U.P.M.A.C.S.

User Programmable Monitor, Alarm, and Control Software

version 6.0
Development System

DEVELOPING DEVICE DRIVERS

UPMACS Communications Inc.

January 6, 2005

Table of Contents

DATA OBJECT SOURCES   61

DEVELOPING U.P.M.A.C.S. DEVICE DRIVERS

Introduction

The U.P.M.A.C.S. Development System allows you to develop powerful and flexible device driver for use in station files. You can develop device drivers right in your station file, or you can import them from device driver libraries. The development system also allows you to make your own device driver libraries.

A device driver library is essentially a station file that contains no objects other than device drivers. If you have created only device drivers, you can save your station file as a device driver library with the extension *.upmacs-drivers by selecting "Save as type: Device Driver Library" at the bottom of the Save As dialog.

The formats for station files and device driver libraries are compatible. You can save any device driver library as a station file, and you can save any station file as a device driver library, as long as it contains only device drivers. This means that there is no need for separate "New Station File" and "New Device Driver Library" menu items. The type of a new file is first determined when you save it.

HOW DEVICE DRIVERS WORK

Structure of an U.P.M.A.C.S. Device Driver

Each device in a serial port uses a device driver to specify the behaviour of the equipment. There is one device driver for each type of equipment, but devices with the same serial protocol share a device driver. Device drivers are not tied to individual serial ports; devices on different ports can share the same device driver.

Device Driver Parameters

Each device driver can have one or more parameters whose values are specified by the devices that use the driver. Most device drivers, for example, will ask the device to specify an address. A device driver for a klystron HPA might also ask the device to specify whether the equipment is equipped with a channel changer.

There are four types of parameters a device driver can have:

Bistate Parameters
Bistate parameters can have a value of ON or OFF. You can use bistate parameters to switch device driver features on or off.

Digital Parameters
Digital parameters have an integer number between 0 and 4,294,967,295 as their value. You can use digital parameters to specify a device address, or other numerical data. You could, for example, use digital parameters to specify the number of primaries and backups a smart switch can have.

Analog Parameters
Analog parameters can have almost any numerical value, including negative values and fractions. Analog parameters are rarely used in device drivers.

String Parameters
String parameters can have any arbitrary data as a value. You can use string parameters to specify the device address, if the address is a string ("A", "B", etc.) rather than a number.

Device Driver Objects

An U.P.M.A.C.S. device driver contains four types of objects.

Command Templates
Command templates are used to specify the command data and response, if any, for a group of commands that all have the same basic format. Command templates can have parameters just like device drivers, most typically the command opcode. The parameter values are specified by the commands that are based on the template. A command template can also ask the command to specify additional data to be included in the final string that is sent to the equipment.

Commands
Commands are based on command templates. Commands can also have parameters, whose values are specified in the polling sequence, or by an SCL program that sends the command. A set frequency command, for example, will ask you to specify the frequency to be set when the command is sent.

Data Objects
Data objects contain the information that is decoded from the equipment's responses. Each piece of information you want to access or display must be stored in a data object. Each data object has a source that specifies what response the value is to be taken from, and how. Data objects

are similar to the global registers, in fact, registers usually take their values from serial data objects.

SCL Programs
Device drivers can contain SCL programs. These programs can be used to decode data from a response, to calculate custom checksum values. Device driver programs can also be called by regular SCL programs. This allows you to provide pre-made code for complicated control functions, which can then be used by the actual control programs.

Error Handling

You can specify one or more types of error code that the device might return. You can also specify whether to resend the command, reinitialize the device, or do nothing when a specific error code is returned.

Initialization Sequence

You can specify an initialization sequence for the device. The initialization sequence of each device in a port is sent when the port is opened, and when the device times out. You can specify a sequence of commands to send when the port is opened, when the device times out, and commands that are sent in both cases.

You should only include those commands in the driver's initialization sequence that have to be sent for all devices that use the driver. Any additional initialization commands that depend on the individual device should be specified in the New Device dialog when the device is added to the serial port.

Overview Of Command Templates

A command template specifies the command data and response, if any, for a group of commands that have similar formats.

Many device drivers contain only a single command template that specifies the basic structure of all the commands. How many command templates you will need will depend on the serial protocol of the device. The more closely the commands for a device follow an established pattern, the fewer templates you will need.

The command template specifies the buffer size and timeout for commands that expect a response, or the delay before the next command for commands that don't. Not all the commands based on the template need to use the timeout and buffer size specified by the template, these settings can be overridden by the individual commands. There is no need to make a separate command template for a command merely because it needs a larger data buffer to hold the response data.

Command Template Parameters

Each command template can have one or more parameters whose values are specified by the commands that are based on it. Most command templates, for example, will ask the command to specify the command opcode.

There are four types of parameters a command template can have:

Bistate Parameters
Bistate parameters can have a value of ON or OFF.

Digital Parameters
Digital parameters have an integer number between 0 and 4,294,967,295 as their value.

Analog Parameters
Analog parameters can have almost any numerical value, including negative values and fractions.

String Parameters

String parameters can have any arbitrary data as a value. Most often, you would use a string parameter for the command opcode.

Command Elements

The command template specifies the command data to be sent. This is done by constructing a command string from a number of command elements. Each command element describes the format of a distinct section of the final command string. The command header, the device address, any data length bytes, the command opcode, the checksum, the terminating character: each of these will have its own command element. You can freely choose how many command elements you need, how they are encoded, and in which order they are to be sent.

There is a special type of command element called "command data." The exact content of a command data command element is specified not by the template, but the command itself. You must use a command element to allow the command to encode any of its parameters, since the command's parameters are not accessible to the command template. Other uses for command data command elements are, of course, thinkable. You can use a command data command element whenever you do not want to specify the exact format of a section of the command string in the template.

Each device in a serial port uses a device driver to specify the behaviour of the equipment. There is one device driver for each type of equipment, but devices with the same serial protocol share a device driver. Device drivers are not tied to individual serial ports; devices on different ports can share the same device driver.

Example:

A command template for the Scientific Atlanta's popular SABus protocol, for example, would contain the following six command elements:

1. The start character (STX, ASCII 02h)
2. Device address (Taken from a device driver parameter)
3. The command opcode (Taken from a template parameter)
4. The command data
5. The terminating character (ETX, ASCII 03h)
6. The checksum byte

Response Elements

The command template specifies the format of the response, if any. It can also specify the format of any error response the equipment might send. Just as the command data is specified using command elements, the response data is specified using response elements. Again, each distinct part of the response will be represented by a response element. The response header, the device address, any data length bytes, the command opcode, the checksum, the terminating character: each of these will have its own response element. You can freely choose how many response elements you need, how the data is decoded, and in which order they should appear in the response.

Most response elements are only used to make sure that the response is valid, and the decoded data is then discarded. A checksum response element, for example, will decode the checksum from the response and verify that it is correct. If the checksum is incorrect, the response is simply rejected. If the checksum (together all other response elements) is correct, the response is considered valid and accepted. The same goes for the device address and command opcode, if you specified them. The device address is decoded from the response packet, and then compared to the device address of the driver. If they are the same, the command is accepted, if they are not the same, it is rejected. You could just as well design a response template that accepts any de-

vice address; it would not affect the serial communications adversely. It would merely mean that, if a response has the wrong device address, it would be accepted anyway. In the same spirit, the command template could simply ignore the checksum rather than verify it. This would simply mean that responses with an incorrect checksum would be accepted as valid.

There are, however, three special kinds of response elements. A valid response may have one or more response data response elements. The data contained in a response data response element is not simply verified and then discarded, bit it is stored and can then be decoded by the device driver's data objects. A simple acknowledge response, whose only purpose is to inform the sender that a command has been executed correctly, would hence have no response data response elements. The response to a query for equipment parameters, however, contains actual valuable information, and must hence have at least one response data element. If you fail to include a response data response element in the template for a query, the response will only be checked for validity, and then simply discarded. There will be no way for you to decode and use the data returned with it. Any information that you intend to use in one of the device driver's data objects must be part of a response data element.

Error responses cannot contain response data response elements. Instead, they have error code response elements. The error code or codes you have specified in the device driver are decoded from the error code response elements, and appropriate action is taken. You don't have to decode the error codes if you don't want to: If your device driver has no error codes, all error messages are simply ignored.

Both valid and error responses can also contain value response elements. Value response elements are used to extract either numerical values (integers between 0 and 4,294,967,295) or strings from the response, to be used in one of three ways:

> ➤ as a data length value for a section of the response
> ➤ as a repeat count for another response element
> ➤ to evaluate response element conditions (see below)

Example:

The valid response for equipment that uses the SABus protocol, for example, would have the following six response elements:

1. The start character (ACK, ASCII 06h)
2. Device address (Checked against the corresponding device driver parameter)
3. The command opcode (Checked against the corresponding template parameter)
4. The response data
5. The terminating character (ETX, ASCII 03h)
6. The checksum byte

The error response would be the same, except that the start character is a NAK (ASCII 15h), and there would be one or more error code response elements instead of the response data.

Response Element Conditions

Each response element can have a response element condition attached to it. This simply means that U.P.M.A.C.S. will only look for a response element that has a condition, if the condition is true. The condition is always based on a value extracted by a value response element. You can check if numerical values are less than, greater than, or equal to a certain value, or you can check if a string value is equal to, or contains a certain string.

Response element conditions are an advanced feature, and are rarely used. Use response element conditions if the response may or may not contain a certain block of data, depending on some flag or value in the response.

Default Error Codes

If you specified error codes for the device driver, but the error response to commands based on this template does not contain all of the error codes explicitly, you can specify default values for those error codes. The error code will simply be assumed to be the default value if an error response is received, no matter what that response contains.

Overview Of Commands

Commands are based on command templates. Each command specifies the values of the template parameters, as well as the data contained in any command data command elements the template may have.

The command may also override the buffer size and timeout values for commands that expect a response.

Command Parameters

Each command can have one or more parameters whose values are specified in the polling sequence, or by an SCL program that sends the command. Command parameters are used in two ways:

Some parameters are used to specify different values for a command. A Set frequency command, for example, will have a parameter called "Frequency," whose value specifies the frequency that will be set. The command is treated as the same command no matter which frequency is specified: Any data objects that depend on the command will be updated regardless of the frequency, and if you disable the command, it will be disabled for all frequencies.

Some parameters, however, are used to create different commands with the same functionality. A Query switch position command for a unit that controls eight waveguide switches, for example, might have a parameter that specifies which switch to query. In this case, the eight different variations of the command should be treated as if they were eight separate commands: If I disable switch number 1, I should still be able to query switch 2, and the position of switch 2 must only be updated if I actually query switch 2, not if I query switch 3 or 4.

The sources of data object, and the ENABLECMD and DISABLECMD SCL commands, therefore expect you to specify a switch number for the Query switch position command, but not a frequency for a Set frequency command. The set frequency command is one single command regardless of the frequency, the Query switch position command, however, is eight different commands in one, distinguished by the switch number.

When you add a parameter to a command, you have to specify whether it is used to distinguish between different commands or not.

There are four types of parameters a command can have:

Bistate Parameters
Bistate parameters can have a value of ON or OFF. You can use a bistate parameter to create a command to switch the equipment between two different states, e.g. local and remote.

Digital Parameters
Digital parameters have an integer number between 0 and 4,294,967,295 as their value. You can use a digital parameter to create a command to set a channel, to switch a backup to a particular primary, or to switch between more than two states (e.g. local / remote / remote front panel mode). You can also use digital parameters to create commands to query the status and settings for a particular primary or backup channel of a smart switch.

Analog Parameters
Analog parameters can have almost any numerical value, including negative values and fractions. Use analog parameters to set frequencies, attenuations, etc.

String Parameters
String parameters can have any arbitrary data as a value. Use string parameters to set satellite or channel names, to upload binary data of any sort to the equipment, or to channel arbitrary data through a routing device.

Command Data

The command specifies the content of each of the template's command data command elements. The data string for the command data command elements is itself made up of the same kinds of command elements as the template uses. Only command data command elements are not used by commands.

Overview Of Data Objects

Data objects contain the information extracted from serial responses, as well as any other settings and parameters of the device. A data object may have one of four types:

- Bistate Objects
Bistate objects can have a value of ON or OFF. Use bistate objects to store alarm information, and to store settings that have two possible states (e.g. local / remote).

- Digital Objects
Digital objects have an integer number between 0 and 4,294,967,295 as their value. Use digital objects to hold settings that are integer numbers, like channel selections, or settings that have more than two possible states (e.g. local / remote / remote front panel mode). If you use a digital object for settings with more than two states, each possible state will be represented as a number.

- Analog Objects
Analog objects can have almost any numerical value, including negative values and fractions. Use analog objects to hold frequencies, attenuations, meter readings, antenna positions, and similar data. Analog objects can also have values that include greater than / less than information. Some equipment does not return values outside of certain limits; a modem, for example, might return a value of "<1.0E-20" for an Eb/N0 value. Analog objects can store this type of value.

Analog objects can also contain more than one value. The number of values an analog object has is termed its size. You can use an analog object with a size of more than one value to store such data as the trace of a spectrum analyzer. The values of an analog object with more than one value can be displayed as a bar or line graph by creating an analog register based on the object.

- String Objects
String objects can have any arbitrary data as a value. You can use a string object to hold satellite names, channel designations, or any other data that you cannot conveniently store in any of the other types of objects.

A data object does not necessarily contain a value. If the value of a data object hasn't been updated yet (e.g. because the command it gets its data from hasn't been sent yet), or if an error occurred trying to update a value (e.g. because its command timed out), the data object does not have a value, and is said to be in the error state.

A data object can also be masked. A data object will be automatically masked if the command or object(s) it depends on are disabled or masked. You can also mask a data object from within an

SCL program using the MASKDRVOBJ command. The value of a masked object is not accessible, and any registers that depend on it will be auto masked.

Sources

Each data object has a source that determines where it gets its value from. There are many different types of sources for each type of data object.

Most objects get their value from a response to a query command, but some objects might get their value from one or more other data objects. A bistate object, for example, can bet set to its ON value if the value of a specific analog object goes beyond certain limits. Other data objects don't get their values from anywhere at all, they have to be set using SCL programs.

Object Parameters

A data object can have one or more parameters whose values are specified by the objects and registers that depend on them.

Data object parameters are used to create groups of objects with related values. An object that has parameters is not, in fact, a single data object with a single value, but a whole set of different objects, each with its own value.

If you were writing a device driver for a unit that controls eight waveguide switches, for example, you could create a different data object for each of the switch positions. Your driver would then contain eight data objects, called "Switch position 1", "Switch position 2", "Switch position 3", etc. all the way up to "Switch position 8".

Alternatively, you could create a single data object called "Switch position" which has a digital parameter to specify the switch number. That data object would not, in fact, be a single data object, but eight objects in one, with eight different values. Whenever you wanted to access the value of the object, e.g. in a register source, you would have to specify which of the eight objects you are interested in by specifying a value for the switch number parameter.

So, a data object that has no parameters is a single object with a single value. An object that has parameters, however is really a group of objects, each with its own value.

There are four types of parameters an object can have:

Bistate Parameters
Bistate parameters can have a value of ON or OFF.

Digital Parameters
Digital parameters have an integer number between 0 and 4,294,967,295 as their value.

Analog Parameters
Analog parameters can have almost any numerical value, including negative values and fractions.

String Parameters
String parameters can have any arbitrary data as a value.

Usually, only digital parameters are used for data objects.

Parameter Value Inheritance

If a data object has a parameter of the same tag and type as a command or other object(s) it depends on, it can pass the value of that parameter on to the command or other object. It is then said to inherit the parameter from the command or object(s) it depends on.

Let us say, for example, that your driver has a "Switch position" data object that has a digital parameter called "Switch number". If this object gets its value from a "Query switch position" command that also has digital parameter called "Switch number", the same switch number will be used by both. This means that the "Switch position" object with a switch number of 1 will automatically get its value from the "Query switch position" command with a switch number of 1; the

"Switch position" object for switch number 2 will get its value from the "Query switch position" command for switch number 2, etc.. The object will always use the command with the same switch number as itself.

Parameters are inherited if:

1. The data object and the command have a parameter with the same tag and type

2. The command parameter is used to distinguish between commands (see *Command Parameters* on page 8 for details)

3. The data object does not specify a value for the command parameter

If the data object depends on other data objects rather than on a command, it will inherit parameters from the other object or objects in the same way. In this case, of course, condition 2. above is irrelevant, since object parameters are always used to distinguish between objects.

Overview Of SCL Programs

You can include SCL programs in device drivers. For more details on what SCL programs are, see *SCL programs* in the *Developer's Manual*.

SCL programs are used to evaluate data in processor and summary sources, and to calculate checksums. Programs written to be used in processor and summary sources or for checksums must follow certain guidelines. See *Programs for Sources, Checksums, and SABus Response Data* in the *SCL Language Reference* for details.

You can also provide SCL programs to be called by SCL programs from outside the device driver. This allows you to provide ready-made SCL programs for any tasks related to the equipment.

Device driver programs are edited in program editor windows the same way regular SCL programs are. The program editor windows for device driver programs do not appear in the main Development System window, however. Rather, they are grouped together in a separate top-level window.

DEVICE DRIVER OBJECTS

Device Drivers

You can create, edit, and duplicate device drivers like any other database objects. See *Viewing and Editing Objects* in the *Developer's Manual* for details on how to create and edit device drivers.

The New Device Driver Dialog
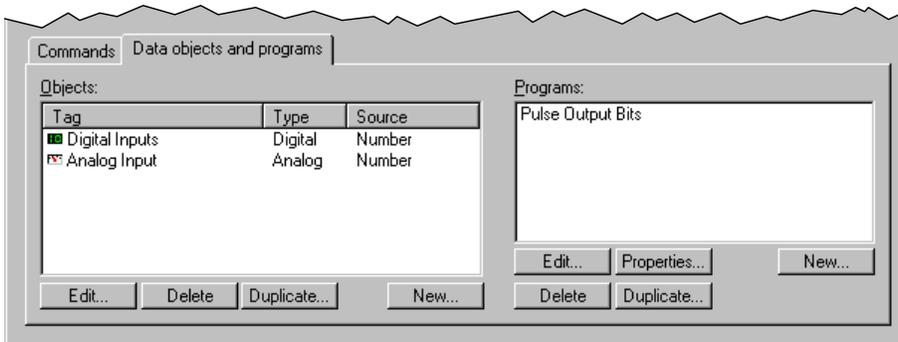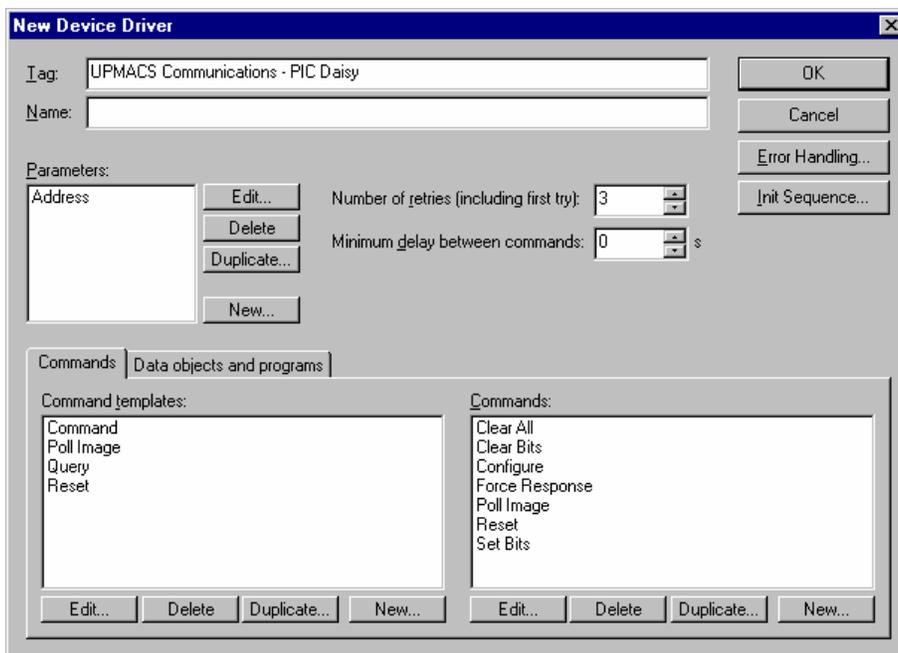
- Tag:
Enter the tag by which the driver is identified. Each driver must have a unique tag.

- Name:
Enter the name of the device driver. Leave this field blank if you want to use the tag as name.

▪ Parameters:
Shows a list of all the device driver's parameters. Use the buttons to edit, delete, duplicate, and create parameters. You can grab the parameters with the mouse and drag them to a new position in the list to change their order.

See *Parameters* on page 14 for a description of the New Parameter dialog.

▪ Number of retries (including first try):
Enter the number of times a command will be resent if there is a timeout. If you specify 1 in this field, commands will not be resent if there is a timeout. If you specify 3, U.P.M.A.C.S. will try to send each command up to three times before logging a timeout.

Not all commands in the driver have to use the number of retries you specify here. You can over-ride the default number of retries in the individual commands.

