



WALKER SYSTEMS

building
intelligence

TECHNICAL DESCRIPTION

ASCII Protocol

(PROGRAMMING THE INTERFACE
BETWEEN WALKER AND
EXTERNAL AUTOMATION SYSTEMS)

FORM # WSC99-001A

REVISION 1

CONTENTS

Introduction	1
External System Requirements	2
Preparing the External System	2
The General ASCII String Format	3
Using the String Format	4
Outputting the Walker ASCII Protocol String	8
Programming Walker Products	10
Preparing Walker Products for the Interface	10
The ASCII Protocol Point (APn)	11
Communication Variables (Comvars) and the MESSAGE Command	13
Using Comvars in GCL	15
Interface Example	16
Programming Technique: Overlapping Memory	22

INTRODUCTION

Walker ASCII Protocol is a breakthrough in Walker communications which allows external systems and Walker products to work together with the use of ASCII mnemonics. Walker is now able to meet tough new job specifications and operate with or control products from other vendors. Another key feature, available due to the development of ASCII Protocol, is the Remote Dial In/ DialOut and Database Mapping of two or more remote Walker controllers.

The Walker ASCII Protocol provides a means to port external data into Walker databases. ASCII mnemonics make this protocol universal and very simple to use — most external systems can interface to Walker products with just a few hours of programming.

In general terms, external systems are programmed to convert their existing output signals into simple ASCII characters. These characters are then attached to dimensional information which indicates point location. The resulting ASCII strings are then read by Walker devices, and data is mapped into memory. The strings are flexible and powerful, and are custom built to best suit the application.

This document contains the complete specification for converting external data into ASCII strings to be read by Walker devices. Also included are discussions of the communication variables used by Walker devices to verify data transmission integrity. Application information and examples are provided to assist both external system programmers and Walker technicians.

Part I focuses on the external system requirements and provides the background required for external system programmers, and Part II details how the Walker operator can use the protocol and new menu items to manage the interface. It is recommended, however, that both external system technicians and Walker technicians review this document in its entirety.

EXTERNAL SYSTEM REQUIREMENTS

Application Overview The Walker ASCII Protocol can be applied to most external systems, including:

- access and security
- CCTV monitoring systems
- fire protection systems
- lighting control
- other HVAC and energy management systems

The Protocol involves translating the output from the external system into ASCII strings that map point data into Walker databases. Once the data has been translated and mapped into the Walker product, it becomes part of the Walker system. Data manipulation can then be performed using GCL programming and other Walker resources, and alarm conditions in the external system will also register at the Walker front end. Walker products can then effectively control both systems, or operate as a backup if desired.

A minimum of engineering and programming are required to prepare the external system for connection. Walker developers are available to coordinate engineering efforts and assist with designing the most efficient use of the protocol for a given site. The Walker ASCII Protocol, like the Walker control system, is highly flexible and adaptive — most interface scenarios have several possible implementations. It is advisable to contact Walker Systems Corporation before preparing the external system for connection to the Walker product line

Preparing the External System

The following workflow is recommended to prepare the external system:

1. Analyze the system and determine the number and type of each point that will be interface Document this analysis in terms of point quantity, point location and output type (signal type). Indicate which points are critical and what level of communication integrity must be maintained.
2. Total the outputs from the above analysis. If possible, separate the points into logical groups, based on criteria such as location and output type. Location criteria can be compiled relative to where the Walker panels will be located, if this information is available. Each WS1600 Node Controller and WS1616 SAC can accommodate about 2000 points, but try to make smaller groups if possible. Walker Systems Corporation should be consulted to review the data and advise the best solution for the job.
3. Prepare the ASCII strings for transmission to the Walker product. This is detailed in the following paragraphs.

The General ASCII String Format

The ASCII string contains all of the information required by Walker: the direction of the transmission, the point location and the value (state) of the point. Optional information may be added at the end of the string to check data and communication integrity (the verification suffix is generated at the Walker control side of the interface). The strings are generated and transmitted to the Walker system via serial communications.

