *NETWORK_TYPE* on page 34).
- Instance name as defined in the VLC project.
- Error code that generated the event.
- Module-specific error string, if the VLC Runtime can find a string identifier that matches the error code (see *Error Strings* on page 49).

The running state functions (**rtInput()**, **rtOutput()**, and **rtSpecial()**) must always return zero. If these functions detect an error, they must report the error to the running project through parameters of the special function.

3.1.5 Direct Calls

The direct call mechanism lets programmers reduce execution time of their drivers during each scan cycle. Using direct calls eliminates several system calls that significantly impact driver execution time. However, this mechanism comes with a cost. This section explains the

issues, provides a warning, and gives programmers the knowledge to decide whether and when to implement direct calls.

Running state functions (**rtInput()**, **rtOutput()**, and **rtSpecial()**) must execute as quickly as possible with the only restriction that they execute to completion within a scan cycle. One component of driver module execution is the system overhead required to call the driver.

The VLC Runtime switches context to call driver module entry points. The amount of time required for context switching depends on the speed of the CPU. For 200 MHz processors, it is in the range of 10µs. The direct call mechanism available for running state functions can reduce this system overhead by 50%. The VLC Runtime engine saves time with direct calls by calling the driver from within the Runtime engine context. This saves two task switches for every runtime function call.

You can specify that the VLC Runtime engine directly call **rtInput()**, **rtOutput()**, and/or **rtSpecial()**). To request this feature, **rtLoad()** must set the appropriate direct call control bit(s) in the rDirectCalls parameter (see *rtLoad* on page 69).

WARNING: *Because directly called functions execute in the VLC Runtime engine context, you cannot call C runtime API functions. Functions like* **malloc()**, **free()**, *and* **sprintf()** *will lock the entire system.*

3.2 Important File Definitions

This section describes files that you will copy and modify whenever you create driver-based functions (your module).

3.2.1 Driver.h File Definition

The **driver.h** file is the master include file for any module that you create. You must customize this file for your module. You can find the prototype in the **\Program Files\VLC\Development\Ctoolkit\Inc** folder. The DevStudio C-Toolkit prototype projects include this file in both the GUI and Runtime projects.

There are three important sections in the **driver.h** file. They are:

Module identification	This is the unique ID defined for the module (driver). At runtime, the module reports this identifier to the Runtime engine through the **rtIdentify()** function.
DRIVER_INST	This structure contains runtime data local to the module. The VLC Runtime allocates one copy of this structure in the UIOT for every instance of the module. As part of the calling sequence, the Runtime provides most of the module entry points (see Chapter 4, *Function Reference*) with a pointer to the structure in the UIOT assigned to the current instance of the module.
SPECIAL_INST	This structure contains runtime data local to special function elements. The VLC compiler allocates an area in the UIOT for every instance of a special function element in flow chart and RLL programs. The VLC Runtime provides the **rtSpecial()** entry point with a pointer to the structure in the UIOT assigned to the current instance of the special function element.

Module Identification

Each module must have a unique identifier. As part of the procedure to create a new module (see *Creating a New Driver* on page 58), you define a unique identifier and edit the **driver.h** file to include the new ID. The VLC Runtime obtains the module ID when it calls your **rtIdentify()** function. You also use this ID number in the release command (see *Using a New Driver* on page 62).

In addition to a module ID, you specify a version number in the **driver.h** file. When the version number changes, the Control Designer notifies the control programmer to recompile a running project with a different version.

Figure 3-4 shows the module identification segment at the beginning of the prototype **driver.h** file. In this section, you specify the module ID number and version number as defined macros.

NOTE: *You must rename these macros as part of the module creation process (see* Creating a New Driver *on page 58).*

The macro definitions are:

DriverCTOOLKIT — Choose a new name consistent with your module name. Specify the value as a long hexidecimal constant. In Figure 3-4, the value is: **0x5e7f8d43L**.

CTOOLKITVERS — Choose a new name consistent with your module name. Specify the value as a long hexadecimal constant. The first word is a major version number and the second word is a minor version number. In Figure 3-4, the value is: **0x00010001L**.

```
// "CTOOLKIT" id num. Make sure this id is unique across all VLC C-modules and
//                   drivers.

#define DriverCTOOLKIT          0x5e7f8d43L


/*
    Version 01.0001  (last chg 11/10/98)
    Bump this version # every time DRIVER_INST, DEVICE_INST or DEVICE_IO
    structs have changed.
    This will force old projects to be recompiled before execution.
*/
#define CTOOLKITVERS            0x00010001L
```

Figure 3-4. Module Identifier section of the **driver.h** file.

DRIVER_INST Structure

The VLC Control Designer lets application developers use a driver any number of times in a control program. In addition, the control pro-

grammer can give the same driver different names. In some cases, this may be required, for example, when there are multiple physical devices that use the same driver (see Chapter 4, *Drivers, Devices, and Tags* in the Visual Logic Controller *Control Designer User's Manual*. Each name a control programmer gives a driver defines a unique instance for the driver.

The **DRIVER_INST** structure contains runtime data local to a particular instance of a driver (module). The VLC Runtime allocates this structure in the UIOT for every instance of the module. As part of the calling sequence, the Runtime provides most of the module entry points (see Chapter 4, *Function Reference*) with a pointer to the current structure in the UIOT.

Typical variables that you might include in this structure have the following uses:

- Communication with background threads.
- Communication among function calls, call history, and other data that needs to be exchanged between instances of special function elements.
- Data values used by state machines within your module.
- Any data that you want to store for each instance of the driver.

Figure 3-5 shows a section of the **DRIVER_INST** structure in the prototype **`driver.h`** file. This structure must have **NETWORK Net** defined as its first member. Other than that, this structure can have any members of any type required to implement your driver. When the VLC Runtime calls **rtOpen()** it also sets all members of this structure to zero.

SPECIAL_INST Structure

The **SPECIAL_INST** structure contains runtime data local to special function elements. The VLC compiler allocates an area in the UIOT for every instance of a special function element in flow chart and RLL programs. The VLC Runtime provides the **rtSpecial()** entry point with a pointer to the current structure in the UIOT.

```
#pragma BYTE_ALIGN(_DRIVER_INST)
typedef struct _DRIVER_INST
{
    NETWORK Net;

        /* Run-time Dynamic Variables */
    UINT32      bFirstCycle;

    LINKED_LIST Pend;           // Pointer to linked list of pending functions
    LINKED_LIST Done;          // Pointer to the linked list of done  functions

    TASK        BackgroundTask; // controls for the background task
    TASK        InterruptTask;  // controls for the interrupt task
    UINT32      Sentenial;      // 0x55667788 - display this value using
                                // Debugger to check correct map
} DRIVER_INST, *LPDRIVER_INST;
#pragma BYTE_NORMAL()
```

Figure 3-5. **DRIVER_INST** section of the **driver.h** file.

You must actually define the data structure for special functions twice in the **driver.h** file — once in the **SPECIAL_INST** structure, and once as a series of **#define** statements that specify offsets and sizes (these are referred to as **FNC_...** statements, and appear at the end of the **driver.h** file).

WARNING: *These two data structure definitions must be identical.*

Typical variables that you might include in this structure have the following uses:

- **[in]** parameters passed by value. The size of these parameters must fit the size of the member of the structure used to store the value. Parameters passed by value include:
  - A **FunctionID** member to implement multiple functions with a single entry point.
  - Constants or tag values defined as parameter values to pass to the special function element.

- **[out]** or **[in, out]** parameters passed by reference. This structure includes references to these parameters as offsets from the beginning of the UIOT. The actual parameters reside elsewhere in the UIOT. For parameters passed by reference, the size of these offsets in the **SPECIAL_INST** structure member is four bytes. To access actual data values, your code must call **BuildUiotPointer(UIOTOffset)** to get an actual pointer, where **UIOTOffset** is a member of the **SPECIAL_INST** structure with the call by reference pointer.

- Any local variables required for execution of the module. Typical variables include (see prototype example in Figure 3-6):

  - **pNext**, which, in the sample prototype, is required to enqueue the function in lists.

  - State machine flags, such as **Status** and **Busy**.

  - **MarkTime**, which, in the sample prototype, keeps track of when the function has started and when it times out.

```
typedef struct _SPECIAL_INST
{       // Compile-time Static variables.  This structure maps
        // to .rcd description
                                        // off,  sz,
    SPECIAL_INST_PARAM  User;           //   0   40
    SPECIAL_INST_PARAM  Work;           //  40   40

        // generic, same for all drivers having asynchronous special functions
    UINT32                MarkTime;     //  80    4  when this s.f. must be
                                       // complete
    SINT16                Status;       //  84    2
    UINT16                Busy;         //  86    2
    struct _SPECIAL_INST* pNext;        //  88    4

} SPECIAL_INST, *LPSPECIAL_INST;       //  92 == sizeof( SPECIAL_INST )
```

Figure 3-6. **SPECIAL_INST** section of the **driver.h** file.

Figure 3-6 illustrates that the **SPECIAL_INST** structure can, itself, contain other structures. In this example, the **SPECIAL_INST** structure includes two instances of the **SPECIAL_INST_PARAM** structure — one called **User** and the other called **Work**. It is important to note that the

SPECIAL_INST structure defines the local module data structure for the runtime project code.

NOTE: The VLC Compiler uses the definition of the structure found in the **<*module_name*>.rcd** file as **FNC_...** definitions. The definition of the **SPECIAL_INST** structure must be equivalent to the **FNC_...** definitions (see the end of the **driver.h** file in the prototype file).

In addition, **FNC_SPECIAL_INST_SIZE** must equal the structure size (for more information on **FNC_SPECIAL_INST_SIZE**, see NET_FUNC_TYPE *on page 36*).

The section of the prototype file in Figure 3-6 shows two **SPECIAL_INST_PARAM** structures to reinforce an important driver programming technique. Because asynchronous functions execute concurrently with a flowchart or RLL program, it is safer to provide a copy of the parameter block for any background threads you create. To accomplish this, you specify the **User** structure as the parameter block used when the VLC Runtime calls your function. You then copy the contents of the **User** structure to **Work**, and pass the **Work** structure along to threads you create.