▪ Minimum delay between commands:
Some poorly designed equipment will loose data or lock up if a second command is sent immedi-ately after it responded to the first. If your equipment needs a delay between its response and the next command, enter that delay here, in seconds. You can enter fractions of a second.

This delay value is applied to all commands sent to the port a device is on, regardless of the driver the commands belong to. The delay between commands on a port will be the largest of all the minimum delays specified in the device drivers used by the port's devices.

▪ Commands / Data objects and programs:
Use these tabs to select whether the list of command templates and commands or the list of data objects and SCL programs is shown.

▪ Command templates:
Shows a list of all the command templates you have defined for the device driver. Use the buttons to edit, delete, duplicate, and create templates. See *Command Templates* on page 15 for a de-scription of the New Command Template dialog. If the device driver dialog shows data objects and SCL programs instead of command templates and commands, click on the "Commands" tab.

▪ Commands:
Shows a list of all the commands you have defined for the device driver. Use the buttons to edit, delete, duplicate, and create commands. See *Commands* on page 17 for a description of the New Command dialog. If the device driver dialog shows data objects and SCL programs instead of command templates and commands, click on the "Commands" tab.

▪ Data objects:
Shows a list of all the data objects you have defined for the device driver. The list shows the name, the type, and the source of the object. You can resize the three columns by dragging the edges of the column headers with the mouse. Click on the "Name" header to sort the objects by name, click on the "Type" header to sort them by type, and on the "Source" header to sort by the source.

The list also shows little icons next to the objects that tell you what type they are:

    ▮ Bistate Object
    ▮ Digital Object
    ▮ Analog Object
    ▮ String Object

Use the buttons to edit, delete, duplicate, and create. See *Data Objects* on page 18 for a descrip-tion of the New Data Object dialog. If the device driver dialog shows command templates and

commands instead of data objects and SCL programs, click on the "Data objects and programs" tab.

- Programs:

Shows a list of all the SCL programs you have defined for the device driver. Use the buttons to edit, delete, duplicate, and create commands. If you press the "Edit" button, an SCL program editor will appear to allow you to edit the program code. To edit the properties of the program, press the "Properties..." button.

See *SCL Programs* on page 19 for a description of the New Program dialog. If the device driver dialog shows command templates and commands instead of data objects and SCL programs, click on the "Data objects and programs" tab.

- Error Handling:

Press this button to edit error codes and error handling for the driver. See *Error Handling* on page 20 for a description of the Error Handling dialog.

- Init Sequence:

Press this button to edit the driver's initialization sequence. See *Initialization Sequence* on page 22 for a description of the Initialization Sequence dialog.

Parameters

Parameters include device driver parameters, command template parameters, command parameters, and data object parameters.

The New Parameter Dialog

- Tag:
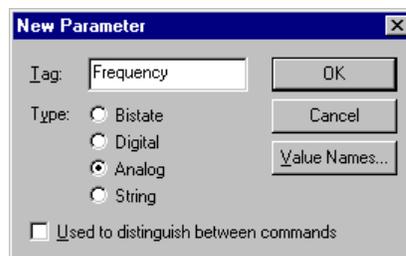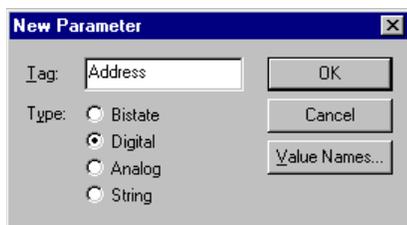
Enter the tag by which the parameter is identified. Each parameter in an object must have a unique tag.

- Type:

Select the parameter's type.

- Value names:

Press this button to assign names to the values of a digital parameter. The value of a digital parameter that has value names is specified by selecting a value from a list of all the value names, rather than entering a number. You should use value names if a digital parameter represents a number of different settings or choices (e.g. local / remote / remote front panel mode), rather than an actual number.

- Used to distinguish between commands:

This check box appears only in dialogs for command parameters. Check this box if you want to use the parameter to distinguish between commands. See *Command Parameters* on page 8 for details.

Command Templates

A command template is a pattern on which one or more commands are based. The command template describes the format of the command data, as well as the format of any response the equipment might send. The template also provides default values for the timeout and the size of the response buffer.

Some equipment sends an error response if a command was badly formatted or could not be executed for some other reason. You can specify the format of a regular (valid) response, and of an error response. If an error response is received, any error codes are extracted from it, and the actions you specified in the error handling of the device driver are taken. See *Error Handling* on page 20 for more details.

If you specify both an error and a valid response, make sure you provide enough information to tell the two apart. If a response received from the equipment matches the format you specified for a valid response and the format you specified for an error response, it is assumed to be an error response. It is hence important that valid responses returned by the equipment do not match the error response you specified.

The New Command Template Dialog
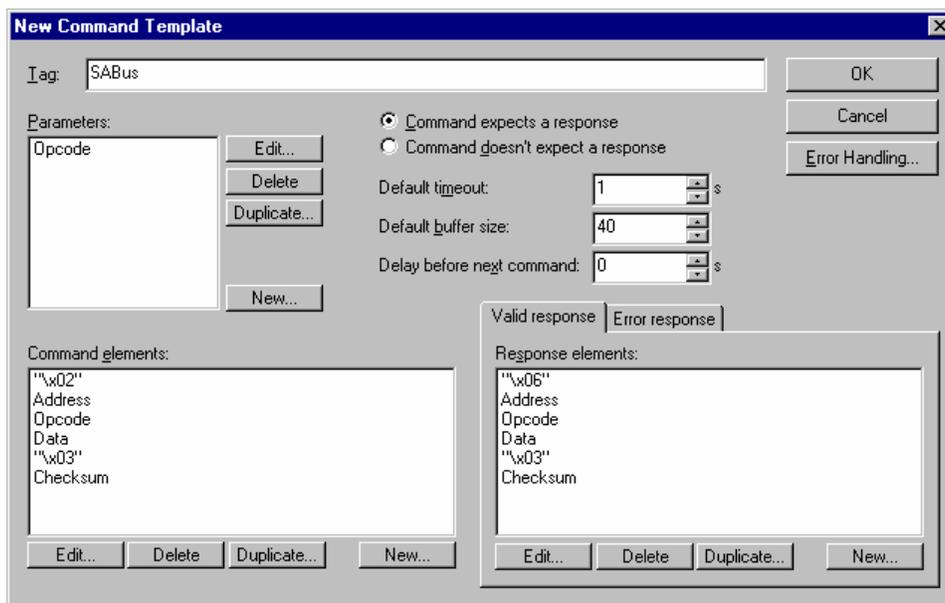
- Tag:
Enter the tag by which the template is identified. Each command template in a device driver must have a unique tag.

- Parameters:
Shows a list of all the template's parameters. Use the buttons to edit, delete, duplicate, and create parameters. You can grab the parameters with the mouse and drag them to a new position in the list to change their order.

See *Parameters* on page 14 for a description of the New Parameter dialog.

- Command expects / doesn't expect a response:
Select whether the equipment sends a response to the commands based on this template. Do not select "Command doesn't expect a response" if the equipment sends any data, even if you do not plan on using it.

▪ Default timeout:
Enter the maximum time U.P.M.A.C.S. will wait for the equipment to return a response. If no valid response has been received after the number of seconds specified, the command is resent, or a timeout is reported. You can enter fractions of a second.

Not all commands based on the template have to use the timeout value you specify here. You can override the default timeout in the individual commands.

This field only applies to commands that expect a response.

▪ Default buffer size:
Enter the size of the data buffer here. The data buffer must be large enough to hold all of the data the equipment sends. If the response has a fixed length, enter that length here. If it has a variable length, enter a buffer size large enough to hold the largest possible response. If a response is too long to fit in the buffer, U.P.M.A.C.S. will not recognize it as valid, even if it is.

Not all commands based on the template have to use the buffer size you specify here. You can override the default buffer size in the individual commands.

This field only applies to commands that expect a response.

▪ Delay before next command:
Enter the number of seconds to wait before sending the next command after commands based on this template. You can enter fractions of a second. This field only applies to commands that do not expect a response.

▪ Command elements:
Shows a list of all the command elements in the order they will be sent. Use the buttons to edit, delete, duplicate, and create elements. You can grab the elements with the mouse and drag them to a new position in the list to change their order.

See *Commands Elements* on page 25 for a description of the different types of command elements.

▪ Valid response / Error response:
Use these tabs to select whether the list of response elements for a valid response or an error response is shown.

This selection only applies to commands that expect a response.

▪ Response elements:
Shows a list of all the response elements in the order they should appear in the response. Use the buttons to edit, delete, duplicate, and create elements. You can grab the elements with the mouse and drag them to a new position in the list to change their order.

A comand template has two sets of response elements: one describes a valid response, the other describes an error response. Which of the response elements are shown depends on the selection of the Valid response / Error response tab described above.

See *Responses Elements* on page 41 for a description of the different types of response elements.

This field only applies to commands that expect a response.

▪ Error Handling:
Click this button to specify default values for the device driver's error codes. You can specify values to be used for error codes that do not appear in the error response of the equipment. The error codes you specify will only be used if an error response is received.

This field only applies to commands that have an error response, and whose device driver has at least one error code. See *Error Handling* on page 20 for details on error codes.

Commands

Commands are sent to equipment to query data or change settings. Each command is based on a command template, which describes the format of the command data and of any response. The command merely specifies the content of any command data command elements and the values of any parameters the template may have.

The New Command Dialog

- **Tag:**
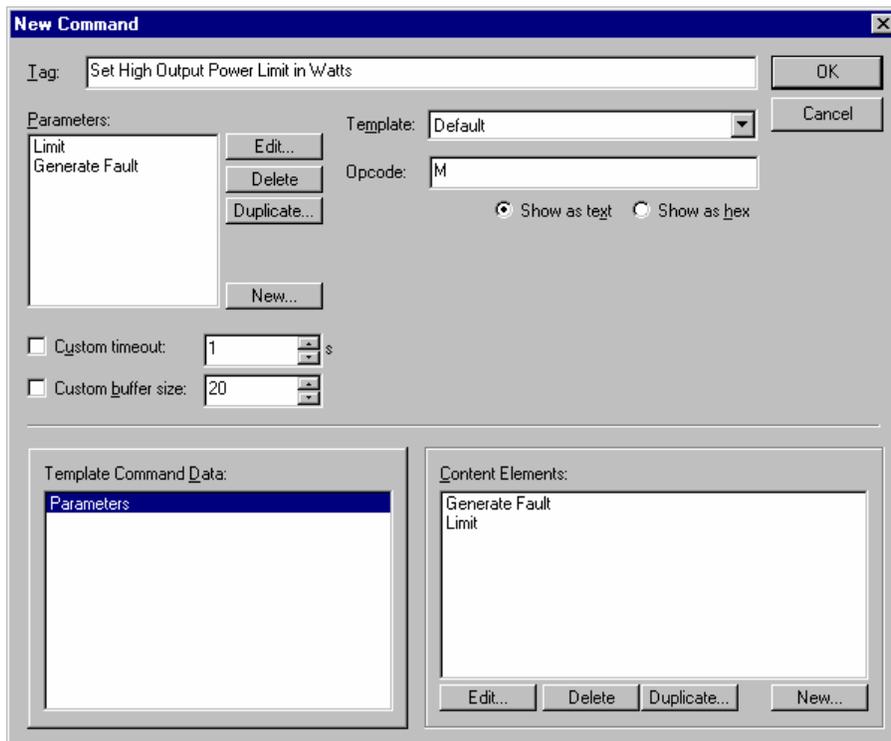
Enter the tag by which the command is identified. Each command in a device driver must have a unique tag.

- **Parameters:**

Shows a list of all the command's parameters. Use the buttons to edit, delete, duplicate, and create parameters. You can grab the parameters with the mouse and drag them to a new position in the list to change their order.

See *Parameters* on page 14 for a description of the New Parameter dialog.

- **Template:**

Select the template upon which the command is to be based. Specify values for all the template parameters immediately below. In the sample dialog, the command uses a template that has one string parameter called "Opcode".

If you do not specify a value for a particular template parameter (i.e., leave the field blank), a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Show as decimal / Show as hex:
If the template has any parameters of type digital that don't have value names, you can select the way you want to enter the parameter values here. Select "Show as decimal" to enter the values in decimal, select "Show as hex" to enter the values in hexadecimal.

Show as text / Show as hex:
If the template has any parameters of type string, you can select the way you want to enter the parameter values here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

▪ Custom timeout, Custom buffer size:
If your command needs a different timeout or buffer size than the defaults specified in the template, check the appropriate box, and enter the value to be used instead. These fields only apply to commands that expect a response.

▪ Custom retries:
If your command needs a different number of retries than the default specified in the driver, check the appropriate box, and enter the new number of retries. This field only applies to commands that expect a response.

▪ Template command data:
Select the command data command element of the command's template whose content you wish to display on the right. This field only applies to commands that expect a response.

▪ Content elements:
Shows a list of all the command elements that are used to make up the string of the command data element selected on the left. The elements appear in the order that they will be sent. Use the buttons to edit, delete, duplicate, and create elements. You can grab the elements with the mouse and drag them to a new position in the list to change their order.

See *Commands Elements* on page 25 for a description of the different types of command elements.

This field only applies to commands whose templates have at least one command data command element.

Data Objects

Data objects contain the information extracted from serial responses, as well as any other settings and parameters of the device. All data required by SCL programs, registers, or other data objects must be contained in data objects.

The New Data Object Dialog

▪ Tag:
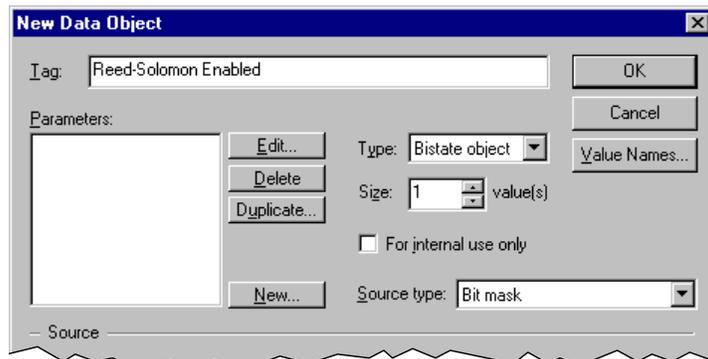
Enter the tag by which the object is identified. Each data object in a device driver must have a unique tag.

▪ Parameters:

Shows a list of all the command's parameters. Use the buttons to edit, delete, duplicate, and create parameters. You can grab the parameters with the mouse and drag them to a new position in the list to change their order.

See *Parameters* on page 14 for a description of the New Parameter dialog.

▪ Type:

Select the data type of the object.

▪ Size:

Select the number of values an analog object has. Use analog objects with more than one value to hold such data as the trace of a spectrum analyzer. This field is only available if you selected "Analog object" as the type.

▪ For internal use only:

Check this box to prevent SCL programs and registers from outside the driver to access this data object. If you check this box, only data objects and SCL programs that belong to this driver can access the data object.

▪ Source type:

Select the source type of the object. See *Data Object Sources* on page 61 for details on object sources.

▪ Value names:

Press this button to assign names to the values of a digital object. The value names can be used by indicators that represent this object's value and by SCL programs. You should use value names if a digital data object represents a number of different settings or choices (e.g. local / remote / remote front panel mode), rather than an actual number.

SCL Programs

SCL programs are used in processor and summary sources, and by checksum command and response elements. They can also be called by other SCL programs, both from within the driver and from outside. For a description of the SCL language, see the *SCL Language Reference*.

Once you have created an SCL program, an SCL Program Editor window will appear to let you edit the code.

If you select a program in the Programs list, and press the "Edit..." button, the editor window will appear rather than the Edit Program dialog. To change the properties of a program, select it and press the "Properties..." button. You can also select "Properties..." from the "Edit" menu when viewing the program's code.

The New Program Dialog

▪ Tag:
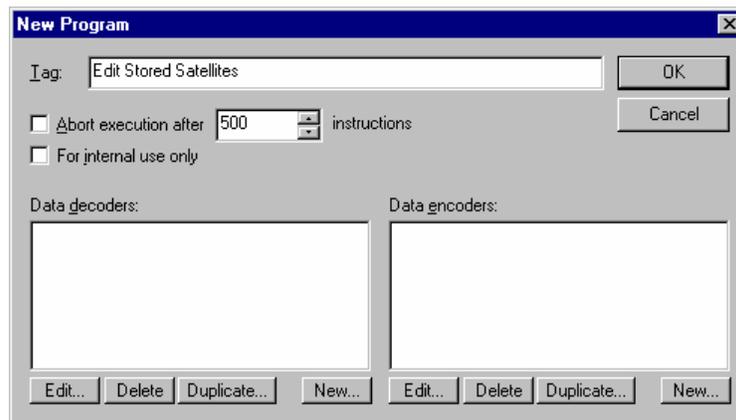
Enter the tag by which the program is identified. Each program in a device driver must have a unique tag.

▪ Abort execution after ... instructions:

Check this box if you want to guard the program against endless loops. The program will be aborted after the specified number of SCL commands or assignments have been executed. Programs used in processor and summary sources have a built-in instruction limit of 500 instructions. Check this box to specify a greater or smaller limit.

▪ For internal use only:

Check this box to prevent SCL programs from outside the driver to call this program. If you check this box, only SCL programs that belong to this driver can call the program.

▪ Data decoders:

Shows a list of all data decoders defined for this program. Use the buttons, to add, delete, and edit decoders.

See *SCL Data Decoders and Encoders* in the *Developer's Manual* for details.

▪ Data encoders:

Shows a list of all data encoders defined for this program. Use the buttons, to add, delete, and edit encoders.

See *SCL Data Decoders and Encoders* in the *Developer's Manual* for details.

Error Handling

You can specify a number of different types of error codes that the equipment may return. You can use the error code to decide what action to take when an error occurs. The error codes can also be retrieved from within an SCL program that sent a serial command.

There are two types of error codes. Numerical error codes can have a value between 0 and 4,294,967,295. String error codes can contain any arbitrary data. Each error code represents a type of error the equipment returns. Do not create an error code for each possible error that can occur. Most equipment will only require a single error code, which will take on different values depending on which error occurred.

Most equipment will only return one type of error code, usually a number or one or two letters. Some equipment, however, returns error messages along with the error code, or it may return

secondary or sub-error codes. You will need to specify one error code object for each type of error code returned. For example, if your equipment returns an error code and an error message in the error response, you should create two error codes, one called "Error Code", and one called "Error Message". If the equipment returns a main error code, and a secondary error code for some or all errors, you should create two error codes, one called "Main Error Code", and one called "Secondary Error Code". If your equipment returns only a simple error code in the error response, you should create only one error code called "Error Code". The names given in these examples are, of course, only suggestions. You can name your error codes anything you like.

There are three types of actions that U.P.M.A.C.S. can take upon receiving an error response:

> ➢ Ignore the error
> ➢ Stop the polling sequence and reinitialize the device
> ➢ Retry sending the command

You can specify which action is to be taken when receiving what error.

The Error Handling Dialog
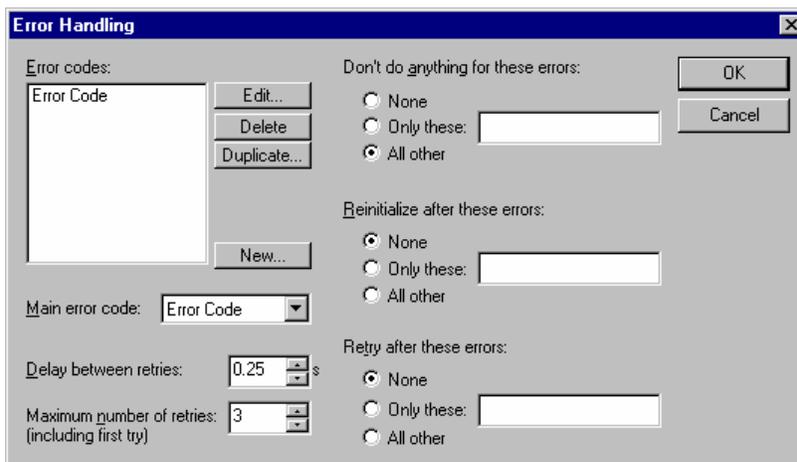
- **Error codes:**
Shows a list of all error codes. Use the buttons to edit, delete, duplicate, and create error codes.

Each error code represents a type of error the equipment returns. Do not create an error code for each possible error that can occur. Most equipment will only require a single error code, which will take on different values depending on which error occurred.

- **Main error code:**
Select the error code whose value is used to decide what action to take. If you do not specify a main error code, all errors will be treated the same.

- **Delay between retries:**
Enter the number of seconds to wait before sending the command again if it has to be resent. You can enter fractions of a second.

- **Maximum number of retries (including first try):**
Enter the number of times a command will be resent if need be. If you specify 3, U.P.M.A.C.S. will try to send each command up to three times before giving up.

▪ Don't do anything for / Reinitialize after / Retry after these errors:
Select for which error codes to take each of the three actions.

None:
Select this option to never perform this action, regardless of the error returned.

Only these:
Select this option if you would like tp specify specific errors for which the action should be performed. If the main error code you selected is a numerical error code, enter the desired values separated by commas. If the main error is a string error code, enter a regular expression that matches all the desired error codes.