The following general string framework is required from an external system

?_Source>Destination_Location_Value_ChecksumCRLF

with the following illustrating a typical string output

?_J>C_Cxx,Zyy,Pzz_Tn_ChksumCRLF

where:

System Parameters

- ?** String beginning. Every string must begin with this mnemonic.
- _** Field delimiter. Each field must be separated with this character.
- ,** Dimensional field delimiter

The letter characters for the Destination and Location fields are arbitrarily chosen by the system programmer to best represent the system parameters (i.e. J for Jim's Door Company, C for Walker controls).

- J>C** Indicates string source to string destination, J>C indicates input from the source J to the destination C. To send information from source C to destination J, the C>J string is used.
- Cxx** Circuit field, xx is the circuit number. The C field creates the first dimension.
- Zyy** Zone field, yy is the zone number. The Z field creates a second dimension.
- Pzz** Panel field, zz is the panel number. P field creates 3rd dimension. The value field must be one of the following 4 choices:
- Tn** T denotes ASCII character to follow, n is the ASCII character.
- En** E denotes decimal value to follow, n is the decimal value.
- Sn** S denotes ASCII character string to follow, n is the character string.
- Fn** F denotes decimal string value to follow, n is the decimal string.

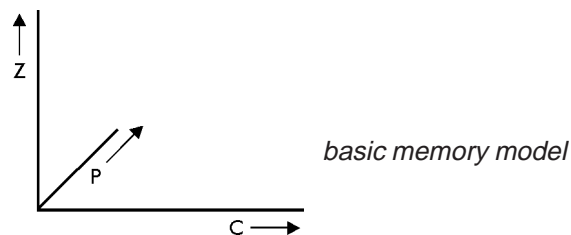
ChksumCRLF ChksumCRLF indicates data checksum followed by a carriage return (CR) and a line feed (LF).

The general string format detailed above uses an arbitrary analogy which separates the incoming data into circuits, zones and panels. Using an analogy such as this reinforces the logical groups made during the engineering of the interface, and provides a memory aid to technicians who will use the protocol. The use of the characters C, Z and P as the dimensional mnemonics is also arbitrary, selected to reflect the Circuit, Zone, Panel scheme. The dimensional fields can represent any logical groups desired, and any letters can be chosen for use as the mnemonics. For example, an interface could be designed with the logical groups of Building, Floor and Room, and the string could then include Bxx,Fyy,Rzz to convey the dimensional information.

The Chksum is an optional field used for verification of the data transmission and can be implemented into the interface by having both systems calculate the checksum for a given string. The external system can be programmed to calculate the 1 byte cumulative checksum of the string it is sending, and the Walker product can perform the same calculation upon receipt of the string. Comparison of the checksums at either end of the interface provides a low-overhead means of verifying transmission integrity. Automatic Checksum tools provided by Walker are detailed later in this document.

Using the String Format

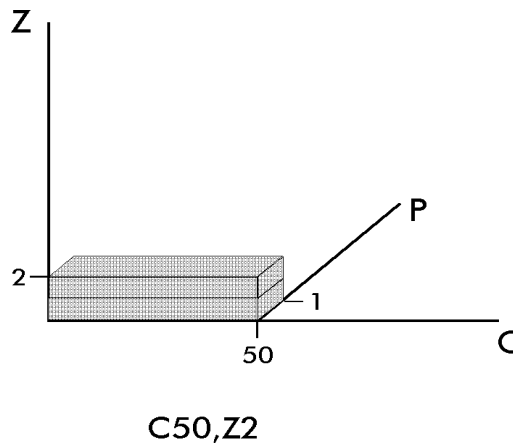
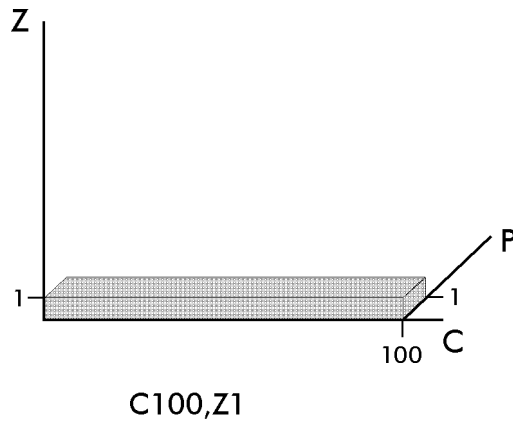
The C, Z and P fields in the string tell Walker devices where to map the value contained in the Tn, En, Sn, or Fn field. Using Cxx,Zyy,Pzz creates a three dimensional array for mapping the incoming data into memory. Remember that C, Z and P are arbitrary selections used here for the purpose of example; any letters can be used. Each output from the external system is mapped into a unique Walker variable point (Part II provides details for the Walker programmer).