Looking at the definition of the **SPECIAL_INST_PARAM** structure, shown in Figure 3-7, you can see that it is composed of three structures. In this simple example, all these structures are unioned to zero offset. If you look at the **driver.h** file, you will see that **SPECIAL_INST_COMMAND** and **SPECIAL_INST_PORT** both have **SPECIAL_INST_HEADER** as their first member.

```
typedef union _SPECIAL_INST_PARAM
{       // Compile-time Static variables.
        // This structure maps to .rcd description
                                        // off, sz, ob.sz
    SPECIAL_INST_HEADER  paramHeader;   //  0   12
    SPECIAL_INST_COMMAND paramCommand;  //  0   40
    SPECIAL_INST_PORT    paramPort;     //  0   24
} SPECIAL_INST_PARAM;                   //      40 ==
                                        //         sizeof(SPECIAL_INST_PARAM)
```

Figure 3-7. **SPECIAL_INST_PARAM** section of the **driver.h** file.

The **SPECIAL_INST_HEADER** structure Figure 3-8) has two members required by all functions — a function ID and return status. We recommend that you always use the **SPECIAL_INST_HEADER** structure as the first member of any custom parameter structure you create. This will simplify your structure design by placing **FunctionID** and **ofsStatus** at offset zero and four of all parameter structures, and help minimize total memory allocated as you add custom parameters.

```
#pragma BYTE_ALIGN( _SPECIAL_INST_HEADER ) // Must be first block in all
                                           // parameter blocks
typedef struct _SPECIAL_INST_HEADER
{       // Compile-time Static variables.
        // This structure maps to .rcd description
                                           // off, sz, ob.sz
    UINT16          FunctionId;            //  0    2   2L
                                           //          PT_CONST-->UINT16, _SIZE 2L
    UINT16          align;                 //  2    2
    UIOTREF2UINT16  ofsStatus;             //  4    4   2L
                                           // PT_REF    --> tag's offset in UIOT
    UIOTREF2UINT16  ofsResult;             //  8    4   2L
                                           // PT_REF    --> tag's offset in UIOT
} SPECIAL_INST_HEADER;                     //      12 ==
                                           //         sizeof(SPECIAL_INST_HEADER)
```

Figure 3-8. **SPECIAL_INST_HEADER** section of the **driver.h** file.

For definitions of the other structures, see the **driver.h** file in **\Program Files\VLC\Development\Ctoolkit\Inc** folder. Although the C-Toolkit sample project doesn't need the complexity of the structures defined in this file, we provided the architecture to handle more complex modules you may need to define.

If you implement a driver with multiple functions, the VLC Runtime sets **FunctionID** to the value you set for the called function (see *NET_FUNC_TYPE* on page 36). Then, your C code can use the **FunctionID** to access a particular section of code and **SPECIAL_INST_** structure. By unioning each function's **SPECIAL_INST_HEADER** structure to offset zero, you minimize the total space required for your driver.

3.2.2 RCD File Definition

The ***<module_name>*.rcd** file is a resource file that defines the configuration page of the **Edit Special Function** dialog or the configuration dialog (see Figure 3-9) used for RLL special functions for your driver-based functions. This is a text file that contains VLC-specific resources, and you compile it using the MS resource compiler. You can find the prototype, **Ctoolkit.rcd**, in the **\Program Files\VLC\Development\Ctoolkit\Gui** folder.

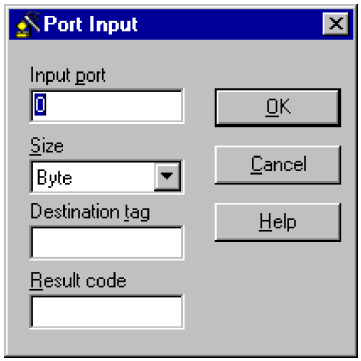

Figure 3-9. The **Port Input** configuration dialog for the C-Toolkit sample project.

You must customize this file for your module because it provides:

- Definitions of configuration pages or configuration dialogs.
- How to interpret data values provided by the configuration pages or configuration dialogs.
- Relationship of configuration fields to members of the **SPECIAL_INST** structure.

As in all resource files, the ***<module_name>*.rcd** file has paragraphs that define the driver resources. The paragraphs are:

DRIVER_DLL_TYPE Defines the module filename and licensing information.

NETWORK_DLL_TYPE Defines the module name, module ID, and a brief description of the module.

NET_FUNC_TYPE Contains a list of functions implemented in the module (driver).

NET_PARAM_TYPE Each function defined in the module has one of these paragraphs to define its list of parameters and their formats.

NET_CFGLIST_TYPE Defines miscellaneous lists as required.

These VLC-specific paragraphs form a hierarchical tree (see Figure 3-10). You must follow this layout closely, since the RC compiler is very unforgiving, and this is currently a manual process.

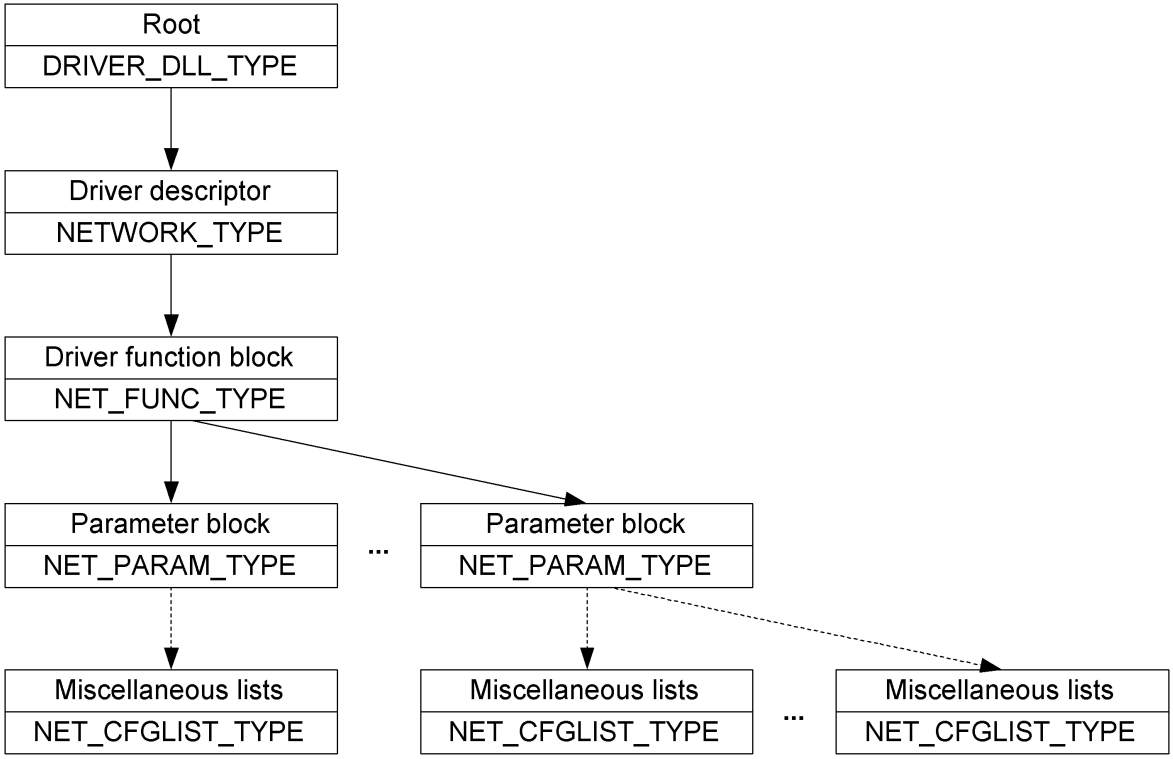

Figure 3-10. VLC-specific paragraphs for resource must appear in this tree hierarchy.

RCD file Editing Guidelines

There are a few important consideratons that you should keep in mind when you edit the **<*module_name*>.rcd** file. They are:

- Make sure that you byte align all resources.
- To ensure the integrity of this structure, the VLC Runtime checks a **DRIVER_SENTINEL** in pre-defined locations.
- The resource compiler interprets numbers as 2-byte values. When the RCD format requires 4-byte values, make sure that you append an "**L**" suffix or "**,0**" to force the compiler to use four bytes.
- The resource compiler accepts commas, spaces, or carriage returns between items. Although it's not mandatory, it is always safer to use a comma between items.
- Plus (+) and minus (-) are valid binary or unary operators for numbers and **#define** definitions.
- Multiply (*) and divide (/) operators are not supported by the resource compiler. The compiler ignores these operators without providing an error message or warning.
- The resource compiler supports strings, but doesn't add the terminating null (zero) value. All string constants you specify must explicitly include a terminating null, for example: **"sample string\0"**
- Resources require the **SPECIAL_INST** structure, but the resource compiler doesn't support the **struct** keyword. Therefore, the **driver.h** file defines the structure a second time in terms of offsets and sizes. You must ensure that the two definitions are the identical. The RCD file uses the **FNC_...** definitions that appear at the end of the **driver.h** prototype file.

The following sections define each paragraph of the RCD file.

DRIVER_DLL_TYPE

This paragraph is the root of the RCD tree. Its identifier **DRIVER_DLL_ID** is defined at the VLC level. Figure 3-11 shows the prototype paragraph from the **Ctoolkit.rcd** file. Do not modify this paragraph.

```
// Do not modify this paragraph.
DRIVER_DLL_ID DRIVER_DLL_TYPE LOADONCALL DISCARDABLE
BEGIN
DRVINST_CONTRACT,
DRIVER_VERSION,
“\270\245\311\222\324\225\355\374”,
1,
NETWORK_TYPE, THIS_DRIVER,
DRIVER_SENTINEL,
0x30FFDD2BL,
END
```

Figure 3-11. **DRIVER_DLL_TYPE** paragraph of the **Ctoolkit.rcd** file.