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

All other:
Select this option to perform this action for all errors for which you didn't specify anything else. You must select this option for one and only one of the actions.

Initialization Sequence

You can specify a default initialization sequence for the device. The initialization sequence consists of a number of commands that are sent to initialize the equipment.

There are three types of commands in the initialization sequence:

➢ First-time initialization commands that are sent when the port is first opened
➢ Reinitialization commands that are sent when the device needs to be reinitialized after a timeout or error
➢ Common initialization commands that are sent in both cases

You should only include those commands in the initialization sequence that have to be sent for all devices. You can specify additional commands when you add a device that uses the driver to a serial port. Each device also has the option to ignore the default initialization sequence.

When the port is first opened, the initialization commands are sent in the following order:

1. First-time initialization commands specified in the driver, in the order in which they appear in the list
2. Common initialization commands specified in the driver, in the order in which they appear in the list
3. First-time initialization commands specified in the device, in the order in which they appear in the device's initialization sequence
4. Common initialization commands specified in the device, in the order in which they appear in the device's initialization sequence

When a timeout occurs, or when an error occurs that requires reinitialization (see *Error Handling* on page 20 for details), the initialization commands are sent in the following order:

1. Reinitialization commands specified in the driver, in the order in which they appear in the list

2. Common initialization commands specified in the driver, in the order in which they appear in the list

3. Reinitialization commands specified in the device, in the order in which they appear in the device's initialization sequence

4. Common initialization commands specified in the device, in the order in which they appear in the device's initialization sequence

The Initialization Sequence Dialog
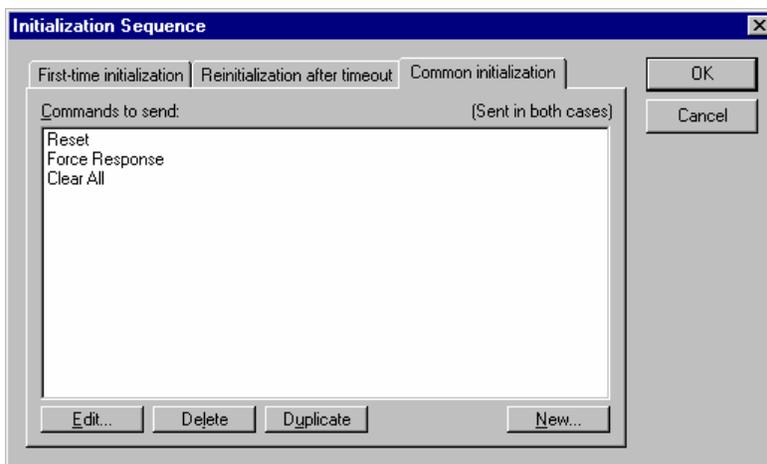
▪ **First-time initialization / Reinitialization after timeout / Common initialization:**
Select which commands should be displayed in the list. These tabs only affect which commands are displayed, not which type of initialization the driver supports. All drivers always contain all three types of initialization sequence (though some of them may be empty).

▪ **Commands to send:**
Shows a list of all the commands in the initialization sequence in the order in which they will be sent. Use the tabs to select which initialization sequence is shown in the list.

Use the buttons to edit, delete, duplicate, and add new commands. You can grab the commands with the mouse and drag them to a new position in the list to change their order.

The New Initialization Command Dialog
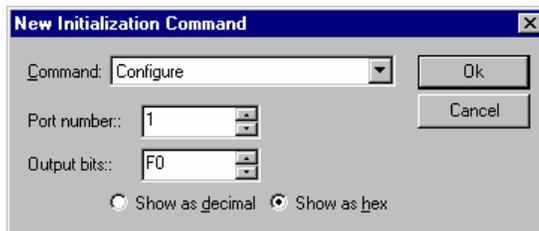
▪ **Command:**
Select the command to send. Specify values for all the command parameters immediately below. In the sample dialog, the command has two digital parameter called "Port Number" and "Output Bits".

If you do not specify a value for a particular command parameter (i.e., leave the field blank), a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Show as decimal / Show as hex:
If the command has any parameters of type digital that don't have value names, you can select the way you want to enter the parameter values here. Select "Show as decimal" to enter the values in decimal, select "Show as hex" to enter the values in hexadecimal.

Show as text / Show as hex:
If the command has any parameters of type string, you can select the way you want to enter the parameter values here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

COMMANDS ELEMENTS

Overview

Command elements are used in command templates and commands to construct the data string that will be sent to the equipment. Each command element represents one section of the command string. The sections are then concatenated to form the final command string.

You create and edit command elements from within the New Command Template and New Command dialogs. When you create a new command element, you will be asked to select its type. There are nine types of command elements.

General types of command elements:

- ➢ Fixed string
- ➢ Delay

Elements that encode the value of one or more parameters:

- ➢ Bistate parameter
- ➢ Digital parameter (number)
- ➢ Digital parameter (strings)
- ➢ Analog parameter
- ➢ String parameter
- ➢ Bistate parameter bits

Special types of command elements:

- ➢ Data length
- ➢ Checksum
- ➢ Command data*

* Command data command elements are only used by command templates, not by commands

Fixed String Commands Elements

Fixed string command elements are used for the parts of the command that never change, e.g. start and stop bytes.

The New Fixed String Element Dialog

▪ String:

Enter the data to be sent here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details.

▪ Show as text / Show as hex:

Select the way you want to enter the data string. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

Delay Commands Elements

Delay command elements don't actually encode any data, but cause U.P.M.A.C.S. to wait between sending the data of the previous and the next command element.

The New Delay Element Dialog
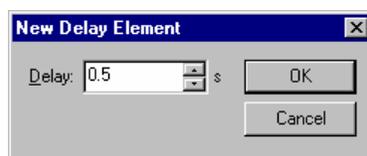
▪ Delay:

Enter the delay time to wait. You can enter fractions of a second.

Bistate Parameter Commands Elements

Bistate parameter command elements encode the value of a bistate parameter. Command elements for command templates can encode device driver or template parameters; command elements for commands can encode device driver or command parameters.

The New Bistate Parameter Element Dialog
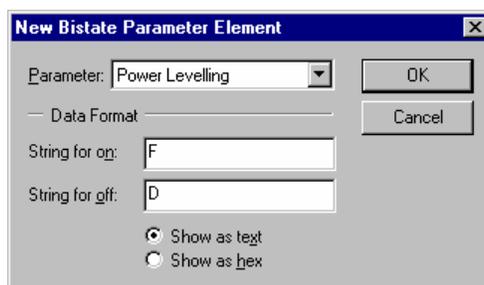
▪ Parameter:

Select the parameter whose value you want to encode. Device driver parameters and command template or command parameters are shown in the same list.

▪ String for on:

Enter the data to be sent if the parameter is ON. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

- String for off:

Enter the data to be sent if the parameter is OFF. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

- Show as text / Show as hex:

Select the way you want to enter the data strings. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details.

Digital Parameter (Number) Command Elements

Digital parameter (number) command elements encode the value of a digital parameter as a number. Command elements for command templates can encode device driver or template parameters; command elements for commands can encode device driver or command parameters.

The New Digital Parameter Element Dialog

- Parameter:

Select the parameter whose value you want to encode. Device driver parameters and command template or command parameters are shown in the same list.

- Encoding:

Select the encoding method for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:

A single byte (character) in the command string is used to represent the parameter as an 8-bit value. Requires a fixed width of 1.

Multibyte:

Two or more bytes (characters) in the command string are used to represent the parameter as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) will be sent first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) will be sent first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The parameter is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The parameter is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding writes the parameter out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), it does not encode it using the individual bits of each byte.

▪ Use fixed width:
Check this check box if you want the resulting string to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Pad with zeros / Pad with spaces / Custom padding character:
Specify the character that is to be used to pad the parameter to the fixed width if it is too short. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Align left / Align right:
Select whether the number should be left or right aligned within the fixed width. If the number is left aligned, any padding characters will be added to the end the number; if it is right aligned, they will be added to the beginning. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ Use capital letters A–F / small letters a–f as hex digits:
Select whether capital or small letters should be used for hex digits. This setting only applies to the hexadecimal encoding.

Digital Parameter (Strings) Command Elements

Digital parameter (strings) command elements encode the value of a digital parameter. You explicitly specify the data string to be used for the different values. Command elements for command templates can encode device driver or template parameters; command elements for commands can encode device driver or command parameters.

The New Digital Parameter Element Dialog
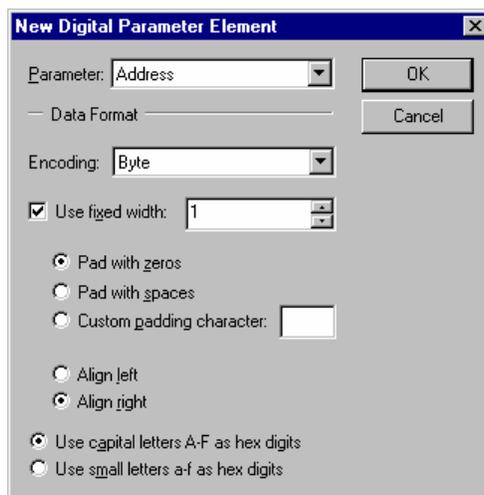
- Parameter:

Select the parameter whose value you want to encode. Device driver parameters and command template or command parameters are shown in the same list.

- Strings to send for values:

Shows a list of all values that you specified data strings for, together with their strings. Use the buttons to edit, delete, duplicate, and add values.

- String for other values:

Enter the data to be sent for all values that do not appear in the list above. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

- Show as text / Show as hex:

Select the way you want to enter the data strings. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details.

Analog Parameter Commands Elements

Analog parameter command elements encode the value of an analog parameter. Command elements for command templates can encode device driver or template parameters; command elements for commands can encode device driver or command parameters.

You can provide a factor and an offset to be applied to the parameter before formatting it. The number written to the data string is calculated from the parameter as follows:

number = parameter · factor + offset

To use the parameter unaltered, specify a factor of 1 and an offset of 0.

The New Analog Parameter Element Dialog

- ▪ Parameter:

Select the parameter whose value you want to encode. Device driver parameters and command template or command parameters are shown in the same list.

- ▪ Factor:

Enter the factor with which the parameter is to be multiplied before writing it to the data string. The factor is applied before the offset.

- ▪ Offset:

Enter the offset that is to be added to the parameter before writing it to the data string. The offset is applied after the factor.

- ▪ Encoding:

Select the encoding method for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the parameter as an 8-bit value. If you select signed byte encoding, the most significant bit of the byte will be used to denote the sign in the standard way (-1 is $FF, -2 is $FE, etc.). If you select unsigned byte encoding, the sign of negative numbers will be ignored, i.e. -20 will be encoded the same as 20. The byte encoding method requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the parameter as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) will be sent first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) will be sent first. If you select signed mul-

tibyte encoding, the most significant bit of the most significant byte will be used to denote the sign in the standard way (for 16-bit values -1 is $FFFF, -2 is $FFFE, etc.). If you select unsigned multibyte encoding, the sign of negative numbers will be ignored, i.e. -20 will be encoded the same as 20. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The parameter is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The parameter is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding writes the parameter out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), it does not encode it using the individual bits of each byte.

- Show positive (+) sign:
Check this check box if you want a plus sign to be shown for positive values. The plus sign does not actually have to be the "+" character. You can specify any character you want.

This setting only applies to the decimal encoding.

- Show negative (–) sign:
Check this check box if you want a minus sign to be shown for negative values. The minus sign does not actually have to be the "-" character. You can specify any character you want.

This setting only applies to the decimal encoding.

- Use thousands separator:
Check this check box if you want digits to be separated into groups of three using a thousands separators (usually a comma). If you check this box, you must also specify the actual character used to group digits.

This setting only applies to the decimal encoding.

- Number of decimals:
For the decimal encoding, this field allows you to specify the number of digits shown after the decimal point. If you do not want decimal digits or a decimal point, enter 0 here.

For all other encodings, this field allows you to specify the number of implied decimals. Implied decimals are a way of encoding fractions without using a decimal point by multiplying it with a power of ten before encoding it. A number that is encoded using 3 implied decimals, for example, will be multiplied by 1000 before it is encoded. This will move the three decimals from the right to the left of the decimal point, and the decimal point will no longer be needed. 21.304, for example, will simply be encoded as 21304.

You can use implied decimals with the decimal encoding method as well. Simply set the required number of decimals in the "Number of decimals" field, and uncheck the "Show decimal marker" check box described below. No decimal point will then be shown.

Please note that the implied decimals for the hexadecimal, octal, and binary encoding methods are decimal fractions, not hexadecimal, octal or binary fractions. In hexadecimal encoding with three implied decimals, 43.249 (which is ~2B.3FC in hex) will be written as A8F1 (which is 43249 in decimal) rather than 2B3FC. In other words, the number is always multiplied by powers of 10, even if the encoding method uses base 16, 2 , or 8.

Show decimal marker:
Clear this check box if you do not want a decimal point to be shown (implied decimals). If you leave this box checked, you can specify the character to be used as a decimal marker. The decimal marker does not have to be a period, you can use any character you like. This setting only applies to the decimal encoding.

▪ Use fixed width:
Check this check box if you want the resulting string to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Pad with zeros / Pad with spaces / Custom padding character:
Specify the character that is to be used to pad the parameter to the fixed width if it is too short. Only applies to the decimal, hexadecimal, binary, and octal encodings.

Alignment:
Choose the alignment of the data within the fixed width. You can choose to have any padding characters added before the sign, between the sign and the number, or after the number. If the sign is not shown, the Sign-padding-number and Padding-sign-number alignments are the same. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ Use capital letters A–F / small letters a–f as hex digits:
Select whether capital or small letters should be used for hex digits. Only applies to the hexadecimal encoding.

▪ Use exponential notation:
Check this box to use exponential (scientific) notation with the decimal encoding method.

Number of non-decimal digits in the mantissa:
Enter the number of digits that should appear on the left of the decimal point of the mantissa (the part of the number that is not the exponent). Note that this setting does not necessarily reflect the number of digits that actually appear before the decimal point. It is merely used to determine the range of the mantissa. If you specify a value of 0, the mantissa will always be between 0 and 1. If you specify 1, it will be either 0, or between 1 and 10, etc.. The number 13895.468, for example, will be shown as 0.13895468E+5 with 0 mantissa digits, as 1.38954680E+4 with one digit, and as 138.95468000E+2 with three digits. The number 0 will always be shown as 0.00000000E+0, regardless of the number of mantissa digits specified. If you want to force 0 to be shown as 000.00000000E+0, you must specify a fixed width, and "0" as the padding character.

Number of digits in the exponent:
Enter the number of digits shown in the exponent. Unlike the number of digits in the mantissa digits, the exponent will always have exactly the number of digits specified here.

Prefix for positive / negative exponent:
Enter the exponent markers for positive and negative exponents here. The exponent markers must include the sign of the exponent. Usually, the positive exponent marker is "E+" or "e+", and the negative marker is "E-" or "e-". If you want to omit the plus sign for positive exponents, specify "E" or "e" for the positive exponent prefix. The exponent prefixes can be any arbitrary data. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

Show as text / Show as hex:
Select the way you want to enter the exponent prefixes. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details.

String Parameter Commands Elements

String parameter command elements encode the value of a string parameter. Command elements for command templates can encode device driver or template parameters; command elements for commands can encode device driver or command parameters.

The New String Parameter Element Dialog
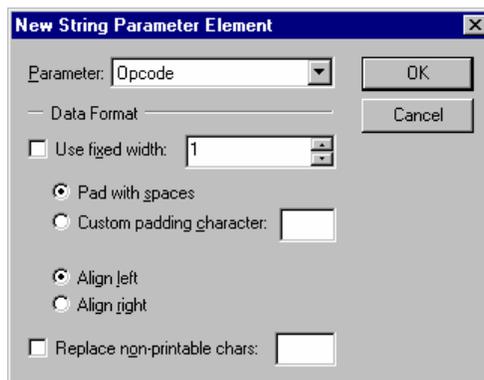
- **Parameter:**

Select the parameter whose value you want to encode. Device driver parameters and command template or command parameters are shown in the same list.

- **Use fixed width:**

Check this check box if you want the resulting string to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

Pad with spaces / Custom padding character:
Specify the character that is to be used to pad the parameter to the fixed width if it is too short.

Align left / Align right:
Select whether the string should be left or right aligned within the fixed width. If the number is left aligned, any padding characters will be added to the end the string, if it is right aligned, they will be added to the beginning.

- **Replace non-printable chars:**

Check this check box if you want all non-printable characters (characters other than ASCII $20-ASCII $7E) in the parameter replaced. Enter the character that you would like to replace them with in the box provided.

Bistate Parameter Bits Commands Elements

Bistate parameter bits command elements encode the value of up to eight bistate parameters in the bits of a one-byte number. Command elements for command templates can encode device driver or template parameters; command elements for commands can encode device driver or command parameters.

The New Bistate Parameter Bits Element Dialog
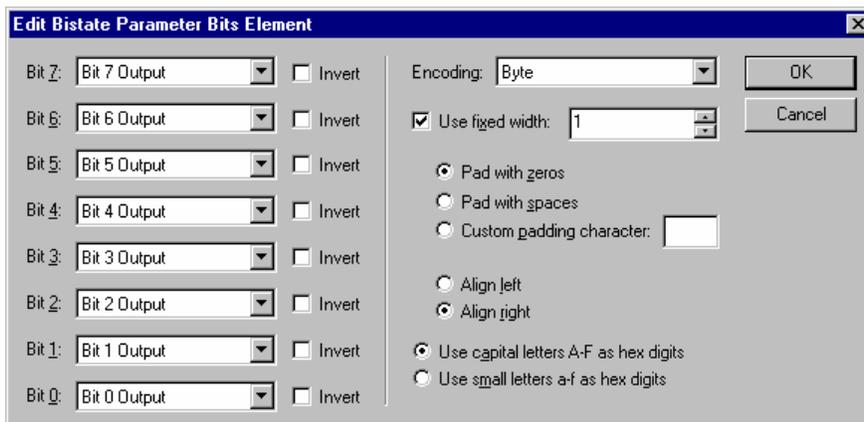
- **Bit 0–7:**
Select the value of the corresponding bit. Select <always 0> to force the bit to 0, select <always 1> to force the bit to one, or select the parameter you want the bit to reflect. If you leave the "Invert" check box cleared, the bit will be set if the parameter is ON, and cleared if it is OFF. If you check the box, the bit will be cleared if the parameter is ON, and set if it is OFF. Device driver parameters and command template or command parameters are shown in the same list.

- **Encoding:**
Select the encoding method for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the number as an 8-bit value. Requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the number as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) will be sent first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) will be sent first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The number is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The number is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding writes the number out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), it does not encode it using the individual bits of each byte.

- **Use fixed width:**
Check this check box if you want the resulting string to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Pad with zeros / Pad with spaces / Custom padding character:
Specify the character that is to be used to pad the number to the fixed width if it is too short. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Align left / Align right:
Select whether the number should be left or right aligned within the fixed width. If the number is left aligned, any padding characters will be added to the end the number; if it is right aligned, they will be added to the beginning. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ Use capital letters A–F / small letters a–f as hex digits:
Select whether capital or small letters should be used for hex digits. This setting only applies to the hexadecimal encoding.

Data Length Commands Elements

Data length command elements calculate and encode the length of the data produced by one or more command elements that follow it.

> Note: The data length of a checksum element will only be calculated correctly of the checksum element has a fixed width.

The New Data Length Element Dialog
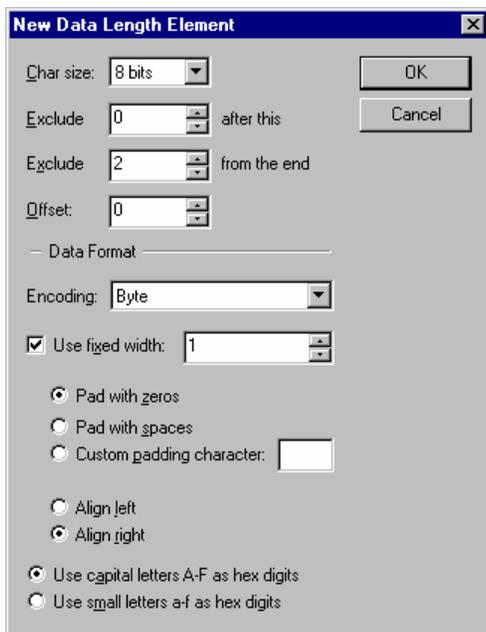
▪ Char size:
Select the size of the characters. If you select 8 bits, the length will be the number of bytes in the data. If you select 16, 24, or 32 bits, the length will be the number of 16, 24, or 32 bit words in the data. Most equipment uses 8-bit characters.

Please note that 7-bit characters are represented internally using 8 bits, with the highest bit being always 0. Select a character size of 8 bits if the equipment uses 7-bit characters.

▪ Exclude ... after this:

By default, data length command elements calculate the total length of all command elements that follow it. If you want to exclude the length of the data produced by some command elements immediately after this one, enter the number of elements to exclude here. If you specify 2 in this field, the length of the data produced by the two next command elements will not be included in the data length value.