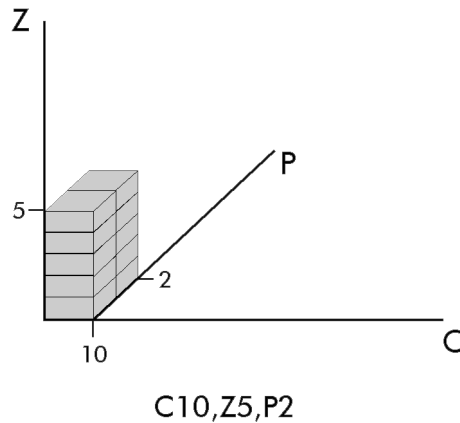


One, two or three dimensions can be used to map the data, depending on how many points there are, and what logical point groups have been developed. Deciding how to construct the memory map will depend primarily on the overall strategy of the interface. The same data structure can usually be implemented in one, two or three dimensions with choices optimizing memory better than others. The ASCII string is built to contain

fields for each dimension you will use — the P field is left out for two dimensional mapping, and both Z and P are omitted for one dimensional mapping.

For example, an external system with 100 points can be mapped as 1x100 (a one dimension array); 2x50 (a two dimension array), or perhaps as 2x5x10 (three dimensions). A visual interpretation of these three options is thus





Limits In order to predict the result of the data transfer, the $C \times Z \times P$ array must be given limits. The maximum amount of circuits, zones and panels must be communicated to the Walker programmer so the array can be set up properly. The maximum possible array size is 2000x2000x2000, but an array of this size is likely to exceed memory limitations. Decide how many points the external system will send, then organize your array appropriately.

Example You are Jim from Jim's Door Company. You have 25 floors in a building, and there are 50 doors on each floor. Your system sends a signal out for each door lock, and you want to register the state of each door with the energy management system.

Solution Consider each single lock as a circuit, resulting in $C=50$. Each floor can be considered as a zone, so $Z=25$. In this case there is no need to utilize the 3rd dimension, so $P=1$.

You have established the limits for the array as $C50,Z25,P1$. As $P=1$, this forms a 2 dimensional array. If there were 100 doors on each floor, you could use the same limits for C and Z , but set $P=2$. The array would stay the same size in the CZ plane, but double in "depth" in the P plane.

Each door on a given floor is a circuit, and would be numbered consecutively in the C field, from 1 through 50. Each floor is a zone, and would be numbered consecutively in the Z field from 1 to 25.

To establish limits, use any dimensional combination which best suits the application. Keep the 2000x2000x2000 maximum limits in mind, and split up the data in any logical progression.

The ASCII field (Tn)

From our previous example we have established the following string fragment:

?_J>C_C50,Z25,P1 or, simplified ?_J>C_C50,Z25

where J>C denotes input from Jim's Door Company (J) to Walker (C), and the array size is 50x25 for a total of 1250 data points to be mapped.

We now need to assign either ASCII or decimal values to the system and determine if the data is sent out in strings or singular. As mentioned earlier, you use either one of the four techniques ASCII (Tn) , ASCII strings (Sn), Decimal (En), or decimal strings (Fn). For this example we will use single ASCII field (Tn).