NETWORK_TYPE

This paragraph describes the current module. Its identifier **THIS_DRIVER** is defined locally. Figure 3-12 shows the prototype paragraph from the **Ctoolkit.rcd** file.

```
THIS_DRIVER NETWORK_TYPE LOADONCALL DISCARDABLE     // paragraph #2
                                                   // driver associated!
BEGIN
    DriverCTOOLKIT,                    // DrvCookie
    CTOOLKITVERS,                      // Driver Uiot Version
    MAX_DRV_INSTANCES,                 // max # of instances
    "Ctoolkit\0",                      // Driver name
    "The Ctoolkit driver for "
    "the Visual Logic Controller"
    " (" PRODUCT_VERSION ").\r\n"
    "This driver is meant to be a starting point in developing other drivers. "
    "It offers 4 devices with different I/O point combinations and some “
    "special functions at both driver and device levels"
    "\0",                              // Driver Description

    0,
    DRVTAG_DONE,
    0L, 0L, 0L,                        // Expansion
    0,                                 // Count of device names
    0,                                 // NeedsBuffer - <> 0 if requires extra
                                       // buffer area for async transfers
    NET_FUNC_TYPE,                     // p.#3
                                       // RT_USER FuncType - Resource type for
                                       // network/device functions, 0 if none
    DRIVER_FUNC,                       // RR_USER FuncRef - Ref to list of
                                       // network function head, 0 if none
    0,                                 // p.#4
                                       // RT_USER CfgType - Resource type for
                                       // network/device cfg
    0,                                 // RR_USER CfgRef - Ref to structure
                                       // def of network config data, 0 if
                                       // none
    0,                                 // RT_USER ErrorsType - Resource type
                                       // for errors codes the driver can
                                       // return
    DRIVER_SENTINEL
END
```

Figure 3-12. **THIS_DRIVER** paragraph of the **Ctoolkit.rcd** file.

When you create a new module from this prototype paragraph, you must perform the following operations and edits on this paragraph:

[1] Modify the defined macro names **DriverCTOOLKIT** and **CTOOLKITVERS** to be consistent with the names you specified in the **`driver.h`** file.

[2] Enter a short name for the module. This is the name that appears in the **Edit Special Function** dialog **Drivers/Devices** list. Make sure the string is null terminated.

[3] Enter a module description. This description appears in the **Edit Special Function** dialog as the **Driver/Device Description**. Make sure the string is null terminated.

[4] If your driver supports devices, enter the number of devices in the location identified by the commentary.

NET_FUNC_TYPE

This paragraph describes the functions that are available for the driver. Its identifier **DRIVER_FUNC** is defined locally. The first field in this paragraph specifies the number of functions included in the driver. The number of sub-paragraphs must equal this number. Function identifiers, such as **DRVF_COMMAND**, are defined in the **driver.h** file, and are also used by the module Runtime project. Figure 3-13 shows the **Ctoolkit.rcd** prototype paragraph.

```
DRIVER_FUNC NET_FUNC_TYPE LOADONCALL DISCARDABLE       // paragraph #3
BEGIN
    3,                                  // how many functions are described

    DRVF_COMMAND, 0,                    // UINT32  Function id,  fake long
        "Command ctoolkit\0",           // short name
        "Asynchronous special call. "
        "Copies the contents of Write Buffer into Read Buffer 1 character per"
        " second. Possible function status codes:\r\n"
        "0: Function in progress. Keep waiting.\r\n"
        "1: Function completed. See Result.\r\n"
        "3: User restarted this function before the previous call"
        " completed.\r\n"
        "Result codes:\r\n"
        "0: No errors. The command succeeded.\r\n"
        "4: Timeout.\r\n"
        "5: Read buffer too short.\r\n"
        "6: Write buffer too short.\r\n"
        "7: Zero bytes to be transferred.\r\n"
        "\0",
        0L, 0L, 0L, 0L,                 // Expansion
        IDD_DRVF_COMMAND,               // DialogId
        IDHELP,                         // HelpId
        IDH_DRVF_COMMAND,               // HelpIdx
        FNC_SPECIAL_INST_SIZE,          // TotalSize
        NET_PARAM_TYPE, DRVF_COMMAND,   // p.#8
    DRIVER_SENTINEL

        ...... repeat this sub-paragraph for each function beginning
        ...... with DRVF_COMMAND through DRIVER_SENTINEL


DRIVER_SENTINEL
END
```

Figure 3-13. **NET_FUNC_TYPE** paragraph of the **Ctoolkit.rcd** file.

When you create a new module from this prototype paragraph, you must perform the following operations and edits on this paragraph:

[1] Specify the number of functions implemented.

[2] Choose a unique short name for each function. This name must be unique among all functions with your driver, plus you can not use any of the following function names defined by the VLC Control Designer:

- **`"Engineering Unit Conversion\0"`**
- **`"GetDevStatDeviceLevel\0"`**
- **`"Get Driver's Status\0"`**

[3] If the function requires a unique configuration dialog, be sure to specify the correct **IDD_DRVF_COMMAND** for each function.

[4] Specify a unique **IDHELP** as required for the configuration dialog.

[5] Specify a unique help ID, shown as **IDH_DRVF_COMMAND** in Figure 3-13, as required for the configuration dialog.

[6] Specify the size of the **SPECIAL_INST** structure used by the function.

[7] Make sure you specify the correct function ID for the pointer to **NET_PARAM_TYPE** for the function sub-paragraph.

[8] Repeat steps [2] through [7] for each function.

NET_PARAM_TYPE

This paragraph describes the parameters for each function. There is one of these paragraphs for each function. The first field in this paragraph for each function specifies the number of parameters required by the function plus one (for the function ID). The number of sub-paragraphs that follow must equal this number. Function identifiers, such as **DRVF_COMMAND**, are defined in the **`driver.h`** file, and are also used by the module Runtime project. Figure 3-14 shows the prototype paragraph from the **`Ctoolkit.rcd`** file.

```
DRVF_COMMAND NET_PARAM_TYPE LOADONCALL DISCARDABLE          // paragraph #8
BEGIN
    8,                                  // Number of parameters required by
                                        // this function

// 1 - UINT16 FunctionId; Offset FNC_HD_FUNCTIONID, Size FNC_HD_FUNCTIONID_SIZE

        "\0",                           // short name
        "\0",
        0L, 0L, 0L, 0L                  // Expansion
        0,                              // CTRL_ID CtrlId_type - Id of control
                                        // in dll dialog for this param value
                                        // or type if using default dlg
        HELP_IDX_NONE                   // HELP_IDX m_HelpIdx
        FNC_HD_FUNCTIONID,              // DWORD Offset - Offset in config
                                        // block for this param value
        FNC_HD_FUNCTIONID_SIZE,         // DWORD Size - Number of bytes in
                                        // config block for this param value
        // ParamTypes (WORD) Type - How the data is to be processed
        PT_CONST,
        DRVF_COMMAND, 0,                // function's id.  fake long
    DRIVER_SENTINEL
//  2 - UIOTREF2UINT16 ofsStatus; Offset FNC_HD_STATUS, Size FNC_HD_STATUS_SIZE
        "Status\0",                     // short name
        "Status\0",
        0L, 0L, 0L, 0L                  // Expansion
        IDC_E_STATUS,                   // CTRL_ID CtrlId_type - Id of control
                                        // in dll dialog for this param value
                                        // or type if using default dlg
        HELP_IDX_NONE                   // HELP_IDX m_HelpIdx
        FNC_HD_STATUS,                  // DWORD Offset - Offset in config
                                        // block for this param value
        FNC_HD_STATUS_SIZE,             // DWORD Size - size of the parameter
                                        // pointed to by the reference
        // ParamTypes (WORD) Type - How the data is to be processed
        PT_REF,
    DRIVER_SENTINEL
        ..... repeat this sub-paragraph for each additional parameter
        ..... beginning with //2 through DRIVER_SENTINEL
DRIVER_SENTINEL
END
```

Figure 3-14. **NET_PARAM_TYPE** paragraph of the **Ctoolkit.rcd** file.

The first sub-paragraph of each **NET_PARAM_TYPE** structure (see Figure 3-14) defines the function ID and data for the entire function. The comment at the beginning of each sub-paragraph shows the name of the parameter in the UIOT and the names used for the **FNC_** offset and size. The members of the **NET_PARAM_TYPE** structure sub-paragraphs define the relationship between the UIOT structure and the configuration dialog definition (see *RC File Definition* on page 47). In particular, each parameter sub-paragraph specifies the **IDC_** that appears in the configuration dialog definition, and the **FNC_** offset and size in the UIOT. What you specify in these sub-paragraphs determine:

- How the VLC compiler and Runtime interpret control programmer input to the configuration dialog
- Where these input values appear in the UIOT.

Sub-Paragraph Contents

The **NET_PARAM_TYPE** paragraph see *NET_PARAM_TYPE* on page 37) contains a variable number of sub-paragraphs. Each sub-paragraph contains:

- A UINT16 control ID, defined in **`Resource.h`** or **`0`**. The PT_format type specifies the data type of the control.
- A UINT32 offset (in bytes) in the **SPECIAL_INST** structure, where the VLC Runtime stores either a copy of the actual parameter (call by value) or a UIOT offset to the parameter (call by reference).
- A UINT32 parameter size (in bytes), which is interpreted based on the parameter type format (see *Parameter Type Formats* on page 42) as:
  - For parameters passed by value (**PT_CONST**, **PT_STRING**, **PT_VALUE**, **PT_SVALUE**, **PT_VALUE_LIST**, **PT_STRING_LIST**, **PT_FLOATVALUE**), the VLC Runtime stores a copy of the actual parameter at the specified offset. If the actual data is larger than the specified UINT32 parameter size, the VLC Runtime truncates the value. Make sure an appropriate field type is specified in the **SPECIAL_INST** structure at the designated UINT32 offset.

- For parameters passed by reference (**PT_REF**, **PT_SREF**, **PT_FLOATREF**), the VLC Runtime stores the UIOT offset to the actual parameter in **SPECIAL_INST** at the specified offset. The size of the parameter must match the specified UINT32 parameter size. The size of the associated field in the **SPECIAL_INST** structure must be four bytes.