▪ Exclude ... from the end:

By default, data length command elements calculate the total length of all command elements that follow it. If you want to exclude the length of the data produced by some command elements at the end of the command string, enter the number of elements to exclude here. If you specify 2 in this field, the length of the data produced by the last two command elements in the list will not be included in the data length value.

▪ Offset:

If, for some reason, you need to add a fixed number to the calculated data length, enter that number here. To subtract a fixed number, enter a negative number.

▪ Encoding:

Select the encoding method for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:

A single byte (character) in the command string is used to represent the data length as an 8-bit value. Requires a fixed width of 1.

Multibyte:

Two or more bytes (characters) in the command string are used to represent the data length as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) will be sent first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) will be sent first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:

The length is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:

The data length is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding writes the data length out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), it does not encode it using the individual bits of each byte.

▪ Use fixed width:

Check this check box if you want the resulting string to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Pad with zeros / Pad with spaces / Custom padding character:

Specify the character that is to be used to pad the data length to the fixed width if it is too short. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Align left / Align right:
Select whether the number should be left or right aligned within the fixed width. If the number is left aligned, any padding characters will be added to the end the number; if it is right aligned, they will be added to the beginning. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ Use capital letters A-F / small letters a-f as hex digits:
Select whether capital or small letters should be used for hex digits. This setting only applies to the hexadecimal encoding.

Checksum Command Elements

Checksum command elements calculate and encode the checksum of the data produced by one or more command elements that precede it.

The New Checksum Element Dialog
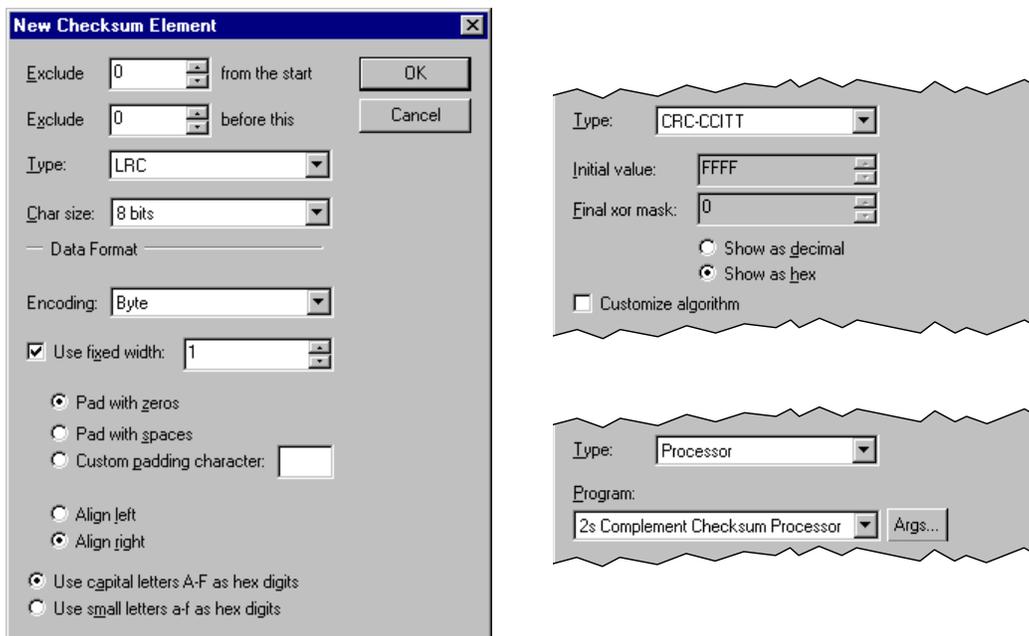
▪ Exclude ... from the start:
By default, checksum command elements calculate the checksum of all command elements that precede it. If you want to exclude the data produced by some command elements at the beginning of the command string, enter the number of elements to exclude here. If you specify 2 in this field, the data produced by the first two command elements in the list will not be included in the checksum calculation.

▪ Exclude ... after this:
By default, checksum command elements calculate the checksum of all command elements that precede it. If you want to exclude the data produced by some command elements immediately before this one, enter the number of elements to exclude here. If you specify 2 in this field, the data produced by the last two command elements before this one will not be included in the checksum calculation.

▪ Type:
Select the type of checksum calculation. U.P.M.A.C.S. supports the following types of check-sums:

Simple sum:
The checksum is the sum of all the character values in the data string.

You can specify the size of the individual characters. Most equipment uses 8 bits (1 byte) per character. If you select 16, 24, or 32 bit characters, each group of 2, 3, or 4 bytes will be treated as a single character value. For character sizes greater than 8 bits, you can also choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the first byte in each character is the least significant byte (the lower 8 bits); if you select hi-lo byte ordering, the first byte in each character is the most significant byte (the upper 8 bits). Select the size of each character, as well as the byte ordering (lo-hi or hi-lo), in the "Char size" field.

Please note that 7-bit characters are represented internally using 8 bits, with the highest bit being always 0. Select a character size of 8 bits if the equipment uses 7-bit characters.

LRC:
Longitudinal redundancy check. The checksum is calculated by XORing all the character values together.

You can specify the size of the individual characters. Most equipment uses 8 bits (1 byte) per character. If you select 16, 24, or 32 bit characters, each group of 2, 3, or 4 bytes will be treated as a single character value. For character sizes greater than 8 bits, you can also choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the first byte in each character is the least significant byte (the lower 8 bits); if you select hi-lo byte ordering, the first byte in each character is the most significant byte (the upper 8 bits). Select the size of each character, as well as the byte ordering (lo-hi or hi-lo), in the "Char size" field.

Please note that 7-bit characters are represented internally using 8 bits, with the highest bit being always 0. Select a character size of 8 bits if the equipment uses 7-bit characters.

CRC-16, CRC-CCITT, CRC-32:
Cyclic redundancy check. The checksum is calculated using a binary polynomial. The different CRCs use different standardized polynomials.

The CRC algorithm has two arbitrary parameters: The initial value (sometimes called the seed) of the calculation, and a value that the final result is XORed with. U.P.M.A.C.S. provides default values for these parameters, shown in the "Initial value" and "final xor mask" boxes. Select the "Show as decimal" or "Show as hex" radio buttons to view the values as decimal or hex numbers.

If you need to use values other than those provided, you can check the "Customize algorithm" check box and edit the parameters. CRC-16 and CRC-CCITT use 16-bit polynomials, so the parameters must be 16-bit numbers. CRC-32 uses a 32-bit polynomial, so the parameters are 32-bit numbers.

Modulo 128:
The checksum is the sum of all the data bytes, modulo 128. This is equivalent to ANDing the checksum with hex 7F. Only the 7 least significant bits of the sum are used.

Modulo 256:
The checksum is the sum of all the data bytes, modulo 256. This is equivalent to ANDing the checksum with hex FF. Only the least significant byte of the sum is used.

Printable checksum:
This is a special checksum designed to produce a checksum that is always a printable character. It is calculated as follows:

> ➢ Subtract 32 from each character code,

> ➢ add the results together,

> ➢ take the sum modulo 95,

> ➢ add 32 to the result.

This type of checksum is used, for example, by Miteq equipment.

Example:

To calculate the checksum for the string "Prometheus", proceed as follows:

Subtract 32 from each character code:

Char:	P	r	o	m	e	t	h	e	u	s
Code:	80	114	111	109	101	116	104	101	117	115
	-32	-32	-32	-32	-32	-32	-32	-32	-32	-32
result:	48	82	79	77	69	84	72	69	85	83

Add the results together:

48 + 82 + 79 + 77 + 69 + 84 + 72 + 69 + 85 + 83 = 748

Take the sum modulo 95:

748 mod 95 = 83

And add 32 to the result:

83 + 32 = 115 (" s ")

Processor:
If you need a checksum calculation that is not listed above, you can write an SCL program to calculate the checksum. Select the program that will do the calculation from the "Program" list, and press the "Args..." button to specify program arguments. See *Specifying Arguments for SCL Programs* in the *Developer's Manual* for a description of the Edit Program Arguments dialog.

See *Programs for Sources, Checksums, and SABus Response Data* in the *SCL Language Reference* for details on writing SCL programs that calculate checksums.

▪ Encoding:
Select the encoding method for the checksum. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the checksum as an 8-bit value. Requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the checksum as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) will be sent first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) will be sent first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The checksum is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The checksum is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding writes the checksum out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), it does not encode it using the individual bits of each byte.

- Use fixed width:

Check this check box if you want the resulting string to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Pad with zeros / Pad with spaces / Custom padding character:
Specify the character that is to be used to pad the checksum to the fixed width if it is too short. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Align left / Align right:
Select whether the number should be left or right aligned within the fixed width. If the number is left aligned, any padding characters will be added to the end the number; if it is right aligned, they will be added to the beginning. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

- Use capital letters A–F / small letters a–f as hex digits:

Select whether capital or small letters should be used for hex digits. This setting only applies to the hexadecimal encoding.

Command Data Command Elements

Command data command elements are only used in command templates, not in commands. The data string for this type of command element is specified by the command that uses the template, not by the template itself.

The New Command Data Element Dialog

- Tag:
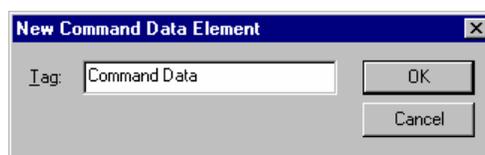
Enter the tag by which the element is identified. Each command data command element in a template must have a unique tag.

RESPONSES ELEMENTS

Overview

Response elements are used in command templates to check the data returned by the equipment in response to a command. Each response element is used to verify the validity of a section of the response.

You create and edit command elements from within the New Command Template dialog. Each command template can have a set of response elements that describes a valid response, and a set that describes an error response.

If you specify both an error and a valid response, make sure you provide enough information to tell the two apart. If a response received from the equipment matches the format you specified for a valid response and the format you specified for an error response, it is assumed to be an error response. It is hence important that valid responses returned by the equipment do not match the error response you specified.

Each response element can have a condition attached to it. The condition checks the value of numerical or string value response element to see whether the element should appear in the response or not. See *Response Element Conditions* on page 42 for details.

When you create a new command element, you will be asked to select its type. There are nine types of command elements.

General types of response elements:

> ➢ Regular expression
> ➢ Command element

Elements that check response data against the value of a parameter:

> ➢ Bistate parameter
> ➢ Digital parameter (number)
> ➢ Digital parameter (strings)
> ➢ Analog parameter
> ➢ String parameter

Special types of response elements:

> ➢ Numerical value
> ➢ String value
> ➢ Checksum
> ➢ Response data*
> ➢ Numerical error code**
> ➢ String error code**

* Command data command elements are only used by the valid response

** Error code response elements are only used by the error response

Response Element Conditions

Each response element can have a condition attached to it. The condition checks the value of numerical or string value response element to see whether the element should appear in the response or not.

To add, remove or change a condition to a response element press the "Condition..." button in the New Response Element dialog.

The Response Element Condition Dialog
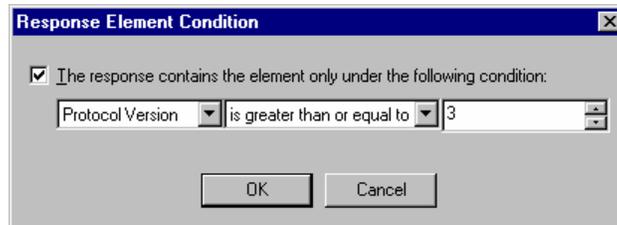
▪ **The response contains the element only under the following condition:**
Check the check box to specify the condition under which the element will appear in the response. If you leave the check box blank, the element should always appear in the response.

Select the numerical or string value response element whose value should be checked on the left. Select the condition in the center. Which conditions appear in the list depends on the type of the value you selected.

For numerical values, you will get the following options:

is equal to

is not equal to

is less than

is greater than

is less than or equal to

is greater than or equal to

Enter the number you want to compare the value to on the right.

For string values, you will get the following options:

is

is not

contains

does not contain

Enter a regular expression on the right. If you selected *is* or *is not*, the expression must (or must not) match the string exactly. If you select *contains* or *does not contain*, the expression merely has to appear somewhere in the string.

If the value response element you specify does itself not appear in the response because of a condition attached to it, this response element will not appear either.

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

Regular Expression Responses Elements

Regular expression response elements check a section of the response data using a regular expression. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

The New Regular Expression Element Dialog
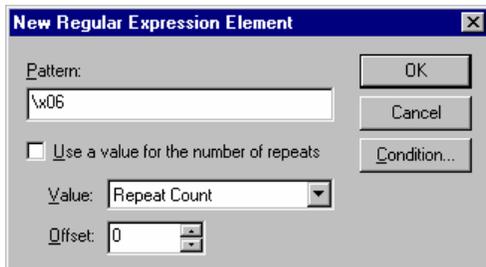
- Pattern:

Enter a regular expression that describes the data.

- Use a value for the number of repeats:

Check this box if you want to use a numerical value response element as a repeat count. The regular expression must then appear as many times in the response as specified in the value.

Value:

Select the value element to use as a repeat count here.

Offset:

Enter a number to add to the value. Enter a positive value if the element is repeated more often than the value specifies, enter a negative value if it is repeated less often.

- The "Condition" button:

Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

Command Element Responses Elements

Command element response elements check that the same data that was sent for one of the template's command elements was returned in the response.

The New Command Element Dialog
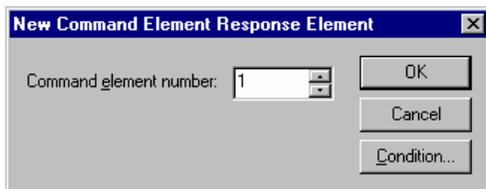
▪ Command element number:
Enter the number of the command element. The first command elements in the template has command element number 1, the second one 2, and so on.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

Bistate Parameter Responses Elements

Bistate parameter response elements extract a bistate value from the response using two regular expressions, and check it against the value of a bistate parameter of the device driver or template. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

The New Bistate Parameter Element Dialog
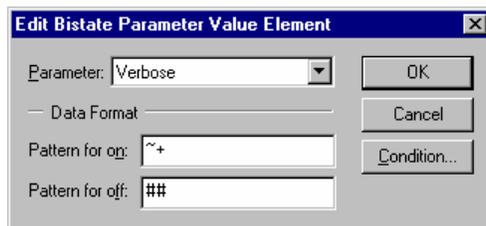
▪ Parameter:
Select the parameter whose value should appear in the response. Device driver parameters and command template parameters are shown in the same list.

▪ Pattern for on:
Enter a regular expression that describes the data string meaning ON.

▪ Pattern for off:
Enter a regular expression that describes the data string meaning OFF.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

Digital Parameter (Number) Responses Elements

Digital parameter (number) response elements extract a number from the response and check it against a digital parameter of the device driver or template.

The New Digital Parameter Element Dialog

▪ Parameter:

Select the parameter whose value should appear in the response. Device driver parameters and command template parameters are shown in the same list.

▪ Encoding:

Select the encoding method that the equipment uses for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:

A single byte (character) in the command string is used to represent the parameter as an 8-bit value. Requires a fixed width of 1.

Multibyte:

Two or more bytes (characters) in the command string are used to represent the parameter as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:

The parameter is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:

The parameter is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the parameter to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

▪ Skip leading characters:

Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

▪ Use fixed width:
Check this check box to require the number to be a fixed number of bytes (characters) long.
Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the re-
sponse. This only applies to characters within the fixed width. All characters are always allowed
to appear beyond the fixed width. Specify the characters that are allowed to appear after the
number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element.
See *Response Element Conditions* on page 42 for a description of the Response Element Condi-
tion dialog.

Digital Parameter (Strings) Responses Elements

Digital parameter (strings) response elements extract a number value from the response using a
set of regular expressions, and check it against the value of a digital parameter of the device
driver or template. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on
regular expressions.

The New Digital Parameter Element Dialog
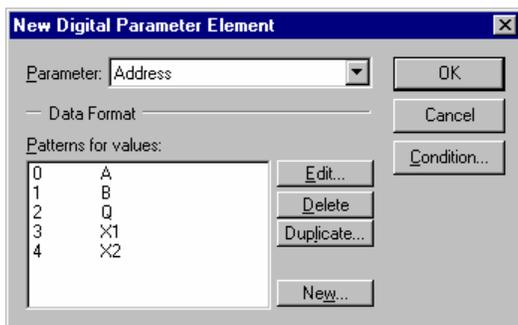
▪ Parameter:
Select the parameter whose value should appear in the response. Device driver parameters and
command template parameters are shown in the same list.

▪ Patterns for values:
Shows a list of all values that you specified regular expressions for, together with the expression
that describes the data string for that value. Use the buttons to edit, delete, duplicate, and add
values.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element.
See *Response Element Conditions* on page 42 for a description of the Response Element Condi-
tion dialog.

Analog Parameter Responses Elements

Analog parameter response elements extract a number from the response and check it against an analog parameter of the device driver or template.

You can provide a factor and an offset to be applied to the value after reading it. The value that the parameter must match is calculated from the number in the response as follows:

parameter = number · factor + offset

To use the number unaltered, specify a factor of 1 and an offset of 0.

The New Analog Parameter Element Dialog
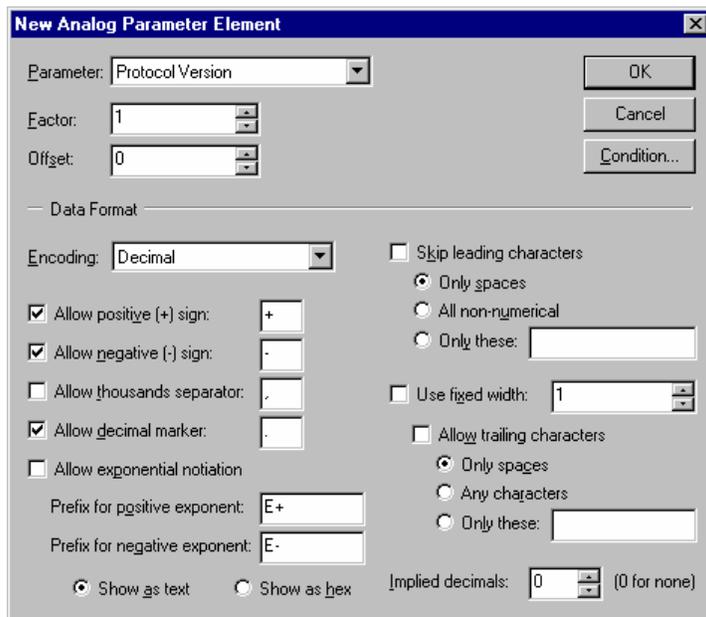
- **Parameter:**
Select the parameter whose value should appear in the response. Device driver parameters and command template parameters are shown in the same list.

- **Factor:**
Enter the factor with which the number in the response is to be multiplied before verifying it. The factor is applied before the offset.

- **Offset:**
Enter the offset that is to be added to the number in the response before verifying it. The offset is applied after the factor.

- **Encoding:**
Select the encoding method that the equipment uses for the parameter. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the parameter as an 8-bit value. If you select signed byte encoding, the most significant bit of the byte will be interpreted as

the sign in the standard way ($FF is -1, $FE is -2, etc.). The byte encoding method requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the parameter as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. If you select signed multibyte encoding, the most significant bit of the most significant byte will be interpreted as the sign in the standard way (for 16-bit values $FFFF is -1,$FFFE is -2, etc.). The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The parameter is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The parameter is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the parameter to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

- Allow positive (+) sign:
Check this box to allow a plus sign to denote positive numbers. The plus sign does not actually have to be the "+" character. You can specify any character you want.

This setting only applies to the decimal encoding.

- Allow negative (−) sign:
Check this box to allow a minus sign to denote negative numbers. If you leave this check box blank, all numbers will be positive. The minus sign does not actually have to be the "-" character. You can specify any character you want.

This setting only applies to the decimal encoding.

- Allow thousands separator:
Check this check box if you want digits to be separated into groups of three using a thousands separators (usually a comma). If you check this box, you must also specify the actual character used to group digits.

This setting only applies to the decimal encoding.

- Allow decimal marker:
Check this box to allow a decimal marker and decimals. If you leave this check box blank, all numbers will be whole numbers. If you leave this box checked, you can specify the character used as a decimal marker.

This setting only applies to the decimal encoding.

- Allow exponential notation:
Check this box to allow exponential (scientific) notation with the decimal encoding method.

Prefix for positive / negative exponent:
Enter the exponent markers for positive and negative exponents here. The exponent markers must include the sign of the exponent. Usually, the positive exponent marker is "E+" or "e+", and

the negative marker is "E-" or "e-". If the plus sign is omitted for positive exponents, specify "E" or "e" for the positive exponent prefix. The exponent prefixes can be any arbitrary data. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

Show as text / Show as hex:
Select the way you want to enter the exponent prefixes. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details.

▪ Skip leading characters:
Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

▪ Use fixed width:
Check this check box to require the number to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the response. This only applies to characters within the fixed width. All characters are always allowed to appear beyond the fixed width. Specify the characters that are allowed to appear after the number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ Implied decimals:
Specify the number of implied decimals here. Implied decimals are a way of encoding fractions without using a decimal point by multiplying it with a power of ten before encoding it. A number that is encoded using 3 implied decimals, for example, will be multiplied by 1000 before it is encoded. This will move the three decimals from the right to the left of the decimal point, and the decimal point will no longer be needed. 21.304, for example, will simply be encoded as 21304.