Your system registers the state of each door lock as open or closed using whatever method you have devised. What must be done now is the translation of your data into ASCII symbols. You must take whatever data you receive as the door open or closed condition and assign to it some ASCII value:

Your system says: OPENDOOR, so assign OPENDOOR=O

Your system says: CLOSEDDOOR, so assign CLOSEDDOOR=C

The assignment of an ASCII character to your conditional arguments is completely arbitrary. We chose O and C above because of the obvious logical connection. You have now completed the building blocks for the string to read as follows:

For the fifth door on the third floor, in the open condition:

?_J>C_C5,Z03_TO

For the same door, now in the closed condition:

?_J>C_C5,Z03_TC

Other options for transmission of data

The ASCII String Field (Fn)

Strings of data can be sent by the Source using the F(n) field format. When the character string mode is used, data is grouped together and sent all at once. The following line would be transmitted to indicate that in Zone 3: C1, C2, & C3 are closed and C4 and C5 are open.

?_J>C_C1,Z03_SCCCOO

The Decimal Field (En) Data can be sent through the ASCII string in a decimal form if required. This is done by replacing the character T with the character E in the string (refer to the general string format discussion). When the decimal mode is used, external data must be set to equal some decimal value instead of ASCII characters.

The Decimal String Field (Sn) Strings of decimal data can be sent by the Source using the S(n) field format. When the decimal string mode is used, data is grouped together and sent all at once. The following line would be transmitted to indicate that in Zone 3: C1, C2, & C3 are closed and C4 and C5 are open.

```
?_J>C_C1,Z03_F11100
```

Outputting the Walker ASCII Protocol String

In the above steps you have translated both dimensional (location) information and your system's conditional arguments into simple ASCII strings. Now create a program that will take all of this pertinent data and perform an ASCII dump to the serial connection between the two systems. The Walker Systems' device polls the serial port and collects the strings as they are generated.

In general terms, the program written to output the ASCII strings will perform these functions:

1. Read the output from each external point that will be sent to the Walker system.
2. Convert the output into either an ASCII character or a decimal value which will represent the output condition. The ASCII character or decimal value assigned will be used to represent a given condition in every transaction.
3. Attach the character, decimal value, character string or decimal string to the appropriate ASCII string. The string will be pre-built based on the information given in this document. The first five fields (the start-of-string mnemonic, the direction field and the dimensional information) will never change for a given point.
4. Dump the complete ASCII string to the serial connection with the Walker device. Depending on the application, strings can be sent up to several times a second.

Optionally, the program can be designed to calculate a 1 byte cumulative checksum for each string sent, and Walker can automatically perform the same calculation upon receipt of the string and transmit the checksum back. Simple routines can then be enacted to verify transmission integrity. Walker Systems Corporation can assist in the development of these routines if required.

Checksum Calculation

Generating a 1 byte cumulative checksum involves adding up the values of each character in the string. Note the complete Walker ASCII Protocol string requires a field delimiter before the last 2 digits of the checksum are added. This is illustrated by the following example:

Example Calculate the checksum for the string

?_J>C_C01,R01,S01_TA_

Solution Referring to the *IBM family character set*, we find:

character	hex value	decimal value
?	3F	63
_	5F	95
J	4A	74
>	3E	62
C	43	67
0	30	48
1	31	49
,	2C	44
R	52	82
S	53	83
T	54	84
A	41	65

Given the above data, add the values together (you may use hexadecimal values throughout, or use decimal values for the addition and then convert the sum to hex). The ASCII characters hex values added together looks like this:

$$3F+5F+4A+3E+43+5F+43+30+31+2C+52+30+31+2C+53+30+31+5F+54+41+5F=57E$$

The last 2 digits of the sum 57E are 7E, so the checksum added to the string sent to the Walker system is:

?_J>C_C01,R01,S01_TA_7E

PROGRAMMING WALKER PRODUCTS

Overview The Walker controls system requires some programming to properly interpret and assimilate the incoming ASCII strings. A new database point type, the ASCII Protocol point (APn), new communication variables and a new GCL command have been added to the operating system, creating the functionality required for the interface.