- For arrays passed by reference (**PT_BUFFER**), VLC Runtime stores the UIOT offset to the actual array in **SPECIAL_INST** at the specified offset in four bytes. The next four bytes receive the size of the array, in bytes. As a result the UINT32 parameter size must be set to eight and the associated field in **SPECIAL_INST** structure must be **PTBUFFER** as defined in **`driver.h`**.

VLC Runtime Use of NET_PARAM_TYPE

The VLC Runtime fills in the fields described in the **NET_PARAM_TYPE** structure each time a flow chart or RLL program executes the special function element. The Runtime overwrites any data prevously stored in the UIOT structure members for the special funtion. It is possible (and often desirable) to reserve space in the UIOT parameter block structure, but not describe it in the **NET_PARAM_TYPE** structure. The VLC Runtime ignores this space, and preserves it between special function element executions. You can use this space to retain data between calls of your module. To define additional, reserved space in a special function parameter block, define members in the **SPECIAL_INST** structure (and **FNC_...** **`#define`** statements) in **driver. h**, but don't use the members in this **NET_PARAM_TYPE** structure

In the RCD file, the **NET_PARAM_TYPE** object defines the parameter block, and **NET_FUNC_TYPE** specifies the total size reserved in the UIOT for a specific special function.

Procedures for Creating NET_PARAM_TYPE Paragraphs

When you create a new module from this **NET_PARAM_TYPE** prototype paragraph, you must perform the following operations and edits on this paragraph:

[1] Specify one of these **NET_PARAM_TYPE** paragraphs for each function. Figure 3-14 on page 38 shows the paragraph for the first and second parameter of **DRVF_COMMAND**.

[2] Specify the number of parameters for each function, plus one for the function ID.

[3] Enter a short name for the parameter and a description that appears in the configuration dialog for the parameter field. (Note, for example, the static text title, **Status**, in the configuration dialog in Figure 3-15. The second parameter in Figure 3-14 on page 38 is for the **Status** parameter. The short name is the same as the field name in the configuration dialog.) Parameter names must be unique within each function, however, the same parameter name can appear in different functions. It is desirable to use the same description and parameter short name. This makes it easier for control programmers to interpret error messages from the VLC compiler.

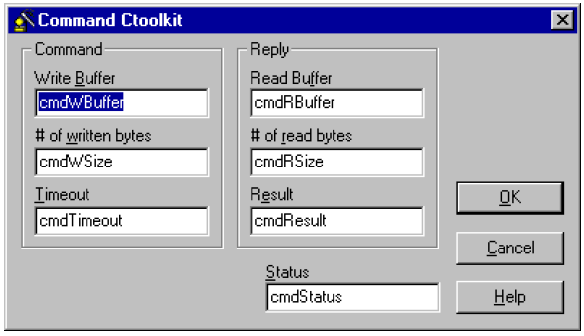

Figure 3-15. The Command configuration dialog from the C-Toolkit sample project.

[4] Use the appropriate **FNC_** name (from the **`driver.h`** file) for the parameter offset and size.

[5] Specify the appropriate parameter type and data required by the parameter type (see *Parameter Type Formats* on page 42).

[6] Repeat steps [3] through [5] for each parameter of a function.

[7] Repeat steps [1] through [6] for each function.

NET_CFGLIST_TYPE

This paragraph describes miscellaneous lists required in the RCD file. A list maps a UINT32 identifier to a null terminated string. Duplicate identifiers in a list are not permitted. The first field in this paragraph is a UINT16 value that specifies the number of items in the list. The list ID is a UINT16 value and is defined locally. Figure 3-16 shows a prototype paragraph from the **Ctoolkit.rcd** file. In this case, the list is the set of values that appear in a drop-down parameter list. The list ID is **ID_CO_LENGTH** and it is used in the definition of the **Input Port Size** parameter in the **DRVF_PORT_INPUT** function definition.

```
IDC_CO_LENGTH NET_CFGLIST_TYPE LOADONCALL DISCARDABLE       // paragraph #6
BEGIN
    2,
    1L,  "Byte\0"                        // UINT32, string
    2L,  "Word\0",
    DRIVER_SENTINEL
END
```

Figure 3-16. **NET_PARAM_TYPE** paragraph of the **Ctoolkit.rcd** file.

Parameter Type Formats

Parameter type formats (PT_formats) appear in the **NET_PARAM_TYPE** paragraph. They are generally associated with a control ID in the function configuration dialog. PT_formats map configuration dialog controls to **SPECIAL_INST** structure members.

PT_formats define parameters passed by value and parameters passed by reference.

The following sections define the data required for each parameter format type. Each section shows a prototype of a section of the **NET_PARAM_TYPE** paragraph in the ***<module_name>*.rcd** file. The sections begin with either a zero or the ID of a control in the configuration dialog. The control IDs used in the sections that follow are only for illustrative purposes.

Constants

Specifies a constant value for the associated parameter.

```
.................................
0,          // no associated control
...
PT_CONST
UINT32      // store this 4-byte value in the parameter
            // block at the specified offset

.................................
```

Strings

Specifies a string constant for the associated parameter.

```
.................................
0,          // no associated control
...
PT_STRING
"string \0"  // store this string in the parameter block
            // at the specified offset
.................................
```

Unsigned Values

Specifies an unsigned value for the associated parameter. When the control programmer configures the function he/she enters either either a number or the name of tag in the configuration dialog edit field. The VLC compiler checks a number to verify that it satisfies the min/max/step restrictions. If the field contains a tag name, the VLC Runtime evaluates the tag every time the control program calls the function. If the tag is a different type than unsigned word of the specified size, the VLC Runtime converts the tag contents.

The VLC compiler or Runtime copies the value to the specified offset in the **SPECIAL_INST** parameter block. When the VLC Control Designer displays the special function element configuration dialog the first time, it displays the default value in the edit field.

```
.................................
IDC_EDIT                    // requests an edit field
...
PT_VALUE                    // implicit %lu format
LONG                        // default
LONG                        // min
LONG                        // max
LONG                        // step
.................................
```

Signed Values

Specifies a signed value for the associated parameter. When the control programmer configures the function he/she enters either either a number or the name of tag in the configuration dialog edit field. The VLC compiler checks a number to verify that it satisfies the min/max/step restrictions. If the field contains a tag name, the VLC Runtime evaluates the tag every time the control program calls the function. If the tag is a different type than signed word of the specified size, the VLC Runtime converts the tag contents.

The VLC compiler or Runtime copies the value to the specified offset in the **SPECIAL_INST** parameter block. When the VLC Control Designer displays the special function element configuration dialog the first time, it displays the default value in the edit field.

```
.................................
IDC_EDIT                        // requests an edit field
...
PT_SVALUE                       // implicit %ld format
LONG                            // default
LONG                            // min
LONG                            // max
LONG                            // step
.................................
```

Combo Fields with Numeric Values

Specifies a drop-down combo field for the associated parameter. When the VLC Control Designer displays the function configuration dialog, the drop-down list shows numbers in the min – max range and the specified step. These numbers appear in the specified format. The control programmer must select a value from the list. The VLC compiler copies the selected value to the **SPECIAL_INST** parameter block at the specified offset.

When the VLC Control Designer displays the special function configuration dialog the first time, it displays the default value in the edit field of the combo control.

```
.................................
IDC_COMBO                       // requests a drop list
...
PT_VALUE_LIST
LONG                            // default
LONG                            // min
LONG                            // max
LONG                            // step
"format %ld \0"                 // "printf" like format
.................................
```

Combo Fields with String Values

Specifies a drop-down combo field for the associated parameter. When the VLC Control Designer displays the function configuration dialog, the drop-down list shows text strings. These strings appear in the order defined by the **NET_CFGLIST_TYPE** pointed to by ***<list_id>***. The control programmer must select a value from the list. The VLC compiler copies the selected value to the **SPECIAL_INST** parameter block at the specified offset.

When the VLC Control Designer displays the special function configuration dialog the first time, it displays the default value in the edit field of the combo control pointed to by the **LONG** default ***<item_id>***.

```
.....................................................................
  IDC_COMBO                         // requests a drop list
  ...
  PT_STRING_LIST
  LONG                              // UINT32 default
                                    // <item_id>
  NET_CFGLIST_TYPE, <list_id>,      // pointer to a
                                    // NET_CFGLIST_TYPE
                                    // paragraph
  ..................................
```

Floating Point Values

Specifies a floating point value for the associated parameter. When the control programmer configures the function he/she enters either either a floating point number or the name of tag in the configuration dialog edit field. If field contains a tag name, the VLC Runtime evaluates the tag every time the control program calls the function. If the tag is a different type than floating point, the VLC Runtime converts the tag contents.

The VLC compiler or Runtime copies the value to the specified offset in the **SPECIAL_INST** parameter block. The parameter size in the **SPECIAL_INST** must be **8L** and the field type must be **double**.

```
  ..................................
  IDC_EDIT                          // requests an edit field
  ...
  PT_FLOATVALUE                     // No default because
                                    // float numbers are not
                                    // supported by the RC
                                    // compiler
  ..................................
```

Referenced Unsigned Values

Specifies a pointer to an unsigned value for the associated parameter. When the control programmer configures the function he/she enters the name of tag in the configuration dialog edit field. The VLC compiler checks the tag to verify that it is an unsigned word and satisfies the size requirement specified.

The VLC Runtime copies the UIOT offset of the tag to the specified offset in the **SPECIAL_INST** parameter block. The parameter size in **SPECIAL_INST** must be four bytes, and the size defined in the **NET_PARAM_TYPE** paragraph is the size of the parameter pointed to by the tag referenced.

```
.................................
IDC_EDIT                        // requests an edit field
...
PT_REF                          // size of the referenced
                                // object (1L, 2L or 4L)
.................................
```

Referenced Signed Values

Specifies a pointer to a signed value for the associated parameter. When the the control programmer configures the function he/she enters the name of tag in the configuration dialog edit field. The VLC compiler checks the tag to verify that it is a signed integer and satisfies the size requirement specified.