Please note that the implied decimals for the hexadecimal, octal, and binary encoding methods are decimal fractions, not hexadecimal, octal or binary fractions. In hexadecimal encoding with three implied decimals, 43.249 (which is ~2B.3FC in hex) will appear as A8F1 (which is 43249 in decimal) rather than 2B3FC. In other words, the number is always multiplied by powers of 10, even if the encoding method uses base 16, 2 , or 8.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

String Parameter Responses Elements

String parameter response elements extract a data string from the response and check it against a string parameter of the device driver or template.

String parameter response elements use regular expression patterns to recognize the parameter value. The data is divided into three sections: A prefix, the value, and a suffix. The prefix and suffix are discarded, and only the value is checked against the parameter..

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

The New String Parameter Element Dialog
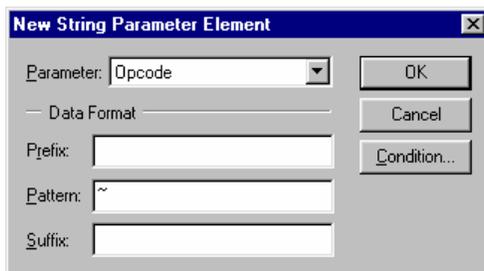
- Parameter:

Select the parameter whose value should appear in the response. Device driver parameters and command template parameters are shown in the same list.

- Prefix:

Enter a regular expression that describes any data that appears before the parameter in the response. Any data that matches the prefix will be discarded.

- Pattern:

Enter a regular expression that describes the parameter here.

- Suffix:

Enter a regular expression that describes any data that appears after the parameter in the response. Any data that matches the suffix will be discarded.

- The "Condition" button:

Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

Numerical Value Responses Elements

Numerical value response elements extract a number from the response. The number can then be used by other response elements as a data length, a repeat count, or in response element conditions.

The New Numerical Value Element Dialog
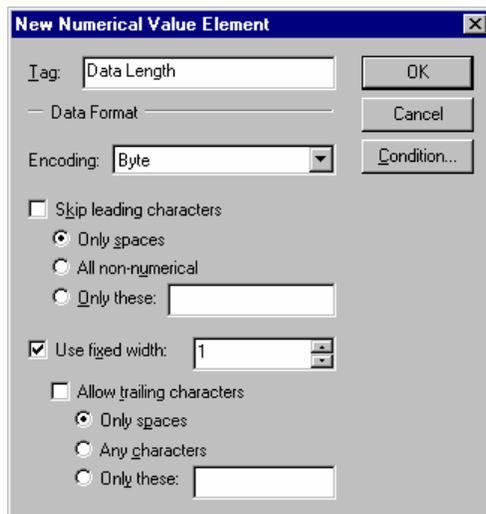
▪ Tag:
Enter the tag by which the value is identified. Each value response element (string or number) in each response (valid and error) must have a unique tag.

▪ Encoding:
Select the encoding method that the equipment uses for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the number as an 8-bit value. Requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the number as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The value is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The value is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the value to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

▪ Skip leading characters:
Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

▪   Use fixed width:
Check this check box to require the number to be a fixed number of bytes (characters) long.
Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the re-
sponse. This only applies to characters within the fixed width. All characters are always allowed
to appear beyond the fixed width. Specify the characters that are allowed to appear after the
number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪   The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element.
See *Response Element Conditions* on page 42 for a description of the Response Element Condi-
tion dialog.

String Value Responses Elements

String value response elements extract a string from the response, to be used by other response
elements in response element conditions.

String parameters use regular expression patterns to recognize the value. The data is divided into
three sections: A prefix, the value, and a suffix. The prefix and suffix are discarded, and only the
value is stored.

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expres-
sions.

The New String Value Element Dialog
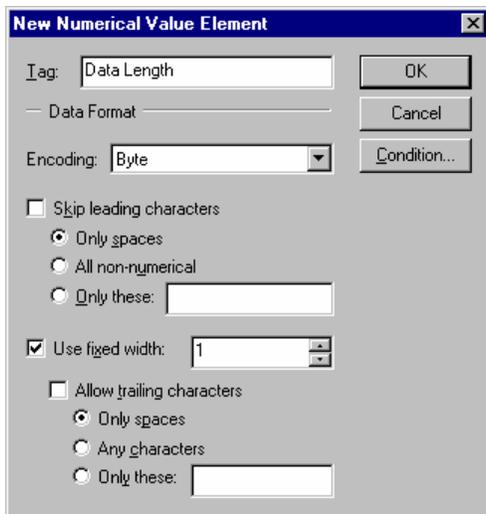
▪   Tag:
Enter the tag by which the value is identified. Each value response element (string or number) in
each response (valid and error) must have a unique tag.

- Prefix:

Enter a regular expression that describes any data that appears before the value in the response. Any data that matches the prefix will be discarded.

- Pattern:

Enter a regular expression that describes the value here.

- Suffix:

Enter a regular expression that describes any data that appears after the value in the response. Any data that matches the suffix will be discarded.

- The "Condition" button:

Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

Checksum Responses Elements

Checksum response elements extract a number from the response and check it against the checksum of the data received for one or more response elements that precede it.
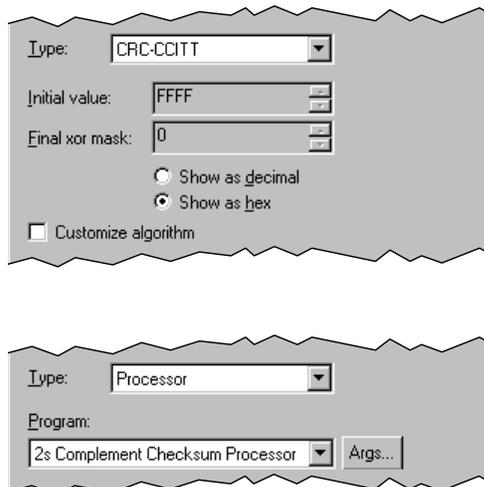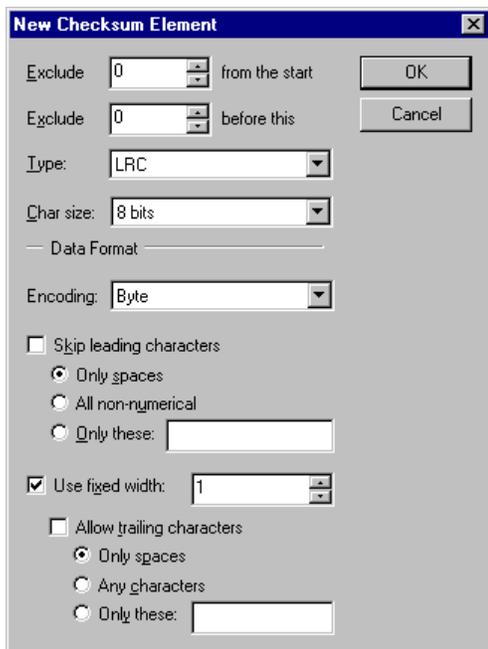
The New Checksum Element Dialog

- Exclude … from the start:

By default, checksum response elements calculate the checksum of all response elements that precede it. If you want to exclude the data of some response elements at the beginning of the response, enter the number of elements to exclude here. If you specify 2 in this field, the data for the first two response elements in the list will not be included in the checksum calculation.

- Exclude … after this:

By default, checksum response elements calculate the checksum of all response elements that precede it. If you want to exclude the data of some response elements immediately before this

one, enter the number of elements to exclude here. If you specify 2 in this field, the data for the last two response elements before this one will not be included in the checksum calculation.

- Type:

Select the type of checksum calculation. U.P.M.A.C.S. supports the following types of checksums:

Simple sum:
The checksum is the sum of all the character values in the received data.

You can specify the size of the individual characters. Most equipment uses 8 bits (1 byte) per character. If you select 16, 24, or 32 bit characters, each group of 2, 3, or 4 bytes will be treated as a single character value. For character sizes greater than 8 bits, you can also choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the first byte in each character is the least significant byte (the lower 8 bits); if you select hi-lo byte ordering, the first byte in each character is the most significant byte (the upper 8 bits). Select the size of each character, as well as the byte ordering (lo-hi or hi-lo), in the "Char size" field.

Please note that 7-bit characters are represented internally using 8 bits, with the highest bit being always 0. Select a character size of 8 bits if the equipment uses 7-bit characters.

LRC:
Longitudinal redundancy check. The checksum is calculated by XORing all the character values together.

You can specify the size of the individual characters. Most equipment uses 8 bits (1 byte) per character. If you select 16, 24, or 32 bit characters, each group of 2, 3, or 4 bytes will be treated as a single character value. For character sizes greater than 8 bits, you can also choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the first byte in each character is the least significant byte (the lower 8 bits); if you select hi-lo byte ordering, the first byte in each character is the most significant byte (the upper 8 bits). Select the size of each character, as well as the byte ordering (lo-hi or hi-lo), in the "Char size" field.

Please note that 7-bit characters are represented internally using 8 bits, with the highest bit being always 0. Select a character size of 8 bits if the equipment uses 7-bit characters.

CRC-16, CRC-CCITT, CRC-32:
Cyclic redundancy check. The checksum is calculated using a binary polynomial. The different CRCs use different standardized polynomials.

The CRC algorithm has two arbitrary parameters: The initial value (sometimes called the seed) of the calculation, and a value that the final result is XORed with. U.P.M.A.C.S. provides default values for these parameters, shown in the "Initial value" and "final xor mask" boxes. Select the "Show as decimal" or "Show as hex" radio buttons to view the values as decimal or hex numbers.

If you need to use values other than those provided, you can check the "Customize algorithm" check box and edit the parameters. CRC-16 and CRC-CCITT use 16-bit polynomials, so the parameters must be 16-bit numbers. CRC-32 uses a 32-bit polynomial, so the parameters are 32-bit numbers.

Modulo 128:
The checksum is the sum of all the data bytes, modulo 128. This is equivalent to ANDing the checksum with hex 7F. Only the 7 least significant bits of the sum are used.

Modulo 256:
The checksum is the sum of all the data bytes, modulo 256. This is equivalent to ANDing the checksum with hex FF. Only the least significant byte of the sum is used.

Printable checksum:
This is a special checksum designed to produce a checksum that is always a printable character. It is calculated as follows:

➢ Subtract 32 from each character code,

➢ add the results together,

➢ take the sum modulo 95,

➢ add 32 to the result.

This type of checksum is used, for example, by Miteq equipment.

Example:
To calculate the checksum for the string "Prometheus", proceed as follows:
Subtract 32 from each character code:

Char:	P	r	o	m	e	t	h	e	u	s
Code:	80	114	111	109	101	116	104	101	117	115
	-32	-32	-32	-32	-32	-32	-32	-32	-32	-32
result:	48	82	79	77	69	84	72	69	85	83

Add the results together:
    48 + 82 + 79 + 77 + 69 + 84 + 72 + 69 + 85 + 83 = 748
Take the sum modulo 95:
    748 mod 95 = 83
And add 32 to the result:
    83 + 32 = 115 (" s ")

▪ **Processor:**
If you need a checksum calculation that is not listed above, you can write an SCL program to calculate the checksum. Select the program that will do the calculation from the "Program" list, and press the "Args..." button to specify program arguments. See *Specifying Arguments for SCL Programs* in the *Developer's Manual* for a description of the Edit Program Arguments dialog.

See *Programs for Sources, Checksums, and SABus Response Data* in the *SCL Language Reference* for details on writing SCL programs that calculate checksums.

▪ **Encoding:**
Select the encoding method that the equipment uses for the checksum. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the checksum as an 8-bit value. Requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the checksum as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The checksum is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as

the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The checksum is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the checksum to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

▪ Skip leading characters:
Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

▪ Use fixed width:
Check this check box to require the number to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the response. This only applies to characters within the fixed width. All characters are always allowed to appear beyond the fixed width. Specify the characters that are allowed to appear after the number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

Response Data Responses Elements

Response data response elements contain the data that is to be used by data objects. Response data response elements are only used by the valid response, not by the error response.

The New Response Data Element Dialog

- Tag:

Enter the tag by which the element is identified. Each response data response element in a template must have a unique tag.

- Get length from a value:

Check this box if you want the data length to be taken from a numerical value response element.

You can provide an offset, a factor, and an offset for the factor, to be applied to the value after reading it. The data length is calculated from the value as follows:

$$\text{data length} = (\text{value} + \text{offset}) \cdot (\text{factor} + \text{factor offset})$$

To use the number unaltered, specify a factor of 1 and an offset of 0.

Value:
Select the value you want to use as the data length.

Char size:
Select the size of the characters. If you select 8 bits, the length will be the number of bytes in the data. If you select 16, 24, or 32 bits, the length will be the number of 16, 24, or 32 bit words in the data. Most equipment uses 8-bit characters.

Please note that 7-bit characters are represented internally using 8 bits, with the highest bit being always 0. Select a character size of 8 bits if the equipment uses 7-bit characters.

Includes ... element before this:
If the value includes the length of one or more other response elements immediately before this, enter a number here. If you enter 3, then the data length gotten from the value (after applying the offset and factor) will be considered to be the length of the data of this response element and that of the three elements before it.

Offset:
Enter the offset that is to be added to the value to get the data length. The offset is applied to the value before multiplying it with the factor.

Fixed factor:
If the factor with which the value is to be multiplied to get the data length is a fixed number, enter it here. You cannot specify a factor offset for fixed factors. Add the offset manually.

Factor value:
If the factor with which the value is to be multiplied to get the data length is to be taken itself from a value, select that value here.

Factor offset:
Enter the offset that is to be added to the factor value to get the final factor.

▪ Pattern:
Enter a regular expression that describes the response data. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

Numerical Error Code Responses Elements

Numerical error code response elements retrieve an error code from the response. They are used only by the error response, not by the valid response. See *Error Handling* on page 20 for more details.

The New Numerical Error Code Element Dialog
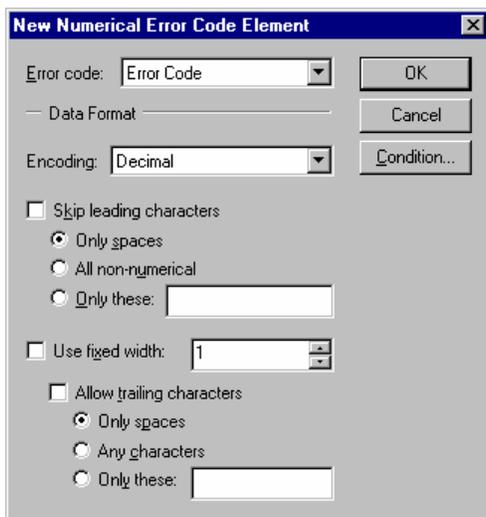
▪ Encoding:
Select the encoding method that the equipment uses for the error code. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the error code as an 8-bit value. Requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the error code as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you

select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The error code is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The error code is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the error code to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

▪ Skip leading characters:
Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

▪ Use fixed width:
Check this check box to require the number to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the response. This only applies to characters within the fixed width. All characters are always allowed to appear beyond the fixed width. Specify the characters that are allowed to appear after the number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ The "Condition" button:
Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.


String Error Code Responses Elements

String error code response elements retrieve an error code from the response. They are used only by the error response, not by the valid response. See *Error Handling* on page 20 for more details.

String error code response elements use regular expression patterns to recognize the error code. The data is divided into three sections: A prefix, the error code, and a suffix. The prefix and suffix are discarded, and only the error code is used.

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

The New String Error Code Element Dialog
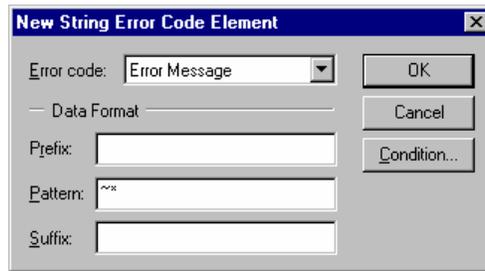
- **Prefix:**
Enter a regular expression that describes any data that appears before the error code in the response. Any data that matches the prefix will be discarded.

- **Pattern:**
Enter a regular expression that describes the error code here.

- **Suffix:**
Enter a regular expression that describes any data that appears after the error code in the response. Any data that matches the suffix will be discarded.

- **The "Condition" button:**
Press this button to add, remove, or edit the response element condition attached to this element. See *Response Element Conditions* on page 42 for a description of the Response Element Condition dialog.

DATA OBJECT SOURCES

Overview

Each data object has one or more sources that determines where it gets its value from. There are many different types of sources for each type of data object.

There are three types of sources used by all types of objects:

- ➢ Processor sources
- ➢ Summary sources*
- ➢ Parameter sources**

Bistate objects can have six additional types of sources:

- ➢ On/Off string sources
- ➢ Bit mask sources
- ➢ Search string sources
- ➢ Digital value list sources*
- ➢ Value limits sources*
- ➢ Pattern match sources*

Digital objects can have four additional types of sources:

- ➢ Digital number sources
- ➢ Set of strings sources
- ➢ Thresholds sources
- ➢ Bit collection sources

Analog objects can have one additional type of source:

- ➢ Analog number sources

String objects can have two additional types of sources:

- ➢ String sources
- ➢ Filter sources

Data objects can have more than one source. See *Multiple Sources* on page 93 for details.

Most sources get the value from a response data response element of a command. See the foot-notes for exceptions.

---

* These sources take their values from other data objects
** This source does not take it's value from anywhere. The value of objects with parameter sources must be set directly by SCL programs.

Serial Data Sources

Most sources use response data from a command's response. You can specify which part of the data is to be used, and how.

Finding the Relevant Data within the Response

There are two ways in which to specify the part of the data that you wish to use.

▪ Using an offset from the beginning of the data:
You can specify a byte offset from the beginning of the response data element. The data used by the source will begin a fixed number of bytes from the beginning of the response data. Use this method if you know how many bytes from the beginning of the response data you are interested in is located.

▪ Using a search key:
You can use a search key to find the relevant data. The section of the data used by the source will begin a fixed number of bytes before or after the end of the first occurrence of the search key in the response data. Use this method if the data objects in the response are variable length and separated by separators, like spaces or commas. You can also use this method if the data is preceded by some character or characters: a piece of equipment might mark the position of a frequency with the letter F, for example.

▪ Using Offset Parameters
You can use one or more of the digital parameters of the data objects to calculate additional offset values that will be added to the offset you specified. You can specify a formula to be used to calculate the additional offset from the parameter value. The formula has the following form:

data offset = (parameter value + value offset) · value factor


Example:

You are creating an object that gets the alarm state of a modem, and all the information about all the modems appears in a single response. The alarm state information for modem number 1 is located at byte offset 10, and that the alarm states for the modems are 12 bytes apart. You should then specify a fixed offset of 10, and use the modem number as an offset parameter with value offset -1 and value factor 12. The total offset for the different modems will then be calculated as follows:

total offset = 10 + (modem number - 1) · 12

This gives the following offsets for the first four modems:

➢ Modem number 1: 10 + (1 - 1) · 12 = 10
➢ Modem number 2: 10 + (2 - 1) · 12 = 22
➢ Modem number 3: 10 + (3 - 1) · 12 = 34
➢ Modem number 4: 10 + (4 - 1) · 12 = 46

▪ The Edit Offset Parameters and Edit Offset Parameter Dialogs
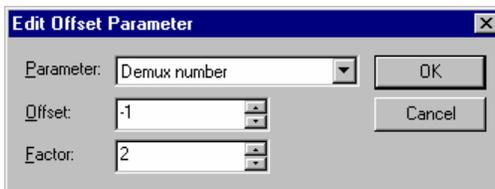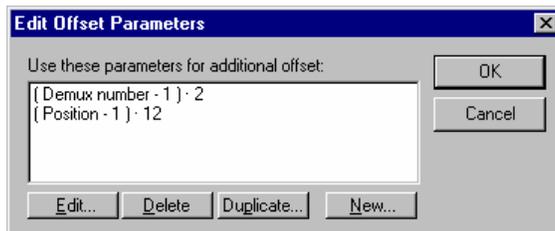
Use these parameters for additional offset:
Shows a list of all the formulas used to calculate the additional offsets. Use the buttons to edit, delete, duplicate, or add formulas.

Parameter:
Select the parameter whose value should be used to calculate the offset.

Offset:
Enter the value offset for the parameter. The value offset is applied before the value factor.

Factor:
Enter the value factor for the parameter. The value offset is applied after the value offset.

The Serial Data Source Dialogs

The Source dialogs for sources that use serial data share the following fields:
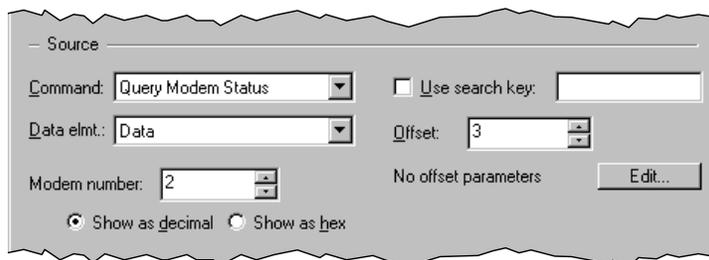
The list below describes only those fields that all the dialogs have. For the remaining fields, see the section on the appropriate source.