The AP point is built by the Walker technician to store the formatted information obtained when the interface is engineered. The string's format and limits are entered into this point, defining how the protocol will be used for a given application.

The communication variables (comvars) are used in conjunction with the new GCL MESSAGE command to control data transfers between the two systems. Simple GCL routines are constructed to direct, monitor and verify the ASCII strings.

Preparing Walker Products for the Interface

The first and most important step in engineering the system interface is communicating with the external system administrators. It is necessary to obtain several pieces of information before any Walker programming can be performed. Preliminary information required includes:

1. The total number of points from the external system that will be included in the interface.
2. The details of the logical groups the external data will be segregated into. This information creates the dimensional (location) section of the ASCII strings.
3. Whether the conditional arguments describing the incoming points will be sent as ASCII characters or decimal values, and what these values will be.
4. If checksum verification is required for the application, and if so, how it will be implemented.

It is highly recommended that representatives of both Walker Systems and the external system work together to define the interface scheme and the particulars of the ASCII strings. Once the above information is ascertained, the required AP point and GCL code can be easily generated.

The ASCII Protocol Point (APn)

The AP point stores all of the information that Walker devices will use to perform the interface with the external system. A separate AP point is required for each string format used in the interface; some jobs will require only one point while others may require several.

AP points are built at a terminal connected to a Network Interface or through the Operator window in Connect-500. In either case, new points are built by selecting Program, Database, Input/Output, Protocol ('PDIP) from the main system menu. Each data field in the AP point is described in detail in the following paragraphs.

When the operator selects 'PDIP from the main Network Interface menu, the following menu appears:

Build Add Copy Revise Erase Download Group Select

This is similar to most other Walker point types. Functionality is as follows:

Build	begins the point creation process
Add	prompts for the number of AP points to add to the system and which AP point to copy
Copy	prompts for which AP point to copy, location to start copying at and the number of points to copy
Revise	prompts for an existing AP point to revise
Erase	prompts for an existing AP point to erase
Download	downloads either the selected AP point or AP point last created
Group	displays the group of existing AP points in the panel
Select	displays the group of existing AP points and prompts for a selection

When Build is selected, the operator sees:

```
Point define --- PA101-AP1
Enable                1/0 (or GCL input)           ? 1
```

The Enable element allows the operator to either enable the AP point by selecting 1 or disable the point by selecting 0. The point must be enabled in order to function.

The next data elements create the ASCII Protocol string format. First are the source and destination mnemonics which allow the system to differentiate between incoming and outgoing strings:

PROTOCOL STRING Format- ?_DIRECTION_LOCATION_VALUE_CHECKSUM
DIRECTION FIELD Format- OUTPUT string eg. C>P ? C>M
INPUT string eg. P>C ? M>C

Next are the dimensional (or location) mnemonics which identify the format and dimensions that the location information will be in when the strings are received:

LOCATION FIELD Format- Pxx_Qyy_Rzz
where - P,Q,R are LETTERS denoting dimension fields (CR if unused)
and - xx,yy,zz are dimensions (CR if 0)
Enter LETTER for 'P' field ? A
Minimum value for xx (0-2000) ? 1
Maximum value for xx (0-2000) ? 10
Enter LETTER for 'Q' field ? B
Minimum value for yy (0-2000) ? 1
Maximum value for yy (0-2000) ? 10
Enter LETTER for 'R' field ? C
Minimum value for zz (0-2000) ? 1
Maximum value for zz (0-2000) ? 10

As discussed earlier, the selection of letters for the dimension fields is arbitrary. Enter carriage returns in the second and/or third fields if only 1 or 2 dimensions are required. Refer to the discussion of Limits earlier in this document for details.