The VLC Runtime copies the UIOT offset of the tag to the specified offset in the **SPECIAL_INST** parameter block. The parameter size in **SPECIAL_INST** must be four bytes, and the size defined in the **NET_PARAM_TYPE** paragraph is the size of the parameter pointed to by the tag referenced.

```
.................................

IDC_EDIT                        // requests an edit field
...
PT_SREF                         // size of the referenced
                                // object (1L, 2L or 4L)
.................................
```

Referenced Floating Point Values

Specifies a pointer to a floating point value for the associated parameter. When the the control programmer configures the function he/she enters the name of tag in the configuration dialog edit field. The VLC compiler checks the tag to verify that it is a float type.

The VLC Runtime copies the UIOT offset of the tag to the specified offset in the **SPECIAL_INST** parameter block. The parameter size in **SPECIAL_INST** must be four bytes, and the size defined in the **NET_PARAM_TYPE** paragraph must be **8L** to match the size of the referenced tag.

```
.................................
IDC_EDIT                        // requests an edit field
...
PT_FLOATREF                     // The referenced
                                // object is "internal
                                // float" size 8
.................................
```

Referenced Internal Arrays

Specifies a pointer to an array for the associated parameter. When the the control programmer configures the function he/she enters the name of tag in the configuration dialog edit field. The VLC compiler checks the tag to verify that it is an array.

The VLC Runtime copies the UIOT offset of the tag and the array size (in bytes) to the specified offset in the **SPECIAL_INST** parameter block. The parameter size in the **SPECIAL_INST** must be **PTBUFFER** (defined in **driver.h**), and the size defined in the **NET_PARAM_TYPE** paragraph must be **8L** to provide space for the pointer and array size.

Your runtime code can check array boundaries based on the array size provided.

```
.................................
IDC_EDIT                        // requests an edit field
...
PT_BUFFER                       // The referenced object
                                // is "internal array".
                                // Size is 4+4= 8L.
.................................
```

3.2.3 RC File Definition

The RC file is similar to any MS DevStudio project resource file. It includes definitions for the configuration page of the **Edit Special Function** dialog or the configuration dialog used for RLL special functions. It also includes definitions of error messages that you want the VLC Runtime to pass along to control program developers and end users.

The **IDC_** names used for configuration dialog controls must appear in the RCD file **NET_PARAM_TYPE** structure sub-paragraph for each parameter. Using the same names, relates dialog input to specific UIOT offsets and sizes (see *NET_PARAM_TYPE* on page 37).

Configuration Dialog Definitions

The **Ctoolkit.rc** file in **\Program Files\VLC\Development\Ctoolkit\Gui** illlustrates three dialog definitions. The first one, shown in Figure 3-17, has seven edit fields and standard **OK**, **Cancel**, and **Help** pushbuttons.

```
IDD_DRVF_COMMAND DIALOG DISCARDABLE  18, 18, 264, 125
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Command Ctoolkit"
FONT 8, "MS Sans Serif"
BEGIN
    GROUPBOX        "Command",IDC_STATIC,6,2,94,90,WS_GROUP
    LTEXT           "Write &Buffer",IDC_STATIC,12,14,81,8
    EDITTEXT        IDC_E_WBUFFER,12,23,81,13,ES_AUTOHSCROLL | WS_GROUP
    LTEXT           "# of &written bytes",IDC_STATIC,12,39,81,8
    EDITTEXT        IDC_E_WLENGTH,12,48,81,13,ES_AUTOHSCROLL | WS_GROUP
    LTEXT           "&Timeout",IDC_STATIC,12,64,81,8
    EDITTEXT        IDC_E_TIMEOUT,12,73,81,13,ES_AUTOHSCROLL | WS_GROUP
    GROUPBOX        "Reply",IDC_STATIC,106,2,94,90
    LTEXT           "Read Bu&ffer",IDC_STATIC,112,14,81,8
    EDITTEXT        IDC_E_RBUFFER,112,23,81,13,ES_AUTOHSCROLL | WS_GROUP
    LTEXT           "# of &read bytes",IDC_STATIC,112,39,81,8
    EDITTEXT        IDC_E_RLENGTH,112,48,81,13,ES_AUTOHSCROLL | WS_GROUP
    LTEXT           "R&esult",IDC_STATIC,112,64,81,8
    EDITTEXT        IDC_E_RESULT,112,73,81,13,ES_AUTOHSCROLL | WS_GROUP
    LTEXT           "&Status",IDC_STATIC,119,97,77,8
    EDITTEXT        IDC_E_STATUS,119,106,81,13,ES_AUTOHSCROLL | WS_GROUP
    DEFPUSHBUTTON   "&OK",IDOK,208,63,50,14,WS_GROUP
    PUSHBUTTON      "&Cancel",IDCANCEL,208,84,50,14
    PUSHBUTTON      "&Help",9,208,105,50,14
END
```

Figure 3-17. **Command Ctoolkit** dialog definition in the **Ctoolkit.rc** file.

Error Strings

The RC file may include module-specific strings that you can use as error messages. You can customize these module-specific string with up to four parameters. To take advantage of this feature, the string must contain the **%1**, **%2**, **%3** and/or **%4** formal parameters. The VLC Runtime fills-in the **%1** and **%2** formal parameters with the module type and instance name. The **%3** and **%4** formal parameters are free for you to use as desired.

All transition notification functions (see *State-Transition Functions* on page 68) contain the **P_ERR_PARAM** (defined in **dcflat.h**) parameter (see Figure 3-19). This pointer provides access to the Param3 and Param4 buffers. Your driver program needs to store parameters 3 and 4 as zero terminated strings.

```
#pragma BYTE_ALIGN( _ERR_PARAM )
typedef struct _ERR_PARAM
{
    UINT16 DriverErr;          /* Error msg resource Id in the driver .io3 */
    UINT16 Spare;
    char   Param3[68];         /* text to replace the formal param %3 */
    char   Param4[68];         /* text to replace the formal param %4 */
                                        /* */
} ERR_PARAM, far* P_ERR_PARAM;
```

Figure 3-18. **P_ERR_PARAM** definition in the **dcflat.rc** file.

Figure 3-19 shows the error strings defined in the **Ctoolkit.rc** file for the prototype driver.

```
STRINGTABLE DISCARDABLE
BEGIN
    IDS_CTOOLKIT_HW_TEST    "Hardware test failure on '%1'"
    IDS_CTOOLKIT_INVALID_ADDERSS "Device '%3' has an invalid address."
    IDS_CTOOLKIT_DEVICE_OFFLINE "Device '%3' is offline."
    IDS_CTOOLKIT_TIMEOUT    "Ctoolkit timeout."
    IDS_CTOOLKIT_READ_SIZE  "Read  buffer too short."
    IDS_CTOOLKIT_WRITE_SIZE "Write buffer too short."
    IDS_CTOOLKIT_RW_ZERO    "Zero bytes to be transferred."
    IDS_CTOOLKIT_DPR_OUT    "Out of DPR image."
END
```

Figure 3-19. Error strings stored in the **Ctoolkit.rc** file.

3.2.4 Other Files in the GUI Project

In addition to **Ctoolkit.rc** (see *RC File Definition* on page 47) and **Ctoolkit.rcd** (see *RCD File Definition* on page 30), the **\Program Files\VLC\Development\Ctoolkit\Gui** folder includes the following prototype files:

Ctoolkit.cpp — This file includes several standard functions required by the VLC compiler. The only changes required to this file for your module are to change the file name, a few comments in the file, and the **static char** statement near the beginning of the file.

Gui.dsp — The Microsoft Developer Studio defines and uses this file for the GUI project. Even though the MS DevStudio places the comment " ** Do not edit this file **" at the beginning of the file, you must change all references to **Ctoolkit** to your new module name.

resource.h — Generated by MS DevStudio. No changes required to this file.

stdafx.h — Standard include file for the VLC Control Designer. Change references to **Ctoolkit** to your new module name.

3.2.5 Other Files in the Module Inc Folder

The **\Program Files\VLC\Development\Ctoolkit\Inc** folder includes files that both the GUI and Runtime MS DevStudio projects require. In addition to **Driver.h** (see *Driver.h File Definition* on page 22), this folder includes the following prototype files:

Ctoolkit.hh — This file specifies **#define** statements required for VLC Control Designer online help implementation. You must change the name of this file for your module. For more information on requirements for this file, see *Creating Online Help for Your Driver* on page 57.

Errors.h — This file specifies **#define** statements that relate driver-specific error names to a numeric value. These are the error codes you should use when you implement **rtOpen()**, **rtReload()**, **rtOnLine()**, **rtOffLine()**, **rtClose()**, and **rtSpecial()** (Chapter 4, *Function Reference*).

Version.h — This file lets you define a major and minor version number for your module. Change all references to **Ctoolkit** to your new module name. Update **PRODUCT_MAJOR_VERSION** or **PRODUCT_MINOR_VERSION** whenever you want to force the control programmer to recompile VLC projects that use your module.

3.2.6 Files in the Runtime Project

The **\Program Files\VLC\Development\Ctoolkit\Runtime** folder includes files that the Runtime MS DevStudio project requires. This folder includes the following prototype files:

Auxrut.c — This file includes functions that let you create and manage background threads for your module. For examples on using these functions, see the **Task.c** file.

Auxrut.h — Definition file for **Auxrut.c**.

Card.c	This file illustrates examples of C functions used to implement driver-based special functions in the VLC Control Designer.
Card.h	Definition file for **Card.c**.
Ctoolkit.c	This file illustrates how you implement the event-driven functions required to interface your module to the VLC Runtime. Look in this file for examples of functions described in Chapter 4, *Function Reference*.
Ctoolkit.h	Definition file for **Ctoolkit.c**.
Release.cmd	This file is a prototype of the NT command file that deletes the module from INtime memory.
Replace.cmd	This file is a prototype of the NT command file that deletes the module from INtime memory, and loads **<*module_name*>.rt3** from **\Program Files\VLC\Driver 6_x\Bin\Jobs** instead.
Runtime.dsp	The Microsoft Developer Studio defines and uses this file for the Runtime project. Even though the MS DevStudio places the comment " ** Do not edit this file **" at the beginning of the file, you must change all references to **Ctoolkit** to your new module name.
stdafx.h	Standard include file for the VLC Control Designer. Change references to **Ctoolkit** to your new module name.
Task.c	This file illustrates how you create and manage background thread processing for your module.
Task.h	Definition file for **Task.c**.