▪ Command:
Select the command whose response contains the data used for the object. You can specify values for all the command parameters below the "data elmt" field.

▪ Data elmt:

Select the response data element that contains the data used for the object. Specify values for all the command parameters immediately below. In the sample dialog, the source uses a command that has one digital parameter called "Modem number".

If you want the command to inherit one of the object's parameters, leave the parameter value field blank. See *Parameter Value Inheritance* on page 10 for details. If you do not specify a value for a parameter that is not inherited, a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Show as decimal / Show as hex:

If the command has any parameters of type digital that don't have value names, you can select the way you want to enter the parameter values here. Select "Show as decimal" to enter the values in decimal, select "Show as hex" to enter the values in hexadecimal.

Show as text / Show as hex:

If the command has any parameters of type string, you can select the way you want to enter the parameter values here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

▪ Use search key:

Check the check box if you want to use a search key. If you check this box, the offset will be calculated from the end of the first match for the search key. Enter a regular expression for the key into the entry field.

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

▪ Offset:

Enter the zero-based offset of the data within the buffer. If you did not specify a search key, the offset is calculated from the beginning of the buffer. If you specified a search key, the offset is calculated from the end of the first match for the search key.

If the data you are interested in is right at the beginning of the buffer, or right after the search key, enter 0.

If there are n bytes between the beginning of the buffer or end of the search key and the data, enter n.

If the data includes the last n bytes of the search key, enter -n.

▪ ... offset parameters:

Shows how many offset parameters you specified. Press the "Edit..." button to add, remove, or change offset parameters.

See *Using Offset Parameters* on page 62 for a description of the Edit Offset Parameters dialog.

Processor Sources

Processor sources take information from a response data buffer in a serial device. You have to provide an SCL program to decode the data. See *Programs for Sources, Checksums, and SABus Response Data* in the *SCL Language Reference* for details on writing SCL programs that calculate checksums.

Use processor sources for objects whose value comes from a response to a command, but none of the other sources types can be used to interpret the data.

The Processor Source Dialog
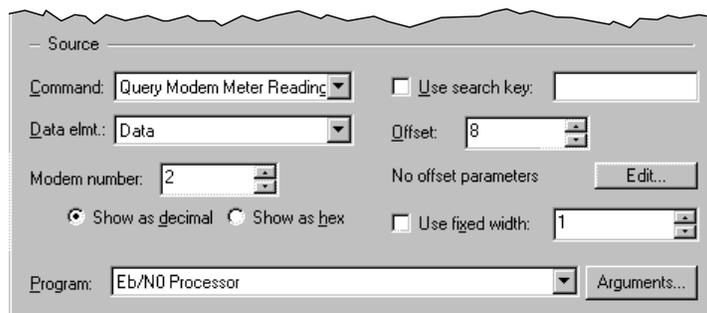
The list below describes only those fields that are specific to the Processor Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

▪ **Use fixed width:**
Check this check box to pass a fixed number of characters to the SCL program. If you leave this box blank, all characters up to the end of the response data element will be passed to the SCL program.

▪ **Program:**
Select the SCL program that is to do the evaluation of the data. The program arguments are shown in parentheses after the name of the program, but you only select the program from the lists, not the arguments. To change the arguments, use the "Arguments..." button. See *Specifying Arguments for SCL Programs* in the *Developer's Manual* for a description of the Edit Program Arguments dialog.

Summary Sources

Summary sources take information from a number of other data objects. You can define the way the values of the summarized objects are evaluated using an SCL program, or you can use the default behaviour. If you want to use the default behaviour, all the summarized objects must be bistate objects. Otherwise, they can be of any type. The default behaviour for the different types of object is as following:

Bistate objects:
The object will be set to ON if any of the summarized objects are ON. The object will be set OFF state if all of the summarized objects are OFF.

Digital objects:
The value of the object will be set to the number of summarized objects that are ON.

Analog objects:
The value of the object will be set to the fraction of summarized objects that are ON, i.e. the number of objects that are ON divided by the total number of objects.

String objects:
The value of the object will contain one line of text for each object that is ON, containing the name of the object.

You can also provide an SCL program to do your own handling. See *Programs for Sources, Checksums, and SABus Response Data* in the *SCL Language Reference* for details.

Use summary source for objects whose value is determined from the value of one or more other objects.

The Summary Source Dialog
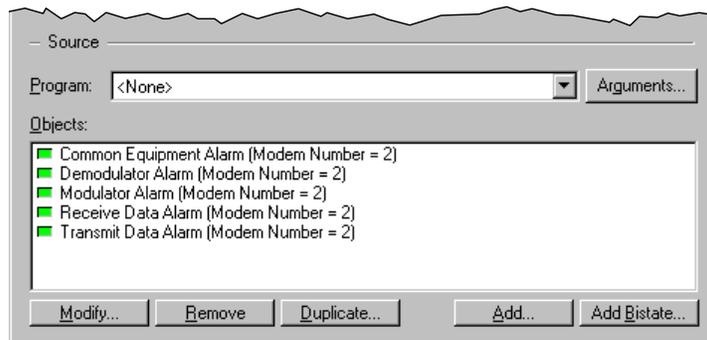
- ▪ Program:

Select the SCL program that is to do the evaluation. Select <none> to use the default implementation. The program arguments are shown in parentheses after the name of the program, but you only select the program from the lists, not the arguments. To change the arguments, use the "Arguments..." button. See *Specifying Arguments for SCL Programs* in the *Developer's Manual* for a description of the Edit Program Arguments dialog.

- ▪ Objects:

Shows a list of all objects that this object summarizes. The object will be updated whenever one of the objects it depends on changes value.

Use the buttons to add, modify, duplicate, and remove object references. The "Add" and "Add Bistate..." button are both used to add objects to the list. If you use the "Add Bistate..." button, you will be asked to select the object from a list that shows only bistate objects.

The "Add Object" dialog allows you to select the object you would like to add, as well as values for all of its parameters. If you want the object to inherit one of the summary object's parameters, leave the parameter value field blank. See *Parameter Value Inheritance* on page 10 for details.

If you do not specify a value for a parameter that is not inherited, a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Parameter Sources

Parameter sources do not get the object's value from anywhere. The value of objects with parameter sources must be set via SCL programs.

Parameter sources support a default value. The object will be set to the default value when the station is opened. If you do not specify a default value, the object will remain in its error state until you set its value from within an SCL program.

Parameter sources can also provide the facility to remember the value of the object across launches of U.P.M.A.C.S.. If you enable this feature, U.P.M.A.C.S. will remember the value the object had when the station is closed, and set it to the same value when it is reopened. In this case, the default value (if any) is used only the very first time the station is loaded.

Use parameter sources for user settings, or for any other data that you maintain yourself rather than getting it from the equipment.

The Parameter Source Dialog
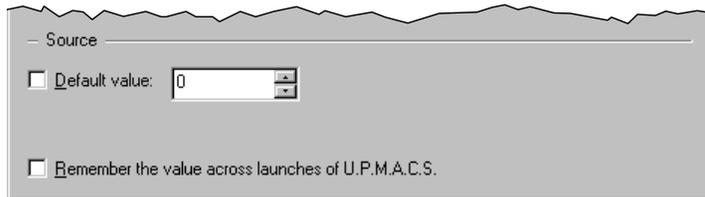
- Default value:

Check this box to specify a default value for the object. Specify or select the default value to the right.

Show as text / Show as hex:

For string objects, you can select the way you want to enter the default value here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

- Remember the value across launches of U.P.M.A.C.S.:

Check this box to store the object's value when the station is closed, and restore it when it is re-opened.

On/Off String Sources

On/Off string sources extract the value of a bistate object from a response using two regular expressions. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

Use on/off string sources if the equipment sends one data string if a setting or alarm is on, and another if it is off.

The On/Off String On/Off String Source Dialog
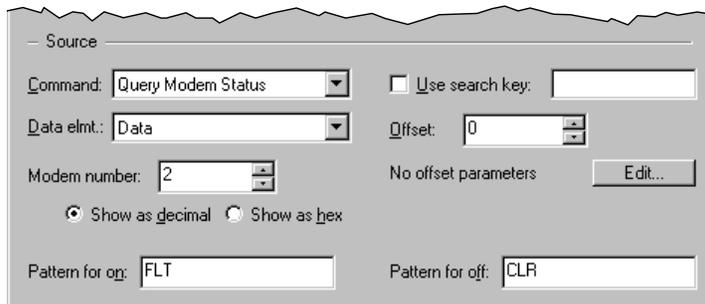
The list below describes only those fields that are specific to the On/Off String Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

- Pattern for on:

Enter a regular expression for the ON value. The object's value will be set to ON if this expression is found at the start of the data.

- Pattern for off:

Enter a regular expression for the OFF value. The object's value will be set to OFF if this expression is found at the start of the data.

Bit Mask Sources

Bit mask sources use one or more bits from a response to determine the value of a bistate object.
The state is calculated as follows:

> ➢ A number is extracted from a data buffer
> ➢ The number is XORed with an XOR mask
> ➢ The number is ANDed with an AND mask

The object is set to OFF if the result is 0, and to ON if the result is not zero, or vice versa (depending on the polarity).

The XOR mask inverts all the bits in the number that are 1 in the XOR mask. The AND mask sets all bits that are 0 in the AND mask to 0. To look at a number of bits do the following:

> ➢ Set all the bits that are of interest to you in the AND mask
> ➢ Set all the bits that need to be inverted (1s become 0s and 0s become 1s) in the XOR mask

```
Example:

    number (in binary):   0 0 1 0 0 0 1 0   0 0 1 1 0 1 1 0   0 0 0 1 1 0 1 1   1 1 0 1 1 0 0 1
    apply XOR mask:       0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 1 0 1 0
    result:               0 0 1 0 0 0 1 0   0 0 1 1 0 1 1 0   0 0 0 1 1 0 1 1   1 1 0 1 0 0 1 1
    apply AND mask:       0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 1 1 1 1
    result:               0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0 1 1

The result is 00000011 00000000 00000000 00000000= 50331648.
```

Use bit mask sources for bistate objects whose state depends on the value of one or more bits in a response.

Using Rotate Parameters

Bit rotation explained:

A bit rotation left moves all bits in a value left. The topmost bits are moved to the bottom, into the spot left vacant when the bottom bits moved left.

A bit rotation right moves all bits in a value right. The bottom most bits are moved to the top, into the spot left vacant when the top bits moved right.

A bit rotation left by 3 looks like this:

You can use one or more of the digital parameters of the data objects to calculate a bit rotation for the value. You can specify a formula to be used to calculate the bit rotation from the parameter value. The formula has the following form:

bits rotated = (parameter value + offset) · factor

You can specify the direction of the rotation (left or right)

Example:

You are creating an object that gets the alarm state of a modem, and all the information about all the modems appears in a single byte in a response. The alarm state information for modem number 1 is bit 0, the state of modem 2 at bit 1, etc. You should then use the modem number as a rotate parameter, rotating right with offset - 1 and factor 1. The number of bits to rotate the value to the right for the different modems will then be calculated as follows:

number of bits to rotate = (modem number - 1) · 1

This gives the following offsets for the first four modems:

➢   Modem number 1: (1 - 1) · 1 = 0
➢   Modem number 2: (2 - 1) · 1 = 1
➢   Modem number 3: (3 - 1) · 1 = 2
➢   Modem number 4: (4 - 1) · 1 = 3

The response byte will thus be rotated for each modem until the alarm state bit is bit 0. You can then set bit 0 in the AND mask to retrieve the alarm state.

For modem 4, the rotation value will have the following effect:

number (in binary):	0 0 1 0 0 0 1 0	0 0 1 1 0 1 1 0	0 0 0 1 1 0 1 1	1 1 0 1 1 0 0 1
rotate right by 3 bits:	1 0 0 1 0 0 0 1	0 0 0 1 1 0 1 1	0 0 0 0 1 1 0 1	1 1 1 0 1 1 0 0
	0 1 0 0 1 0 0 0	1 0 0 0 1 1 0 1	1 0 0 0 0 1 1 0	1 1 1 1 0 1 1 0
	0 0 1 0 0 1 0 0	0 1 0 0 0 1 1 0	1 1 0 0 0 0 1 1	0 1 1 1 1 0 1 1
apply AND mask:	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1
result:	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1

▪ The Edit Rotate Parameters and Edit Rotate Parameter Dialogs
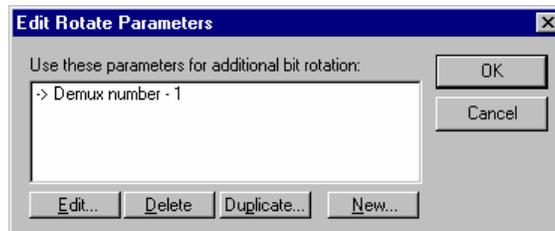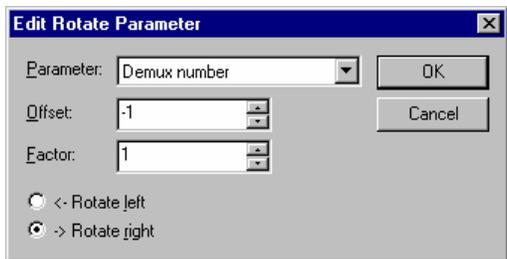
Use these parameters for additional bit rotation:
Shows a list of all the formulas used to calculate the bit rotation values. Use the buttons to edit, delete, duplicate, or add formulas.

Parameter:
Select the parameter whose value should be used to calculate rotation.

Offset:
Enter the offset for the parameter value. The offset is applied before the factor.

Factor:
Enter the factor for the parameter value. The offset is applied after the offset.

Rotate left/Rotate right:
Select the direction of the rotation.

The Bit Mask Source Dialog
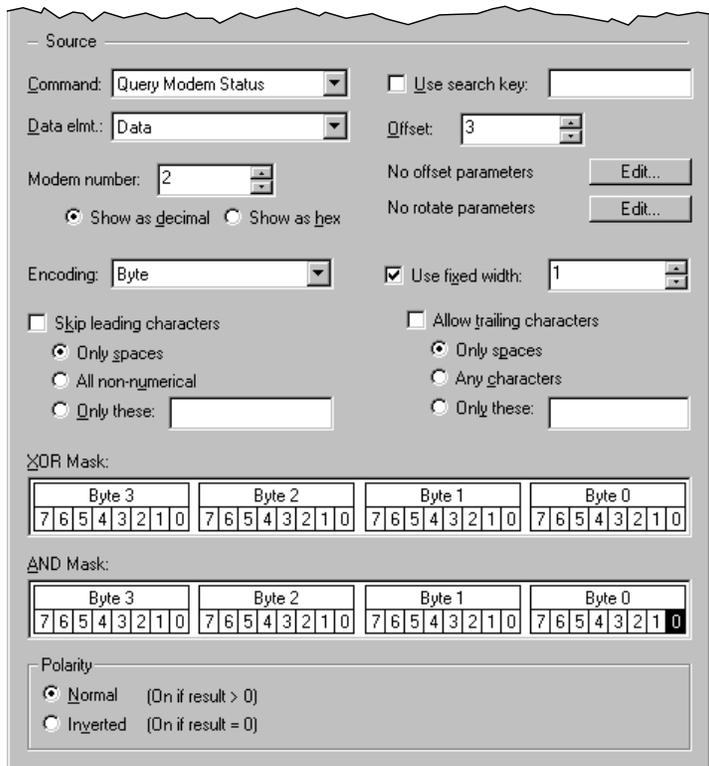
The list below describes only those fields that are specific to the Bit Mask Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

▪ Encoding:

Select the encoding method that the equipment uses for the number. U.P.M.A.C.S. supports the following encoding methods:

Byte:

A single byte (character) in the command string is used to represent the number as an 8-bit value. Requires a fixed width of 1.

Multibyte:

Two or more bytes (characters) in the command string are used to represent the number as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:

The number is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:

The number is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the value to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

▪ Skip leading characters:

Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:

Specify the characters that are allowed to appear before the number.

▪ Use fixed width:

Check this check box to require the number to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:

Check this box to allow characters that are not part of the number to appear after it in the response. This only applies to characters within the fixed width. All characters are always allowed to appear beyond the fixed width. Specify the characters that are allowed to appear after the number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ XOR mask:

Select the bits you want to invert. The least significant bit and byte are shown on the right. Black bits will be inverted; white bits will not be inverted. Click on a bit to toggle it, click on the byte number above the bits to set or clear all the bits in that byte.

- AND mask:

Select the bits you want to use. The least significant bit and byte are shown on the right. Black bits will be used; white bits will be masked out to 0s. Click on a bit to toggle it, click on the byte number above the bits to set or clear all the bits in that byte.

- Polarity:

Select the polarity of the object. The object can either be set to ON if the result is greater than 0 (normal polarity), or if the result is equal to 0 (inverted polarity).

Search String Sources

Search string sources determine the value of a bistate object by searching for a regular expression in the response data or not. The value of the object depends on whether the expression was found or not. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

Use search string sources to implement alarms that are triggered or cleared when data sent by a piece of equipment contains certain keywords. This is useful, for example, for equipment that returns a string listing all currently active fault conditions.

The Search String Source Dialog
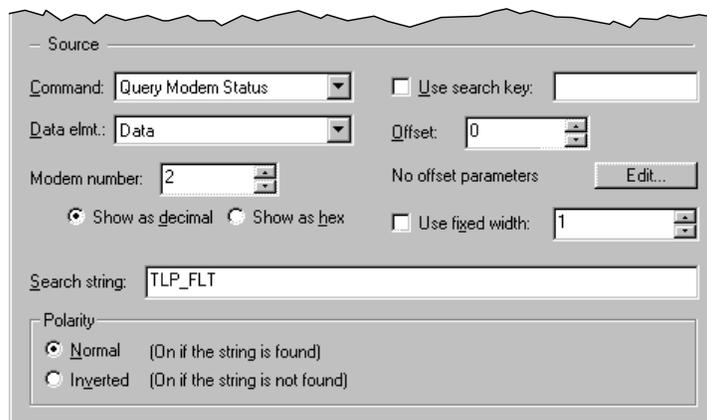
The list below describes only those fields that are specific to the Search String Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

- Use fixed width:

Check this box to limit the search to only a section of the response data. Enter the width of the section in the edit field.

- Search string:

Enter the regular expression to search for.

- Polarity:

Select the polarity of the object. The object can either be set to ON if the regular expression is found (normal polarity), or if it is not found (inverted polarity).

Digital Value List Sources

Digital value list sources determine the value of a bistate object by checking the value of a digital object against a list of values. The bistate object can be set to ON if the value of the digital object appears in the list, and OFF if it doesn't, or vice-versa.

Use digital value list sources if some of the values of a digital object are special in some way. For example, you can use a digital value list source to trigger an alarm for certain values of a digital object.

The Digital Value List Source Dialog
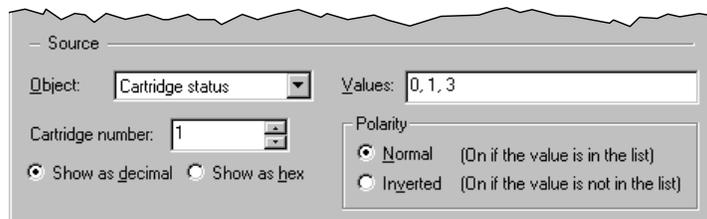
- **Object**:
Select the digital object whose value you would like to use. Specify values for all the object's parameters immediately below. In the sample dialog, the object has one digital parameter called "Cartridge number".

If you want the digital object to inherit one of the object's parameters, leave the parameter value field blank. See *Parameter Value Inheritance* on page 10 for details. If you do not specify a value for a parameter that is not inherited, a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Show as decimal / Show as hex:
If the object has any parameters of type digital that don't have value names, you can select the way you want to enter the parameter values here. Select "Show as decimal" to enter the values in decimal, select "Show as hex" to enter the values in hexadecimal.

Show as text / Show as hex:
If the object has any parameters of type string, you can select the way you want to enter the parameter values here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

- **Values**:
Enter the list of relevant values here. Separate the different values by commas.

- **Polarity**:
Select the polarity of the object. The object can either be set to ON if the value appears in the list (normal polarity), or if it does not appear in the list (inverted polarity).

Value Limits Sources

Value limit sources determine the value of a bistate object based on whether the value of a digital or analog object lies within certain limits or not.

Use value limit sources for trip limit alarms.

The Value Limits Source Dialog
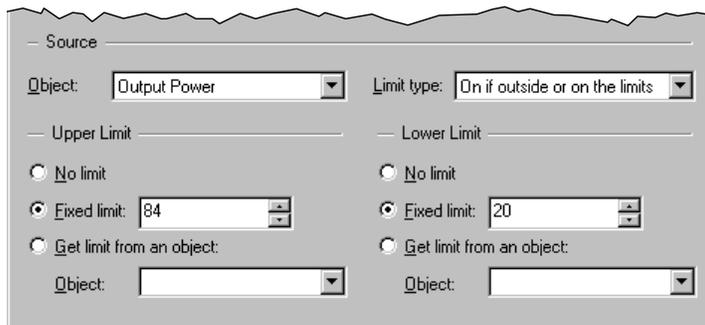
- Object:

Select the digital or analog object whose value you would like to check. Specify values for all the object's parameters immediately below.