Directly after the mnemonics and limits for string dimensions are entered, the system prompts as follows:

First destination for mapping ? VB1

The value entered here determines the starting point for mapping the incoming data in the Walker database. Only VB or VC variables are valid choices. It is necessary to understand how other VB or VC variables are used in the system to determine where the incoming data should be placed. Further, evaluating the limits and dimensional information in the string will determine how many variables will be used.

If a 3-dimensional array is to be used, and the limits for the dimension string fragment are A10,B10,C10, the system will require 10x10x10=1000 variables to handle the incoming data. If the starting point is VB1, the system can be expected to use VB1 through VB1000. The example in Part III of this document shows how the data is mapped based on limits, dimensions and this first mapping destination.

The last element provides the option to use ASCII characters, ASCII strings characters, decimal values or decimal strings in the Protocol:

VALUE FIELD Format- T = character S = character string
E = value F = value string ? T

As shown above, enter the letter T, E, S, or F depending on the type of value field to be used.

As stated earlier, communication with the external system administrators is a crucial first step in engineering the interface. Most of the information input into the AP point will be a function of the external system parameters.

Communication Variables (Comvars) and the MESSAGE Command

There are 6 new communication variables (comvars) and a new GCL command for use with Walker ASCII Protocol applications. The GCL MESSAGE command facilitates the sending of ASCII strings through the modem or terminal port of a Walker panel, and comvars provide a means to verify that the destination received the string sent out. The use of comvars is optional.

Comvars are the letters U, V, W, X, Y, and Z. Any of these letters can be used as comvars to determine the state of an ASCII string being sent out by Walker. Comvars hold certain values which can be read by the system and changed to reflect events. The comvars are similar to internal variables in that they cannot be displayed directly. One must set them equal to another variable in GCL in order to display them in the graphics or operator windows. The comvar states and their decimal values in the system are as follows:

Comvar	Hex Value	Decimal Value
READY	80	128
TIMEOUT	82	130
COMPLETED	83	132
WAIT_RESPONSE	88	135

In order to handle a multitude of interfacing configurations, the Message command is designed to be extremely flexible. Message commands can utilize numbers, letters and variables. It also has the ability to recognize a request to send an ASCII character. Multiple Message commands can be used to build up a single output string. One complete output string can be sent out every scan.

The MESSAGE command is used in conjunction with any one of 5 operators. These operators determine the destination of the string, and facilitate the use of comvar verification. The use of the MESSAGE command, and the MESSAGE command operators are as follows:

format:	MESSAGE ("OPERATOR" STRING)	
operators:	"APCOM1:"	sends string out of the terminal port with comvar verification and ChksumCRLF
	"APCOM2:"	sends string out of the modem port with comvar verifications and ChksumCRLF
	"COM2:"	sends string out of the modem port with no comvar verification or ChksumCRLF

"CON:" sends string out of the terminal port with no comvar verification or ChksumCRLF

Notice that only "APCOM1:" and "APCOM2:" make use of the comvar verification. Comvar verification consists of Walker calculating a 1 byte cumulative checksum for the string sent out. The external system must be programmed to perform the same calculation once it has received the string, and then send the checksum back to Walker.

The comvar used in the string will hold a value of 85 (ACTIVE) until the correct checksum is received back from the external destination. If the matching checksum is returned from the external system, the comvar goes to 83 (COMPLETED). All of the other possible comvar values are useful indicators of what has occurred during a transmission. For this type of verification to occur, the external system must be programmed to send out its cumulative 1 byte checksum after receiving a string from Walker.

The comvar timeout default is 5 seconds and can be changed in GCL by setting the letter of the comvar equal to some decimal value. For example, if you want comvar V to wait 15 seconds before it times out, insert the following line of code into the GCL program above the MESSAGE command that uses the comvar V:

V = 15

The following are examples of MESSAGE strings sent out by Walker devices.