3.3 Coding Requirements

The files in the **\Program Files\VLC\Development\Ctoolkit\Runtime** folder illustrate the topics discussed in this section. Make sure you thoroughly understand this section and the prototype files before you begin coding your driver-based functions.

At execution time all functions enter the runtime code via **rtSpecial()** entry point. The VLC Runtime always calls **rtSpecial()** from the same thread, so reentrancy is not an issue.

It is your responsibility, as the module programmer, to provide a **FunctionID** field to identify the actual function that needs to be executed.

In order to preserve VLC determinism, execution time inside **rtSpecial()** must be less than a VLC scan cycle. Depending on **FunctionId**, **rtSpecial()** can either execute the function or post the function for execution to another thread. Functions executed by **rtSpecial()** are synchronous with the VLC project.

The VLC Runtime engine executes functions posted to another thread asynchronously with regard to the VLC project. It is your responsibility to make sure the executing thread has a lower priority than **rtSpecial()** and the mainline thread does not wait too long for semaphores.

Except for the **rtSpecial()** execution time requirement, the VLC Runtime does not introduce any other restrictions, nor does it provide any helping mechanisms to implement asynchronous processing. You have complete freedom in designing and implementing asynchronous routines. However, the Ctoolkit workspace provides a proven mechanism that implements asynchronous processing that you can adapt for your requirements.

WARNING: *Steeplechase Software assumes no responsiblity for your use of the routines or techniques illustrated in the Ctoolkit prototype project.*

In the design provided by the Ctoolkit workspace, the parameter block required by asynchronous special functions is almost three times as long as the user data area. This is due to multi-tasking complications — background tasks should not modify the part of the parameter block which the VLC program modifies.

To eliminate problems synchronizing the background task with VLC, when special functions complete in the background the task doesn't transfer the termination flag into parameter blocks until the VLC Runtime calls **rtInput()**. The sample project uses a function called **VerifyDoneList()** (see **Task.c**) to complete the transfer. This function updates the status tag that the control programmer specified in the configuration dialog. Once the status tag shows that the operation has completed, the flow chart or RLL program can read valid data from the [out] parameters.

Because of timing requirements, asynchronous functions need control programmer collaboration. The calling flow chart or RLL program needs to loop on a decision element following the special function element until the the status tag becomes non zero (see Figure 3-20).

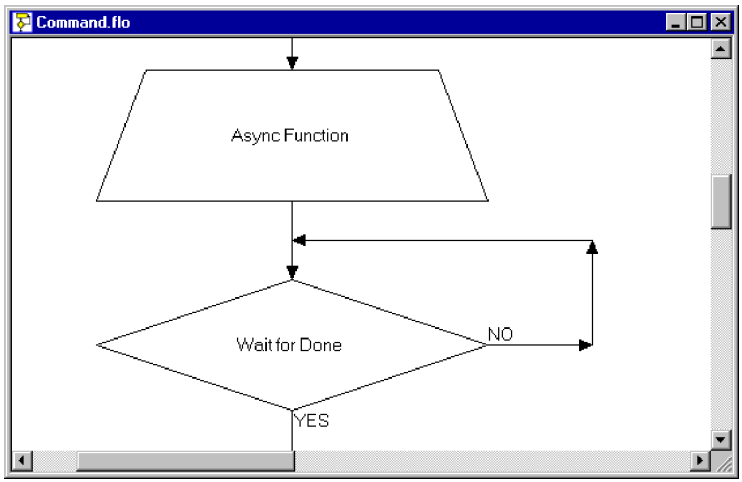

Figure 3-20. An example of an asynchronous special function element followed by a decision element.

Using Local Data

This section provides some guidelines for specifying and using local data in your modules. Key points to remember are:

- You can define **auto** local variables, and access them according to standard C language rules. The contents of these variables are available to the current function only, and are lost between calls.

- The **SPECIAL_INST** structure contains two types of fields:

  - Actual parameters — Fields described in **NET_PARAM_TYPE** (see *NET_PARAM_TYPE* on page 37). The VLC Runtime updates these fields every time it calls the **rtSpecial()** entry point.

  - Function instance local data — Fields not described in **NET_PARAM_TYPE**. You can define such fields here. The contents of these fields can only be seen by the function instance associated with the calling special function element, The VLC Runtime preserves this data between calls, and deletes the data block when the UIOT gets unloaded. On the first call after loading a VLC project, the VLC Runtime sets these fields to zero.

- You can define members of the **DRIVER_INST** structure (see *DRIVER_INST Structure* on page 24). All functions belonging to the same module instance can access the contents of these members. The contents of the **DRIVER_INST** structure are preserved between calls, and are lost when the VLC Runtime unloads the UIOT. On the first call after loading a VLC project, the VLC Runtime sets these members to zero.

- You can define **static** allocated variables in the data segment. These variables are initialized and accessed according to standard C language rules. The contents of these variables are shared by all instances of a C module. The "=" C language static initialization statement only works when the VLC Runtime first loads the C module. To ensure correct initial values on every project load, you should use **rtLoad()** to initialize all static allocated variables.

Call Back Functions

Your C module may call any INtime API and most C standard library functions. In addition the C-Toolkit environment implements these functions:

GetPathInfo	Provides the absolute path to selected files.
SetDebuggingFlag	Disables the VLC watchdog timer. Use this function to permit debugging of your module without interference by the watchdog timer.
spawnv	Spawns an NT process and waits for its completion.

The prototypes for these functions can be found in the **csflat.h** file in the **\Program Files\VLC\Development\Inc** folder. For more information on these functions, Chapter 4, *Function Reference*.

INtime 2.0 offers standard file I/O access through standard Windows functions. Because the INtime file access uses Windows calls, it isn't hard real-time control compliant. The C-Toolkit replaces some standard I/O functions with versions that are hard real-time compliant. You can only use I/O functions provided by the C-Toolkit.

WARNING: *Don't attempt to use any other standard C language I/O functions. They won't work in the VLC Runtime environment.*

The C-Toolkit development environment implements the following file I/O functions (you can find the prototypes for these functions in the **csflat.h** file):

```
FILE*     fopen ( const char* fname, const
          char* pParameter );     // binary only!!!
int       fclose ( FILE *fh );
size_t    fread( void *pbuf, size_t size, size_t count,
          FILE *fh );
size_t    fwrite ( const void *pbuf, size_t size,
          size_t count, FILE *fh );
int       fseek ( FILE* fh, long pos, int origin );
long      ftell ( FILE* fh );
int       open ( const char* filename, int oflag, ... );
int       close ( int fh );
```

```
int       read (  int fh, char* buffer,
          unsigned int count );
int       write ( int fh, const char* buffer,
          unsigned int count );
long      lseek ( int fh, long offset, int origin );
long      tell (  int fh );
```

3.4 Creating Online Help for Your Driver

The **Ctoolkit.dsw** workspace (in **\Program Files\VLC\Development\Ctoolkit**) does not include a help file, but it does provide hooks for a minimal set of help topics. In this prototype project, we have created a single help topic for each driver-based function. The help identifiers appear in the **Ctoolkit.hh** file. When you create your new module, you rename this file to **<*module_name*>.hh**.

The **NET_FUNC_TYPE** paragraph in the RCD file has a sub-paragraph for each function you implement (see *NET_FUNC_TYPE* on page 36). There is a one-to-one relationship between:

- Each function you implement.
- **NET_FUNC_TYPE** sub-paragraphs.
- **IDD_...** function configuration dialogs that have a **Help** button.
- **IDH_...** help IDs that appear in the **<*module_name*>.hh** file.

You can use any development tool you want to create your help file. The VLC Runtime requirements to have your help available to the control programmer are:

- Set the topic ID equal to the **IDH_...** names used in the **NET_FUNC_TYPE** sub-paragrap.
- Use **<*module_name*>.hlp** as your help filename.
- The help file must appear in the same directory as the **<*module_name*>.io3** file.

3.5 Procedures to Follow

This section provides step-by-step procedures that you should follow to create and maintain your driver-based functions with the C Toolkit.

3.5.1 Creating a New Driver

Carefully follow these steps to create your new module.

[1] Copy the entire folder structure:
<drive>:\Program Files\VLC\Development\Ctoolkit
to:
<drive>:\Program Files\VLC\Development*<module_name>*

where ***<module_name>*** is the name of your new module. You can choose a name of up to eight characters.

The remaining steps all apply to your new folder structure, and **.** refers to the
<drive>:\Program Files\VLC\Development*<module_name>*
folder.

[2] Delete the **.\Samples** folder.

[3] If they exist, delete the temporary folders **Release** and **Debug** from **.**, as well as from the **.\Gui** and **.\Runtime** folders.

[4] Delete all ***.aps**, ***.ncb**, ***.plg**, and ***.opt** files from the **.** folder and all subfolders.

WARNING: *DO NOT delete the* **Gui.dsp** *and* **Runtime.dsp** *files.*

[5] Rename all **Ctoolkit.*** files to ***<module_name>.***.

[6] In all remaining files (including **Gui.dsp** and **Runtime.dsp**), replace all instances of (the following are case sensitive —

make sure you use the correct upper/lower case names for your replacements):

- **`CTOOLKIT`** to ***`<MODULE_NAME>`***
- **`ctoolkit`** to ***`<module_name>`***
- **`Ctoolkit`** to ***`<Module_name>`***

[7] Edit the **`driver.h`** file (see *Driver.h File Definition* on page 22). Make sure that you specify a unique **DriverId**. This is a number that you need to select fromthe range of numbers received from Steeplechase (see *Note*, below).