If you want the digital or analog object to inherit one of the object's parameters, leave the parameter value field blank. See *Parameter Value Inheritance* on page 10 for details. If you do not specify a value for a parameter that is not inherited, a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Show as decimal / Show as hex:

If the object has any parameters of type digital that don't have value names, you can select the way you want to enter the parameter values here. Select "Show as decimal" to enter the values in decimal, select "Show as hex" to enter the values in hexadecimal.

Show as text / Show as hex:

If the object has any parameters of type string, you can select the way you want to enter the parameter values here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

- Limit type:

Specify the conditions under which the object should be set to on. You can select one of the following options:

On if within or on the limits

The object will be set to the ON state if $\boxed{\text{upper limit} \geq \text{value}}$ and $\boxed{\text{value} \geq \text{lower limit}}$.

On if strictly within the limits

The object will be set to the ON state if $\boxed{\text{upper limit} > \text{value}}$ and $\boxed{\text{value} > \text{lower limit}}$.

On if outside or on the limits

The object will be set to the ON state if $\boxed{\text{value} \geq \text{upper limit}}$ or $\boxed{\text{lower limit} \geq \text{value}}$.

On if strictly outside the limits

The object will be set to the ON state if $\boxed{\text{value} > \text{upper limit}}$ or $\boxed{\text{lower limit} > \text{value}}$.

- No limit / Fixed limit / Get limit from an object:

Select the type of limit to use for the upper or lower limit. There are three types of limits:

No limit:

The value is not limited in this direction.

Fixed limit:
The limit is a fixed number. Enter the limit in the edit field.

Get limit from an object:
Use the value of another object as the limit. The limit object must be of the same type (digital or analog) as the main object. You can disable the limit by masking the limit object. If the limit object is masked, the limit is treated as if you selected the "No limit" option.

Select the object whose value you would like to use as the limit. Specify values for all the object's parameters immediately below.

If you want the limit object to inherit one of the object's parameters, leave the parameter value field blank. If you do not specify a value for a parameter that is not inherited, a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Show as decimal / Show as hex:
If the object has any parameters of type digital that don't have value names, you can select the way you want to enter the parameter values here. Select "Show as decimal" to enter the values in decimal, select "Show as hex" to enter the values in hexadecimal.

Show as text / Show as hex:
If the object has any parameters of type string, you can select the way you want to enter the parameter values here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

Pattern Match Sources

Pattern match sources determine the value of a bistate object by seeing if the value of a string object matches a regular expression. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

Use pattern match sources if some of the values of a string object are special in some way. For example, you can use a pattern match source to trigger an alarm for certain values of a string object.

The Pattern Match Source Dialog

- Object:
Select the digital object whose value you would like to use. Specify values for all the object's parameters immediately below.

If you want the digital object to inherit one of the object's parameters, leave the parameter value field blank. See *Parameter Value Inheritance* on page 10 for details. If you do not specify a value for a parameter that is not inherited, a default value will be used. The default value for bistate parameters is OFF, the default value for digital and analog parameters is 0, and the default value for string parameters is an empty string.

Show as decimal / Show as hex:
If the object has any parameters of type digital that don't have value names, you can select the way you want to enter the parameter values here. Select "Show as decimal" to enter the values in decimal, select "Show as hex" to enter the values in hexadecimal.

Show as text / Show as hex:
If the object has any parameters of type string, you can select the way you want to enter the parameter values here. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

▪ Pattern:
Enter the regular expression that the object's value must match.

▪ Polarity:
Select the polarity of the object. The object can either be set to ON if the value matches the pattern (normal polarity), or if it does not match the pattern (inverted polarity).

Digital Number Sources

Digital number sources get the value of a digital object directly from a response.

Use digital number sources for digital objects whose value appears as-is in a response.

The Digital Number Source Dialog
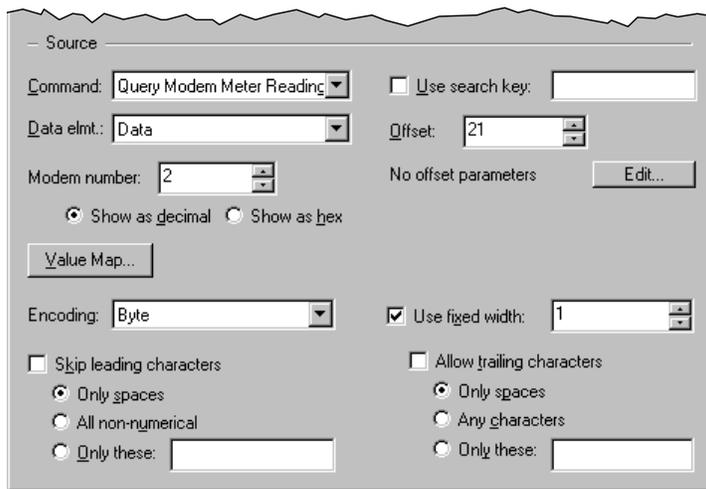
The list below describes only those fields that are specific to the Digital Number Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

▪ Encoding:
Select the encoding method that the equipment uses for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the value as an 8-bit value. Requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the value as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte

ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The value is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The value is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the value to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

▪ **Skip leading characters**:
Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

▪ **Use fixed width**:
Check this check box to require the number to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the response. This only applies to characters within the fixed width. All characters are always allowed to appear beyond the fixed width. Specify the characters that are allowed to appear after the number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Set of Strings Sources

Set of strings sources extract the value of a digital object from the response using a set of regular expressions. See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

Use set of strings sources if the equipment sends different data strings representing different states or settings.

The Set of Strings Source Dialog
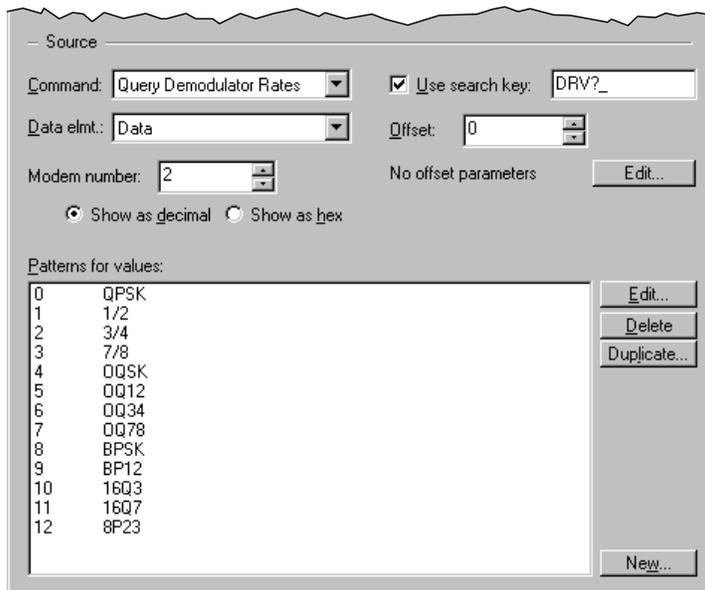
The list below describes only those fields that are specific to the Set of Strings Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

▪ **Patterns for values**:
Shows a list of all values that you specified regular expressions for, together with the expression that describes the data string for that value. Use the buttons to edit, delete, duplicate, and add values.

Thresholds Sources

Thresholds sources set the value of a digital object according to a number from a data buffer and a set of thresholds. You can specify a value that the object should take if the number is a above a certain threshold. You must also specify a bottom value for the object. The bottom value is the value the object should have if the number lies below all the thresholds.

Example:

The sample dialog shows the following thresholds and values:

Bottom value: 0
Threshold at 85:   1
Threshold at 170:   2

The object will have value 0 for numbers less than 85, value 1 for numbers of at least 85 but less than 170, and value 2 for numbers of at least 170.

Use thresholds sources for objects whose value reflects a set of states represented as different analog levels on an input of a data acquisition unit.

The Thresholds Source Dialog
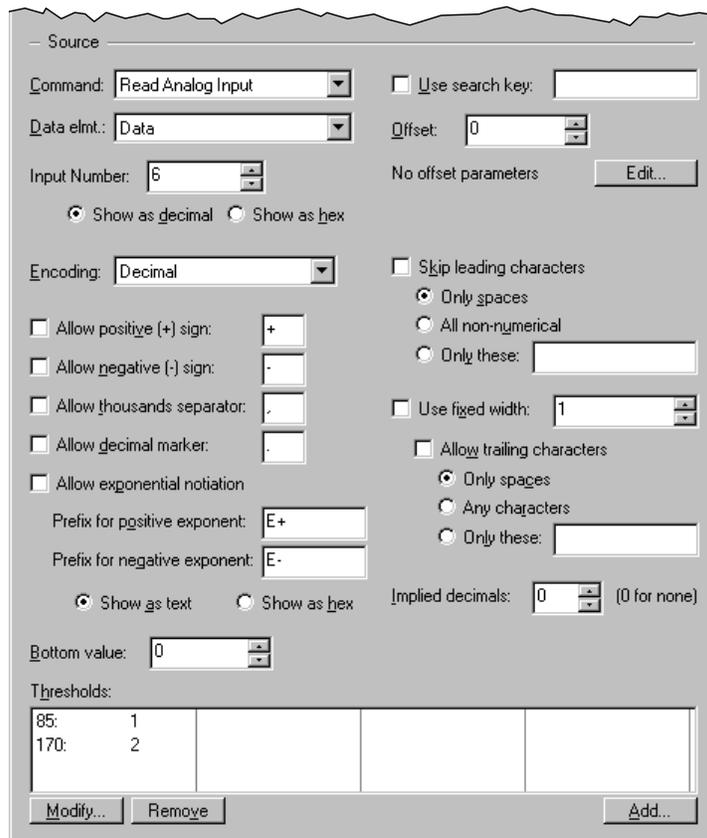
The list below describes only those fields that are specific to the Threshold Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

- Encoding:
Select the encoding method that the equipment uses for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:
A single byte (character) in the command string is used to represent the value as an 8-bit value. If you select signed byte encoding, the most significant bit of the byte will be interpreted as the sign in the standard way ($FF is -1, $FE is -2, etc.). The byte encoding method requires a fixed width of 1.

Multibyte:
Two or more bytes (characters) in the command string are used to represent the value as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you select hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. If you select signed multibyte encoding, the most significant bit of the most significant byte will be interpreted as the sign in the standard way (for 16-bit values $FFFF is -1,$FFFE is -2, etc.). The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:
The value is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the

byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encoding method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:
The value is written out as a decimal, hexadecimal, binary, or octal number using ASCII characters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the value to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

- Allow positive (+) sign:
Check this box to allow a plus sign to denote positive numbers. The plus sign does not actually have to be the "+" character. You can specify any character you want.

This setting only applies to the decimal encoding.

- Allow negative (–) sign:
Check this box to allow a minus sign to denote negative numbers. If you leave this check box blank, all numbers will be positive. The minus sign does not actually have to be the "-" character. You can specify any character you want.

This setting only applies to the decimal encoding.

- Allow thousands separator:
Check this check box if you want digits to be separated into groups of three using a thousands separators (usually a comma). If you check this box, you must also specify the actual character used to group digits.

This setting only applies to the decimal encoding.

- Allow decimal marker:
Check this box to allow a decimal marker and decimals. If you leave this check box blank, all numbers will be whole numbers. If you leave this box checked, you can specify the character used as a decimal marker.

This setting only applies to the decimal encoding.

- Allow exponential notation:
Check this box to allow exponential (scientific) notation with the decimal encoding method.

Prefix for positive / negative exponent:
Enter the exponent markers for positive and negative exponents here. The exponent markers must include the sign of the exponent. Usually, the positive exponent marker is "E+" or "e+", and the negative marker is "E-" or "e-". If the plus sign is omitted for positive exponents, specify "E" or "e" for the positive exponent prefix. The exponent prefixes can be any arbitrary data. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

Show as text / Show as hex:
Select the way you want to enter the exponent prefixes. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details.

- Skip leading characters:
Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

▪ Use fixed width:
Check this check box to require the number to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the response. This only applies to characters within the fixed width. All characters are always allowed to appear beyond the fixed width. Specify the characters that are allowed to appear after the number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

▪ Implied decimals:
Specify the number of implied decimals here. Implied decimals are a way of encoding fractions without using a decimal point by multiplying it with a power of ten before encoding it. A number that is encoded using 3 implied decimals, for example, will be multiplied by 1000 before it is encoded. This will move the three decimals from the right to the left of the decimal point, and the decimal point will no longer be needed. 21.304, for example, will simply be encoded as 21304.

Please note that the implied decimals for the hexadecimal, octal, and binary encoding methods are decimal fractions, not hexadecimal, octal or binary fractions. In hexadecimal encoding with three implied decimals, 43.249 (which is ~2B.3FC in hex) will appear as A8F1 (which is 43249 in decimal) rather than 2B3FC. In other words, the number is always multiplied by powers of 10, even if the encoding method uses base 16, 2 , or 8.

▪ Bottom value:
Enter the value that the object should take if the number lies below any of the thresholds.

▪ Thresholds:
Lists all the thresholds and their corresponding values. Use the buttons to add, remove, or modify thresholds and their values.


Bit Collection Sources

Bit collection sources use bits from one or more responses to determine the value of a digital object. The bits are taken from one or more bit sections, and the values of those sections are ORed together. You can then invert bits in the result using an XOR mask.

The value of each section is determined as follows:

➢ A number is extracted from a data buffer

➢ The number is bit rotated to adjust the position of the relevant bits

➢ Any bits that need to be inverted are inverted using an XOR mask

➢ The relevant bits are masked out using an AND mask

All the sections are then ORed together, and the XOR mask of the bit collection source is applied.

Bit rotation explained:

A bit rotation left moves all bits in a value left. The topmost bits are moved to the bottom, into the spot left vacant when the bottom bits moved left.

A bit rotation right moves all bits in a value right. The bottom most bits are moved to the top, into the spot left vacant when the top bits moved right.

A bit rotation left by 3 looks like this:
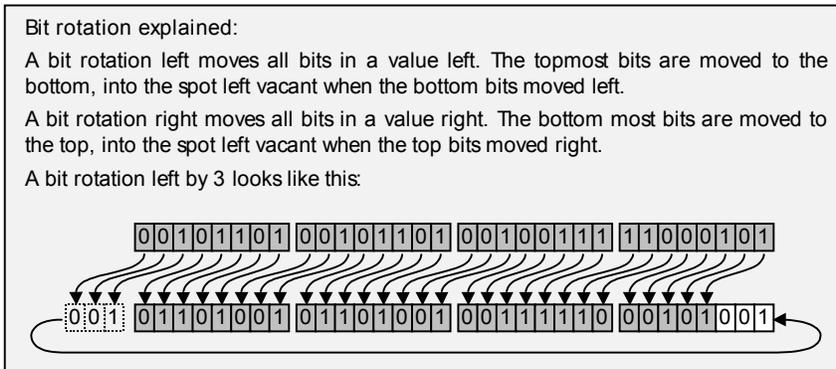
You can also specify a value map for the source. Sometimes, it is not possible to manipulate the bits so that you will get the desired values for the desired states. Other times, different results correspond to the same value. A value map will allow you to substitute other values for values calculated by the source. Normally, the result of the calculation described above is used directly as the value of the object. If the result is 2, the value of the object will also be 2. If, however, you would prefer the value of the object to be 4 if the result is 2, then you could specify a value map that maps a source value of 2 to an object value of 4. You can specify object values to be substituted for any number of source values.

Use bit collection sources for digital objects whose value depends on a number of bits from a number of responses.

Examples

There are two basic ways of using bit collection sources.

▪ Using a number of adjacent bits from a single byte:

Sometimes, the state of a piece of equipment is represented by a number of bits in a status byte. Bits 3 and 4 in a byte might represent a local / computer / remote front panel mode, for example. The two bits might be 00, 01, and 10, depending on the mode. To turn this into a number between 0 and 2, construct one bit section as follows:

number (in binary):	1 1 0 0 1 0 0 1
rotate right by 3 bits:	1 1 1 0 0 1 0 0
	0 1 1 1 0 0 1 0
	0 0 1 1 1 0 0 1
apply XOR mask:	0 0 0 0 0 0 0 0
result:	0 0 1 1 1 0 0 1
apply AND mask:	0 0 0 0 0 0 1 1
result:	0 0 0 0 0 0 0 1

The result of this section is then treated with the global XOR:

result of the section:	0 0 0 0 0 0 0 1
apply global XOR:	0 0 0 0 0 0 0 0
result:	0 0 0 0 0 0 0 1

The value of the object will thus depend on the value of the two bits alone.

▪ Using bits from different data buffers, or using non-adjacent bits:

Sometimes, the bits that represent the value you desire are not right next to each other in the same buffer. You might want to turn a standby / transmit state and a warming / ready state of an HPA into a single object, but the states are represented by different bits in different responses. Let's assume the high bit is bit 0 in one response, and the low bit is bit 2 in another response. You will need two bit sections, as follows:

The first section turns bit 0 of one response into bit 1 of the result:

|	
number (in binary):	0 1 1 0 1 0 1 1
rotate left by 1 bit:	1 1 0 1 0 1 1 0
apply XOR mask:	0 0 0 0 0 0 0 0
result:	1 1 0 1 0 1 1 0
apply AND mask:	0 0 0 0 0 0 1 0
result:	0 0 0 0 0 0 1 0

The second section turns bit 2 of the other response into bit 0 of the result:

number (in binary):	1 1 0 1 1 0 0 1
rotate right by 2 bits:	1 1 1 0 1 1 0 0
	0 1 1 1 0 1 1 0
apply XOR mask:	0 0 0 0 0 0 0 0
result:	0 1 1 1 0 1 1 0
apply AND mask:	0 0 0 0 0 0 0 1
result:	0 0 0 0 0 0 0 0

The result of the sections are ORed together and then treated with the global XOR:

result section 1:	0 0 0 0 0 0 1 0
result section 2:	0 0 0 0 0 0 0 0
ORed together:	0 0 0 0 0 0 1 0
apply global XOR:	0 0 0 0 0 0 0 0
result:	0 0 0 0 0 1 0

The Bit Collection Source Dialog
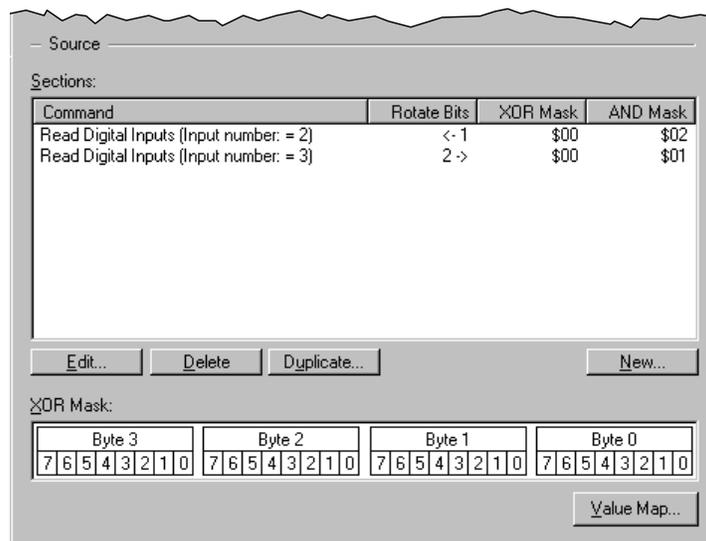
▪ **Sections**:

Lists the commands, rotate values, and XOR and AND masks of all the sections. Use the buttons to create, delete, or edit sections. Use the "Duplicate..." button to duplicate the selected section.

See *Bit Sections* below for a description of the New Bit Section dialog.

▪ **XOR mask**:

Select the bits you want to invert. The least significant bit and byte are shown on the right. Black bits will be inverted; white bits will not be inverted. Click on a bit to toggle it, click on the byte number above the bits to set or clear all the bits in that byte.

▪ **The "Value Map" button**:

Click this button to edit the value map. The value map allows you to substitute different values for the values calculated by the source.

Bit Sections

Bit sections are used in bit collection sources. A bit collection source uses the results of one or more bit sections, ORed together.

The result of a bit section is calculated as follows:

➢ A number is extracted from a data buffer

➢ The number is bit rotated to adjust the position of the relevant bits

➢ Any bits that need to be inverted are inverted using an XOR mask

➢ The relevant bits are masked out using an AND mask

The XOR mask inverts all the bits in the number that are 1 in the XOR mask. The AND mask sets all bits that are 0 in the AND mask to 0. To look at a number of bits do the following:

➢ Bit rotate the number left or right to move the bits to the desired location within the result

> ➢ Set all the bits that are of interested to you in the AND mask
> ➢ Set all the bits that need to be inverted (1s become 0s and 0s become 1s) in the XOR mask

Example:

number (in binary):	1 1 0 1 1 1 1 0	1 1 0 0 1 0 0 1	0 0 0 1 0 0 0 1	1 0 1 1 0 0 0 0
rotate left by 5 bits:	1 0 1 1 1 1 0 1	1 0 0 1 0 0 1 0	0 0 1 0 0 0 1 1	0 1 1 0 0 0 0 1
	0 1 1 1 1 0 1 1	0 0 1 0 0 1 0 0	0 1 0 0 0 1 1 0	1 1 0 0 0 0 1 1
	1 1 1 1 0 1 1 0	0 1 0 0 1 0 0 0	1 0 0 0 1 1 0 1	1 0 0 0 0 1 1 0
	1 1 1 0 1 1 0 0	1 0 0 1 0 0 0 1	0 0 0 1 1 0 1 1	0 0 0 0 1 1 0 1
	1 1 0 1 1 0 0 1	0 0 1 0 0 0 1 0	0 0 1 1 0 1 1 0	0 0 0 1 1 0 1 1
apply XOR mask:	0 0 0 0 1 0 1 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
result:	1 1 0 1 0 0 1 1	0 0 1 0 0 0 1 0	0 0 1 1 0 1 1 0	0 0 0 1 1 0 1 1
apply AND mask:	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
result:	0 0 0 0 0 0 1 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

The result is 00000011 00000000 00000000 00000000= 50331648.