1. MESSAGE ("APCOM1:" U ?_F>C_C01,Z01,P01_TA_)

?_F>C_C01,Z01,P01_TA_E5CRLF will be sent out of the terminal port. The comvar U will be equal to 85 (ACTIVE) until a checksum of E5 is returned from the external system and U goes to 83 (COMPLETED), or until the default 5 second timeout elapses and U goes to 82 (TIMEOUT).
2. MESSAGE ("APCOM2:" V ?_F>C_C02,Z02,P03_TA_)

?_F>C_C02,Z02,P03_TA_E9CRLF will be sent out of the modem port. Comvar V will be equal to 85 (ACTIVE) until either the default 5 second timeout expires (82) or the checksum is returned (83).
3. MESSAGE ("COM2:" ATDT7273651 \0D)

ATDT7273651 CR , will be sent out of the modem port with no checksum attached. The \0D is the ASCII hex value of the carriage return. This ability to Dial out modem strings allows the system to perform Remote Dial in and Database Mapping between two or more Remote Walker Panels. Contact Walker System for more information on this newly developed feature.
4. MESSAGE ("CON:" SYSTEM IN SHUTDOWN MODE \0D)

This Message command will result in SYSTEM IN SHUTDOWN MODE being displayed on the attached terminal with no checksum attached . A checksum from the destination is not expected.

-
5. MESSAGE ("COM2:" ?_C>A_C01,Z01,P01_S)
 MESSAGE ("COM2:" VB1 \0D)

Say VB1 = 110, then the two Message commands combine to send ?_C>A_C01,Z01,P01_S110_CR out the modem port. A checksum from the destination is not expected. When using variables in the Message command: the VBs are sent as integers, VAs & VCs are sent with 1 decimal place and VFs are sent with 0 - 3 decimal places depending on the magnitude of the value.

Using Comvars in GCL

The state that a comvar is in can be used in GCL to perform a variety of functions, such as resending the string or generating an alarm. When programming GCL to perform these functions, remember the time delay inherent to the comvar. As stated above, the default comvar timeout is 5 seconds, and programs designed to check comvar status must take this into account.

The following GCL sample program illustrates the sending of an ASCII Protocol string with a simple comvar check is used to ensure that the transmission was received. The code will send the value of VA1 every hour and utilize comvar to verify transmission.

```
DO-EVERY 1H
J = 1
END-DO

IF ( J = 1 ) OR ( VA3 OFF-FOR 1S ) THEN SEND_STRING
IF VA2 ON-FOR 10S THEN CHECK

:SEND_STRING
J = 0
VA2 = 1
VA3 = 1
MESSAGE ( "APCOM2:" U ?_C>L_F1_T VA1 _ )
RET

:CHECK
VA2 = 0
IF U = ( "COMPLETED" ) THEN VA3 = 1 ELSE VA3 = 0
RET
```

In the above program, the ASCII string following the message command will be sent out every 10 seconds until Walker receives the correct checksum back from the destination system.

The comvar U will timeout in 5 seconds (the default timeout) if the destination system does not respond with the correct checksum. After 10 seconds, the CHECK_SUCCESS subroutine will return J = 1 if the comvar U shows the transmission not completed, and the string will be sent again. If CHECK_SUCCESS finds that U equals completed, it returns J = 0 and stops any more transmissions until the next change of state of IP1.

INTERFACE EXAMPLE

Scenario Overview

A 3 story office complex houses two control systems from different vendors for fire suppression and HVAC/energy management. The building operations manager wishes to interface the two systems so that one master system will have the ability to monitor and display all controlled points. The intention is to have the HVAC system, Walker devices, receive and manipulate all point data from the fire suppression system.

The building also has a comprehensive computer network which links every PC workstation together in a client/server arrangement. At present, this network is used only to share data and hardware resources among the office workers. There is a resident network administrator who is a competent programmer. He suggests using the Walker ASCII Protocol to transmit fire alarms from the control system to the building-wide LAN, so all users will be made aware of fire danger right at their workstations.