> ***NOTE:*** *Each buyer of C-Toolkit receives a range of 256 values from Steeplechase to ensure unique driver IDs among all C-Toolkit developers.*

[8] Edit the **`<module_name>.rcd`** file (see *RCD File Definition* on page 30). Make sure you follow the recommendations in *RCD file Editing Guidelines* on page 32.

[9] Edit the **`<module_name>.rc file`** (see *RC File Definition* on page 47).

[10] Review all the other files to ensure that they provide appropriate information for your project.

[11] Program your **`<module_name>.c`** file located in the **`.\Runtime`** folder.

[12] Compile the new module.

3.5.2 Debugging a Driver

A driver consists of two components: **`<module_name>.io3`** and **`<module_name>.rt3`**. Most of the code you develop resides in the **`<module_name>.rt3`** component, which is located in the **`\Program Files\VLC\Driver 6_x\Bin\Jobs`** folder. The only code in the **`<module_name>.io3`** component is trivial and resides in the **`Gui\<module_name>.cpp`** file. To debug the **`<module_name>.rt3`**, you

can either place debug messages in your code, or you can purchase the optional VLC Realtime Debugger.

Debugging the Driver Runtime Code

If you do not have the VLC Realtime Debugger, you can debug your driver by monitoring VLC Runtime variables and by adding debug string messages to your code. You can implement debug strings using standard C library functions **printf()** and **puts()** or **csPrintf()**. All printed messages appear in an NT window. NT creates one window for each Runtime thread when the thread prints its first line. Figure 3-21 shows coding techniques that you can use to write a debug message.

```
//Sample 1:
int result = foo();            // the value we what we want to monitor
printf( "foo() return code is %d \n", result);
                               // make sure this is a CR, LF terminated line.
Delay( 1 );                    // go to sleep for 1 ms to let NT print
                               // the message


//Sample2:
char buf[5];                   // the 5 values we want to print in the same
                               // line:
for( i=0;  i<5; i++ )
printf( "%02x",  buf[ i ] );    // the C standard library is
                               // building up the line in an internal buffer
printf( " \n");                // send the whole line to NT.
Delay( 1 );                    // go to sleep for 1 ms to let NT print the
                               // message
```

Figure 3-21. Sample debug print statement.

Using the VLC Realtime Debugger

If you have purchased the optional VLC Realtime Debugger, follow these steps:

[1] Start the VLC Control Designer.

[2] If you have a running project, Stop and Unload it.

[3] Release the module (driver) you want to debug. To do this, open an NT command window and execute:
<drive>:\Program Files\VLC\Development\<*module_name*>\Runtime\Release.cmd

or double-click on the **Release.cmd** file in Windows Explorer.

[4] Start the VLC Realtime Debugger.

[5] Select **File/Load** in VLC Realtime Debugger.

[6] Select:
<drive>:\Program Files\VLC\Development\<*module_name*>\Runtime\Debug\<*module_name*>.rta

[7] Select the **Command** check box, and make sure the command is **go :<*module_name*>.rtLoad**.

[8] Click **OK**.

[9] Switch to the VLC Control Designer and open the project that uses the new driver.

[10] Load and Run the project. The VLC Realtime Debugger assumes control at the automatically entered **rtLoad()** break-point.

NOTE: *The VLC Realtime Debugger does not support long filenames. Make sure all your files and the full path use the 8.3 format. The* **Browse...** *button located in the* **Load** *dialog (accessed from* **File/Load...***) may create a path with more characters than the program can load. If the error "System - filename too long appears, it may be that the path is longer than the 63 character maximum. The path character count begins afer c:\ drive letter and first folder separator.*

3.5.3 Using a New Driver

Follow these steps to begin using your new **<*module_name*>.rt3** driver.

[1] Make sure that you specified a "Release" build in the MS Dev-Studio. Verify that the **<*module_name*>.rt3** is in the **VLC\Driver 6_x\Bin\jobs** folder.

[2] Start the VLC Control Designer.

[3] If you have a running project, Stop and Unload it.

[4] Replace the module (driver) you want to debug. To do this, open an NT command window and execute:
<drive>:\Program Files\VLC\Development\<*module_name*>\Runtime\Replace.cmd

or double-click on the **Replace.cmd** file in Windows Explorer.

[5] Switch to the VLC Control Designer and open the project that uses the new driver. Load and Run the project to use the new driver.

3.5.4 Using a New Configuration Dialog for a Driver

Follow these steps to begin using your new **<*module_name*>.io3** module.

[1] Make sure that you specified a "Release" build in the MS Dev-Studio. Verify that the **<*module_name*>.io3** is in the **VLC\Driver 6_x\Bin** folder.

[2] Start the VLC Control Designer.

[3] Open the project that uses the new driver.

[4] Any changes that were made to the configuration dialogs are available in the VLC Control Designer or Maintainer.

3.5.5 Distributing a Driver to Others

To distribute a driver that you created, you must use an installation creation package from a third-party vendor (for example, InstallShield or Wise). The install script you create must perform the following functions:

[1] Search the registry for the **RegPath** key value in **HKEY_LOCAL_MACHINE\SOFTWARE\Runtime**. If this key or its value isn't present, issue an error message something like, "VLC not installed. Please install VLC first and try again." Then exit the installation.

[2] If found, use the **RegPath** value to locate two additional keys, **DriverNTPath** and **DriverRTPath**. If either of these two keys or their values are missing, issue an error message something like, "VLC is incorrectly installed. Please re-install and try again." Then exit the installation.

[3] Install ***<module_name>*.io3** in the folder pointed to by the value in **DriverNTPath**. Install ***<module_name>*.rt3** in the folder pointed to by **DriverRTPath**.

[4] Ask permission to reboot the system.

4. FUNCTION REFERENCE

Your driver module must define specific entry points for the VLC Runtime to call. This section provides a function reference for each entry point.

4.1 Functions by Group

The following is a list of VLC C-Toolkit functions organized by category. This section provides a reference to the VLC C-Toolkit functions. The discussion of each function is divided into the following sections:

- **Description**. A summary of the routine's effect immediately following the function name. (There is no sub-head for the description.)
- **Syntax**. Illustrates the syntax of the function with arguments.
- **Arguments**. A complete description of the arguments and valid values.
- **Remarks**. A more detailed description of the routine and how it is used with special cases identified in this section.
- **Return Value**. Describes common error return values returned by the routine.

4.1.1 Initialization Function

There is one initialization function that the VLC Runtime engine calls for every driver when the engine first begins execution. It is:

- rtIdentify

rtIdentify

Called only once when the module is launched. This entry point provides identification information for the module including the module ID, name, and version.

Syntax:

int rtIdentify(

P_IDENTITY_BLOCK* *ppIdentityBlock*

);

Arguments

ppIdentityBlock

A data structure with the following members:

Member type	Name	Description
char *	pName	As the name will appear in the **RtConsole** window (see Figure 4-1 in the Remarks, below).
UINT32	DriverVers	Stored in **`io3`**, **`rt3`** and compiled project. All must match
UINT32	DriverId	Unique identifier
UINT32	Reserved[3]	

Remarks

The ID value returned must be unique across all modules. The version of the ***`<module_name>`*`.io3`** file must match the version of ***`<module_name>`*`.rt3`**, as reported to the Runtime engine by the **rtIdentify()** function. Figure 4-1 shows an illustration of the Runtime Console Information page with the C-Toolkit sample module loaded.

The values loaded in DriverId and DriverVers are DriverCTOOLKIT and CTOOLKITVERS defined in the **`driver.h`** file.

Return Value

Always returns zero.

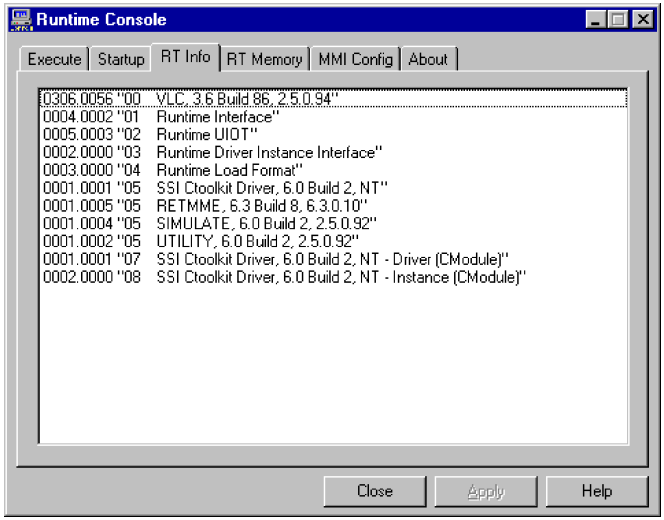

Figure 4-1. An example of the Runtime Console Information page showing the C-Toolkit sample driver loaded.

4.1.2 State-Transition Functions

The VLC Runtime engine calls these functions at specific transition points. Figure 3-1 on page 16 shows a diagram of the state-transitions and when the engine calls each function. The state-transition functions (roughly in the order they are called) are:

- rtLoad
- rtOpen
- rtReload
- rtOnLine
- rtOffLine
- rtClose
- rtUnload

Return Value Processing for *rtOpen*, *rtOnLine*, *rtOffLine*, and *rtClose*

In case of failure, pErrors->DriverERR is the error code. If pErrors->DriverErr is zero, the non-zero returned value is an error code. If the error code matches an IDS_... string in the ***<module_name>*.io3**, the VLC Runtime engine adds that string to the error message. If the IDS_... string contains one or more of the %1, %2, %3, or %4 parameters, the VLC Runtime engine replaces the parameters as follows:

%1	The module name as specified by rtIdentify
%2	The module instance name as specified in the VLC project.
%3	The contents of pErrors.Param3.
%4	The contents of pErrors.Param4.

rtLoad

Called once for every module type. This function provides the module with the specified VLC scan rate in microseconds.

Syntax:

int rtLoad(

UINT32 *ScanRate,*

UINT32* *rDirectCalls*

);

Arguments

ScanRate

The current project scan rate defined in the VLC Control Designer project measured in microseconds.

rDirectCalls

Specifies the whether or not this special function should use direct calls. Valid direct call control bits are:

Name	Value	Meaning
DIRECT_INPUT	1	Use direct call for **rtInput()**.
DIRECT_OUTPUT	2	Use direct call for **rtOutput()**.
DIRECT_SPECIAL	4	Use direct call for **rtSpecial()**.