Using Rotate Parameters

You can use one or more of the digital parameters of the data objects to calculate an additional bit rotation for the value. You can specify a formula to be used to calculate the bit rotation from the parameter value. The formula has the following form:

bits rotated = (parameter value + offset) · factor

You can specify the direction of the rotation (left or right)

Example:

You are creating an object that gets the transmit / standby / warming state of an HPA, and all the information about all four HPAs appear in a single byte in a response. The transmit state information for HPA 1 is contained in bits 0 and 1, the state for HPA 2 in bit 2 and 3, etc. You should then specify use the HPA number as a rotate parameter, rotating right with offset -1 and factor 2. The number of bits to rotate the value to the right for the different HPAs will then be calculated as follows:

number of bits to rotate = (HPA number - 1) · 2

This gives the following offsets for the HPAs:

➢ HPA 1: (1 - 1) · 2 = 0
➢ HPA 2: (2 - 1) · 2 = 2
➢ HPA 3: (3 - 1) · 2 = 4
➢ HPA 4: (4 - 1) · 2 = 6

The response byte will thus be rotated for each HPA until the state bits are bits 0 and 1. You can then set bits 0 and 1 in the AND mask to retrieve the state.

For HPA 3, the rotation value will have the following effect:

number (in binary):	0 1 0 0 0 1 0 0	0 1 1 0 1 1 0 0	0 0 1 1 0 1 1 1	1 0 1 0 0 0 1 0
rotate right by 4 bits:	0 0 1 0 0 0 1 0	0 0 1 1 0 1 1 0	0 0 0 1 1 0 1 1	1 1 0 1 0 0 0 1
	1 0 0 1 0 0 0 1	0 0 0 1 1 0 1 1	0 0 0 0 1 1 0 1	1 1 1 0 1 0 0 0
	0 1 0 0 1 0 0 0	1 0 0 0 1 1 0 1	1 0 0 0 0 1 1 0	1 1 1 1 0 1 0 0
	0 0 1 0 0 1 0 0	0 1 0 0 0 1 1 0	1 1 0 0 0 0 1 1	0 1 1 1 1 0 1 0
apply AND mask:	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0
result:	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0

▪ The Edit Rotate Parameters and Edit Rotate Parameter Dialogs

Click here to see the Edit Rotate Parameters dialog and the Edit Rotate Parameter dialog.
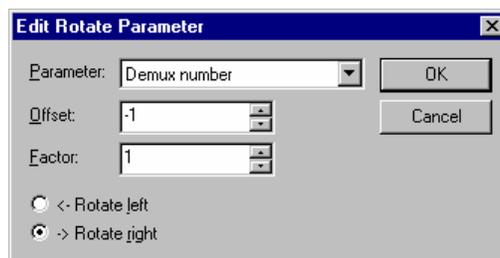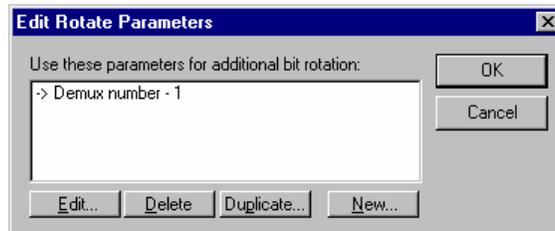
Use these parameters for additional bit rotation:
Shows a list of all the formulas used to calculate the additional bit rotation values. use the buttons to edit, delete, duplicate, or add formulas.

Parameter:
Select the parameter whose value should be used to calculate rotation.

Offset:
Enter the offset for the parameter value. The offset is applied before the factor.

Factor:
Enter the factor for the parameter value. The offset is applied after the offset.

Rotate left/Rotate right:
Select the direction of the rotation.

The New Bit Section Dialog
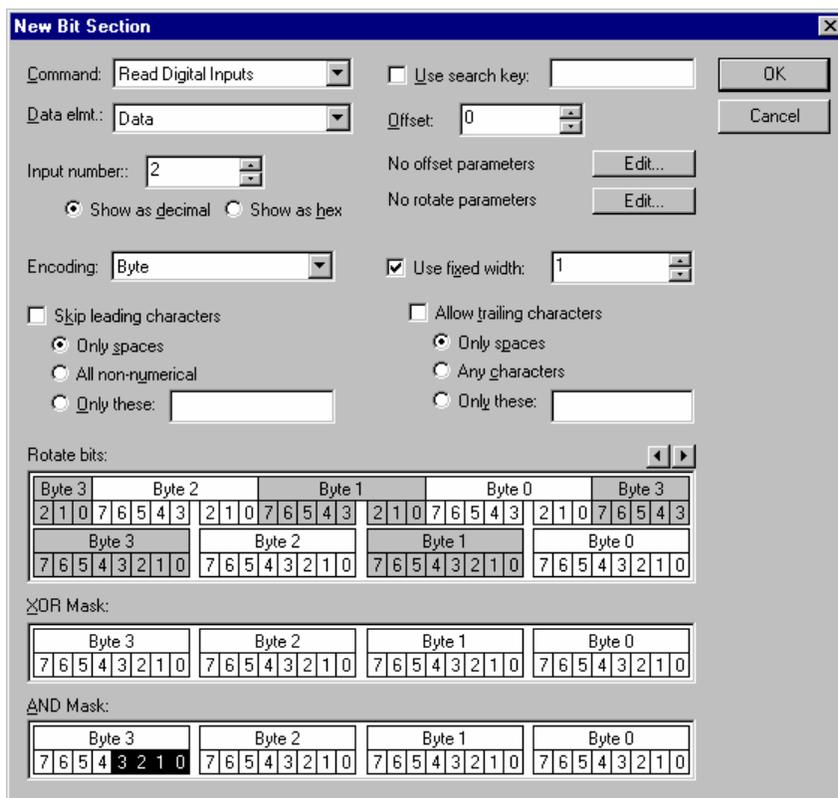
The list below describes only those fields that are specific to the New Bit Section dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

▪ Rotate bits:

Specify the number of bits to rotate to the left or right. Press the ◀ ▶ arrows on the top right until the bits align the way you would like them.

▪ XOR mask:

Select the bits you want to invert. The least significant bit and byte are shown on the right. Black bits will be inverted; white bits will not be inverted. Click on a bit to toggle it, click on the byte number above the bits to set or clear all the bits in that byte. The masks are applied after the bit rotation.

▪ AND mask:

Select the bits you want to use. The least significant bit and byte are shown on the right. Black bits will be used; white bits will be masked out to 0s. Click on a bit to toggle it, click on the byte number above the bits to set or clear all the bits in that byte. The masks are applied after the bit rotation.

Analog Number Sources

Analog number sources get the value of an analog object directly from a response. You can provide a factor and an offset for the value. The value of the object is calculated from the number extracted from the response as follows:

value = number · factor + offset

To use the number unaltered, specify a factor of 1 and an offset of 0.

Analog number sources also support greater than (">") and less than ("<") symbols in the response. Some equipment sends data like "<1.0E-20dB" if a value is too small or too large to be measured. Analog number sources can recognize the greater or less than symbol. The correct symbol will then be displayed by any indicators whose registers use that object.

Use analog number sources for analog objects whose value or values appears as-is in a response.

The Analog Number Sources Dialog
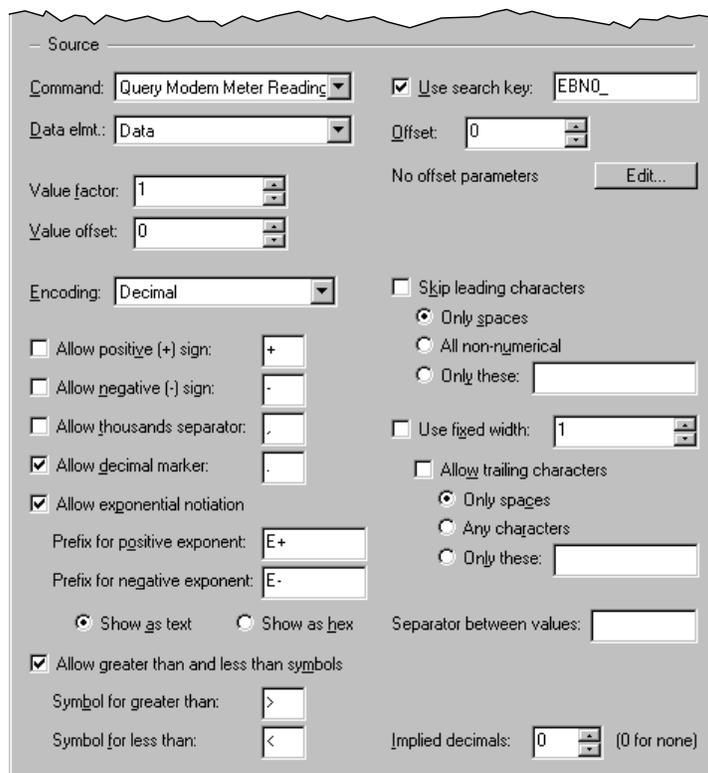
The list below describes only those fields that are specific to the Analog Number Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

▪ Value factor:

Enter the factor with which the number from the response is to be multiplied before writing it to the object. The value factor is applied before the value offset.

▪ Value offset:

Enter the offset that is to be added to the number from the response before writing it to the object. The value offset is applied after the value factor.

▪ Encoding:

Select the encoding method that the equipment uses for the value. U.P.M.A.C.S. supports the following encoding methods:

Byte:

A single byte (character) in the command string is used to represent the value as an 8-bit value. If you select signed byte encoding, the most significant bit of the byte will be interpreted as the sign in the standard way ($FF is -1, $FE is -2, etc.). The byte encoding method requires a fixed width of 1.

Multibyte:

Two or more bytes (characters) in the command string are used to represent the value as a 16, 24, or 32-bit value. You can choose between lo-hi and hi-lo byte ordering. If you select lo-hi byte ordering, the least significant byte (the lower 8 bits) must appear in the response first; if you se-lect hi-lo byte ordering, the most significant byte (the upper 8 bits) must appear first. If you select signed multibyte encoding, the most significant bit of the most significant byte will be interpreted as the sign in the standard way (for 16-bit values $FFFF is -1,$FFFE is -2, etc.). The multibyte encoding method requires a fixed width of 2, 3, or 4.

BCD:

The value is encoded using Binary Coded Decimal encoding. In BCD encoding, each nibble (hex digit) in a byte represents one decimal digit. The number 20,841,057 would be encoded as the byte (character) values hex 20 (32), hex 84 (132), hex 10 (16), and hex 57 (87). The BCD encod-ing method requires a fixed width.

Decimal, Hexadecimal, Binary, Octal:

The value is written out as a decimal, hexadecimal, binary, or octal number using ASCII charac-ters. Hexadecimal encoding recognizes both capital and small letters ("A" to "F" and "a" to "f") as hex digits.

Do not confuse the byte/multibyte and binary encoding methods. The binary encoding expects the value to be written out as a series of ASCII characters 1 (hex 31) and 0 (hex 30), not using the individual bits of each byte.

▪ Allow positive (+) sign:

Check this box to allow a plus sign to denote positive numbers. The plus sign does not actually have to be the "+" character. You can specify any character you want.

This setting only applies to the decimal encoding.

▪ Allow negative (–) sign:

Check this box to allow a minus sign to denote negative numbers. If you leave this check box blank, all numbers will be positive. The minus sign does not actually have to be the "-" character. You can specify any character you want.

This setting only applies to the decimal encoding.

▪ Allow thousands separator:

Check this check box if you want digits to be separated into groups of three using a thousands separators (usually a comma). If you check this box, you must also specify the actual character used to group digits.

This setting only applies to the decimal encoding.

- **Allow decimal marker:**

Check this box to allow a decimal marker and decimals. If you leave this check box blank, all numbers will be whole numbers. If you leave this box checked, you can specify the character used as a decimal marker.

This setting only applies to the decimal encoding.

- **Allow exponential notation:**

Check this box to allow exponential (scientific) notation with the decimal encoding method.

Prefix for positive / negative exponent:
Enter the exponent markers for positive and negative exponents here. The exponent markers must include the sign of the exponent. Usually, the positive exponent marker is "E+" or "e+", and the negative marker is "E-" or "e-". If the plus sign is omitted for positive exponents, specify "E" or "e" for the positive exponent prefix. The exponent prefixes can be any arbitrary data. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details on entering binary data.

Show as text / Show as hex:
Select the way you want to enter the exponent prefixes. See *Appendix B: Entering Binary Data* in the *Developer's Manual* for details.

- **Skip leading characters:**

Check this box to skip any characters that appear before the number in the response. This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

Only spaces / All non-numerical / Only these:
Specify the characters that are allowed to appear before the number.

- **Use fixed width:**

Check this check box to require the number to be a fixed number of bytes (characters) long. Specify the number of bytes in the edit box.

You must check this box for the byte, multibyte, and BCD encoding methods.

Allow trailing characters:
Check this box to allow characters that are not part of the number to appear after it in the response. This only applies to characters within the fixed width. All characters are always allowed to appear beyond the fixed width. Specify the characters that are allowed to appear after the number using the radio buttons.

This setting only applies to the decimal, hexadecimal, binary, and octal encodings.

- **Separator between values:**

Enter a regular expression that describes the separator between the values of an object with a size of more than one value. Leave the field blank if the values appear one after the other, without a separator between them. This field is only available if you specify a size of more than one value.

- **Allow greater than and less than symbols:**

Check this box to allow greater than and less than symbols. The symbols must appear before any sign in the response.

Symbol for greater than:
Enter the symbol used for "greater than." The greater than symbol does not have to be the ">" character, you can enter any character you want.

Symbol for greater than:
Enter the symbol used for "less than." The less than symbol does not have to be the "<" character, you can enter any character you want.

▪ Implied decimals:
Specify the number of implied decimals here. Implied decimals are a way of encoding fractions without using a decimal point by multiplying it with a power of ten before encoding it. A number that is encoded using 3 implied decimals, for example, will be multiplied by 1000 before it is encoded. This will move the three decimals from the right to the left of the decimal point, and the decimal point will no longer be needed. 21.304, for example, will simply be encoded as 21304.

Please note that the implied decimals for the hexadecimal, octal, and binary encoding methods are decimal fractions, not hexadecimal, octal or binary fractions. In hexadecimal encoding with three implied decimals, 43.249 (which is ~2B.3FC in hex) will appear as A8F1 (which is 43249 in decimal) rather than 2B3FC. In other words, the number is always multiplied by powers of 10, even if the encoding method uses base 16, 2 , or 8.

String Sources

String sources get the value of a string object directly from a response.

String sources use regular expression patterns to recognize the value. The data is divided into three sections: A prefix, the value, and a suffix. The prefix and suffix are discarded, and only the value is stored.

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

The String Sources Dialog

The list below describes only those fields that are specific to the String Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

▪ Prefix:
Enter a regular expression that describes any data that appears before the relevant data in the response. Any data that matches the prefix will be discarded.

▪ Pattern:
Enter a regular expression that describes the value here.

▪ Suffix:
Enter a regular expression that describes any data that appears after the relevant data in the response. Any data that matches the suffix will be discarded.

Filter Sources

Filter sources extract a string from the response and use it as the value of the string object, if and only if it matches one of a number of regular expression patterns you specify. If the data does not match any of the patterns, the object's value remains unchanged.

Filter sources use regular expression patterns to recognize the string. The data is divided into three sections: A prefix, the string, and a suffix. The prefix and suffix are discarded, and only the string is used.

See *Appendix A: Regular Expressions* in the *Developer's Manual* for details on regular expressions.

Note: Search string sources also attempt to match a set of regular expressions in a response. Unlike search string sources, however, Filter source objects do not go into the error state if none of the strings are found, but retain their last valid value.

Filter sources are used to cache data that matches a certain pattern. Some equipment, for example, logs messages about events in a message queue and allows you to retrieve the messages in order. In such a case, a filter source lets you remember the last message that referred to a certain piece of information, like for example a certain alarm. If the message in the buffer is overwritten with something that does not refer to that particular alarm, the filter source will ignore the new message, and the object will retain the last relevant one. You can thus attach a number of filter sources to the message buffer; each filtering out messages pertaining to a different alarm. The last message about each alarm will then always be available, even if it has been overwritten in the equipment's message buffer. You can then display the message directly, trigger an alarm based on the message using a pattern match source, or use a summary source to process the information further.

The Filter Source Dialog
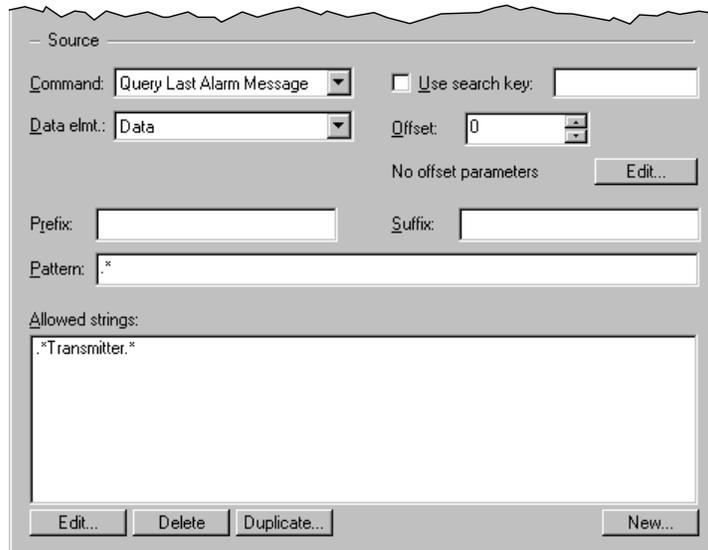
The list below describes only those fields that are specific to the Filter Source dialog. For the remaining fields, see *The Serial Data Source Dialogs* on page 63.

- **Prefix:**
Enter a regular expression that describes any data that appears before the relevant data in the response. Any data that matches the prefix will be discarded.

- Pattern:

Enter a regular expression that describes the string here.

- Suffix:

Enter a regular expression that describes any data that appears after the relevant data in the response. Any data that matches the suffix will be discarded.

- Allowed strings:

Lists regular expressions describing patterns that the data must match. The value of the object will only be changed if the data matches one of the allowed strings.

Use the buttons to edit, delete, duplicate, and add expressions.

Multiple Sources

An object can contain more than one source. Typically, you would use more than one source for an object if the information the object represents appears in the response to more than one command. To use multiple sources with an object, select <Multiple sources> for the source type. The current source will not be deleted, but used as the first of the multiple sources.

Note: You can use a parameter source together with another source type to provide a default value for the object, to remember the object's value across launches of U.P.M.A.C.S., or both. Do not use more than one parameter source for the same object, however, as the resulting behaviour is undefined.

The Multiple Source Dialog
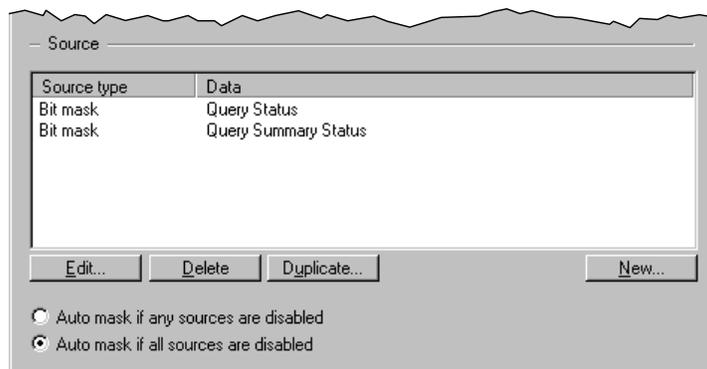
The multiple source dialog lists a list of the types of all the sources, and a brief note on where the data comes from (typically the command whose response is used). Use the buttons to edit, delete, duplicate, or add sources.

- Auto mask if any / all sources are disabled:

Sources automatically mask their object if the command or object(s) they depend on is disabled. If you use multiple sources, you can select whether the object should be auto masked if any of the source's data is disabled, or only if all the sources' data is disabled.

CONTACT INFORMATION

U.P.M.A.C.S. Communications Inc.

6600 Trans Canada Hwy, Suite 501
Pointe-Claire, Québec
Canada
H9R 4S2

Tel: 1-514-697-5500
Toll free: 1-877-697-5500 (Canada & US only)
Fax: 1-514-697-7093
E-Mail: support@upmacs.com

UPMACS is on the Web at http://www.upmacs.com