Remarks

For more information on direct calls, see *Direct Calls* on page 21.

Return Value

Returns zero for successful operation. Any non-zero return code indicates a failure.

In case of failure, the non-zero returned value is an error code. If the error code matches an IDS_... string in the **`<module_name>.io3`**, the VLC Runtime engine adds that string to the error message.

rtOpen

Called once for every instance of a module after successful call to **rtLoad**.

Syntax:

int rtOpen(

LPDRIVER_INST *pNet,*

P_ERR_PARAM *pErrors*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT..

pErrors

Member type	Name	Description
UINT16	DriverErr	Error msg resource Id in the driver **.io3**
UINT16	Spare	
char	Param3[68]	Text to replace the formal param **%3**
char	Param4[68]	Text to replace the formal param **%4**

Remarks

Use this entry point to create a background thread if your driver needs one. If this function fails, it must perform all necessary cleanup before exiting. The Runtime engine won't call **rtClose** when this function fails.

Return Value

Returns zero for successful operation. Any non-zero return code indicates a failure.

See *Return Value Processing for rtOpen, rtOnLine, rtOffLine, and rtClose* on page 68 for more information.

rtReload

Called once for every instance of a module if Control Designer user requests a Load and Go from a stopped state.

Syntax:

int rtReload(

LPDRIVER_INST *pNet,*

P_ERR_PARAM *pErrors*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT..

pErrors

Member type	Name	Description
UINT16	DriverErr	Error msg resource Id in the driver **`.io3`**
UINT16	Spare	
char	Param3[68]	Text to replace the formal param **%3**
char	Param4[68]	Text to replace the formal param **%4**

Remarks

Return Value

Returns zero for successful operation. Any non-zero return code indicates a failure.

rtOnLine

Called once for every instance of a module when the Control Designer user initiates execution of the Runtime project.

Syntax:

int rtOnLine(

LPDRIVER_INST *pNet,*

P_ERR_PARAM *pErrors*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT..

pErrors

Member type	Name	Description
UINT16	DriverErr	Error msg resource Id in the driver **.io3**
UINT16	Spare	
char	Param3[68]	Text to replace the formal param **%3**
char	Param4[68]	Text to replace the formal param **%4**

Remarks

If this function fails, it must perform all necessary cleanup before exiting. The Runtime engine won't call **rtOffLine** when this function fails.

Return Value

Returns zero for successful operation. Any non-zero return code indicates a failure.

See *Return Value Processing for rtOpen, rtOnLine, rtOffLine, and rtClose* on page 68 for more information.

rtOffLine

Called once for every instance of a module when the Control Designer user stops execution of the Runtime project.

Syntax:

int rtOffLine(

LPDRIVER_INST *pNet,*

P_ERR_PARAM *pErrors*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT..

pErrors

Member type	Name	Description
UINT16	DriverErr	Error msg resource Id in the driver **.io3**
UINT16	Spare	
char	Param3[68]	Text to replace the formal param **%3**
char	Param4[68]	Text to replace the formal param **%4**

Remarks

Return Value

Returns zero for successful operation. Any non-zero return code indicates a failure.

See *Return Value Processing for rtOpen, rtOnLine, rtOffLine, and rtClose* on page 68 for more information.

rtClose

Called once for every instance of a module after the VLC Runtime engine stops a project

Syntax:

int rtClose(

LPDRIVER_INST *PNet,*

P_ERR_PARAM *pErrors*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT..

pErrors

Member type	Name	Description
UINT16	DriverErr	Error msg resource Id in the driver **`.io3`**
UINT16	Spare	
char	Param3[68]	Text to replace the formal param **%3**
char	Param4[68]	Text to replace the formal param **%3**

Remarks

This function executes regardless of whether the stop was due to operator command or an error.

Return Value

Returns zero for successful operation. Any non-zero return code indicates a failure.

See *Return Value Processing for rtOpen, rtOnLine, rtOffLine, and rtClose* on page 68 for more information.

rtUnload

Called once for every module type to unload a project.

Syntax:

int rtUnload( );

Arguments

Remarks

Return Value

Returns zero for successful operation. Any non-zero return code indicates a failure.

In case of failure, the non-zero returned value is an error code. If the error code matches an IDS_... string in the **<*module_name*>.io3**, the VLC Runtime engine adds that string to the error message.

4.1.3 Running State Functions

The VLC Runtime engine calls these functions at specific times during the scan cycle or when the Runtime executes a special function element. The running state functions (in the order they are called) are:

- rtInput
- rtSpecial
- rtOutput

NOTE: **rtSpecial** *may be called many times for the same or different functions between calls to* **rtInput** *and* **rtOutput**. *However, the Runtime engine will only call it once for any individual special function element that appears in a flow chart or RLL program.*

rtInput

Called in the running state of the VLC Runtime engine at the beginning (or input phase) of every scan cycle.

Syntax:

int rtInput(

LPDRIVER_INST *pNet,*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT.

Remarks

You can use this entry point to perform operations that need to be synchronized with the state of the VLC Runtime engine. The Runtime calls this entry point at the beginning of every scan cycle.

This function must exit as quickly as possible. No indefinite waits are allowed. See *Direct Calls* on page 21 for information on how to improve execution time for this function.

If the driver encounters problems in this function, set appropriate status flags in pNet that the module can use internally. To let the flow chart program monitor these flags, create appropriate functions, such as **GetModuleSatus**. The VLC Control Designer and Runtime engine impose no special requirements on how to handle these exceptions.

Return Value

Always returns zero.

rtSpecial

Called in the Running state whenever the execution flow reaches a special function element.

Syntax:

int rtSpecial(

LPDRIVER_INST *pNet,*

LPSPECIAL_INST *pData*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT..

pData

Pointer to UIOT parameter block for this instance of a special function.

Remarks

Use this entry point to determine the appropriate driver-based function to execute. Then execute that function-specific code.

This function must exit as quickly as possible. No indefinite waits are allowed. See *Direct Calls* on page 21 for information on how to improve execution time for this function.

If the driver encounters problems in this function, set appropriate status flags in pNet that the module can use internally. To let the flow chart program monitor these flags, create appropriate functions, such as **GetModuleSatus**. The VLC Control Designer and Runtime engine impose no special requirements on how to handle these exceptions.

Return Value

Always returns zero.

rtOutput

Called in the running state of the VLC Runtime engine at the end (or output phase) of every scan cycle.

Syntax:

int rtOutput(

LPDRIVER_INST *pNet,*

);

Arguments

pNet

Pointer to the parameter block for this module in the UIOT..

Remarks

You can use this entry point to perform operations that need to be synchronized with the state of the VLC Runtime engine. The Runtime calls this entry point at the end of every scan cycle.

This function must exit as quickly as possible. No indefinite waits are allowed. See *Direct Calls* on page 21 for information on how to improve execution time for this function.

If the driver encounters problems in this function, set appropriate status flags in pNet that the module can use internally. To let the flow chart program monitor these flags, create appropriate functions, such as **GetModuleSatus**. The VLC Control Designer and Runtime engine impose no special requirements on how to handle these exceptions.

Return Value

Always returns zero.

4.1.4 Service Functions

The C-Toolkit provides special versions of many standard C language I/O functions. It also provides three functions that driver programmers will find useful. These functions are:

- File I/O Functions
- GetPathInfo
- SetDebuggingFlag
- GetPathInfo

File I/O Functions

These functions are replacements for standard C language functions. You can find the prototypes for these functions in the **csflat.h** file:

```
FILE*     fopen ( const char* fname, const
          char* pParameter );     // binary only!!!
int       fclose ( FILE *fh );
size_t    fread( void *pbuf, size_t size, size_t count,
          FILE *fh );
size_t    fwrite ( const void *pbuf, size_t size,
          size_t count, FILE *fh );
int       fseek ( FILE* fh, long pos, int origin );
long      ftell ( FILE* fh );
int       open ( const char* filename, int oflag, ... );
int       close ( int fh );
int       read (  int fh, char* buffer,
          unsigned int count );
int       write ( int fh, const char* buffer,
          unsigned int count );
long      lseek ( int fh, long offset, int origin );
long      tell (  int fh );
```

GetPathInfo

Provides the absolute path to selected files.

Syntax

unsigned int GetPathInfo(

PathType *type,*

PathContent *content,*

char* *path,*

unsigned int *max*

);

Arguments

type

Specifies the path desired.

Name	Value	Meaning
ptJob		path to where the <module_name>.rt3 file has been launched from.
ptVcm		path to the <project_name>.vcm file

content

Type of path desired.

Name	Value	Meaning
pcDir		<absolute path> only
pcDirFname		<absolute path>\<file_name>
pcDirFnameExt		<absolute path>\ <file_name>.extension

path

The character array that will receive the path.

max

The size of the *path* character array.

Remarks

Return Value

SetDebuggingFlag

Use this function to enable Runtime debugging of your module without tripping the VLC watchdog timer.

Syntax

void SetDebuggingFlag(

UINT32 *bDebugging*

);

Arguments

bDebugging

A flag that indicates debug mode when set to 1. Zero indicates normal operation.

Remarks

Set the debugging flag to 1 when stepping through the **rtInput()**, **rtOutput()** or **rtSpecial()** code.

The VLC Runtime engine makes sure the execution time of a project does not exceed a VLC scan cycle. If this happens, the Runtime engine stops the project with an error event. To enable you to step through your code you need to disable the VLC watchdog, by setting the Debugging-Flag.

Return Value

spawnv

Spawns an NT process and waits for its completion.

Syntax

int spawnv(

int *mode,*

const char* *cmdname,*

const char** *argv*

);

Arguments

mode

P_WAIT (the only implemented value)

cmdname

Full path and file name to the executable file.

**argv[0]*

Full path and file name to the executable file.

**argv[1...]*

Parameter list for the target NT process.

Remarks

Return Value

The return code from the NT process.

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W