The fire suppression system, the Meltov & Bernawai 5000, consists of networked smoke detectors and pull stations throughout the building. Network nodes poll the smoke detectors and pull stations and receive conditional statements from each device on a regular basis. These devices are all digital, sending out either "OK" or "ON FIRE" signals.

External System Analysis

System administrators from Meltov & Bernawai submitted the following floor plan typical of all 3 floors in the building (see next page). This plan indicates the locations of all smoke detectors and pull stations.

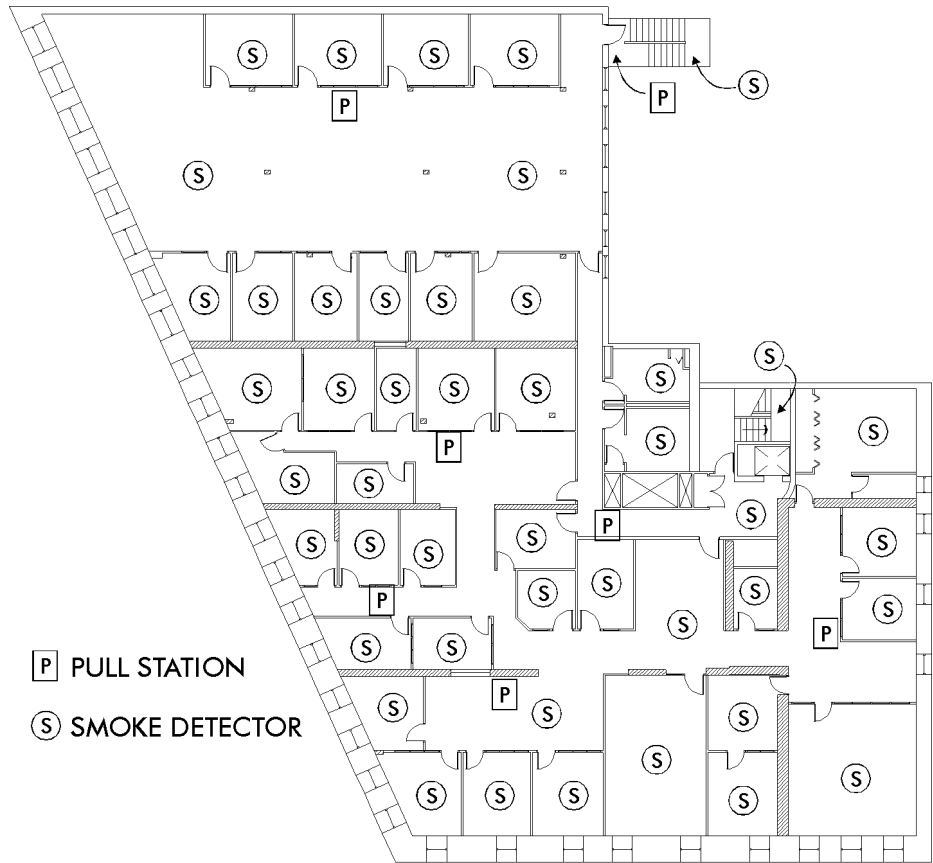
The following information was obtained from meetings with Meltov & Bernawai technicians:

- There are 46 smoke detectors on each floor; 138 in total.
- There are 5 pull stations on each floor; 15 in total.
- Each floor's smoke detectors and pull stations report to one master panel; there are 3 master panels in the fire suppression system.
- M & B master panels poll the fire suppression devices and register a condition every 10 seconds (this can be changed if required).

Walker Analysis

There is a WS1616 panel available to service the three floors of the building. Each floor's external points will be received and assimilated by this local WS1616. This indicates:

- There will be three zones; one for each floor of the building.



TYPICAL FLOOR PLAN

ASCII String Construction

The string required for the interface between the fire suppression system and Walker will be constructed based on the above analyses. The most appropriate interface will contain 3 zones (one for each floor), each containing 51 devices (46 smoke detectors and 5 pull stations). The general ASCII string for the application will look like this:

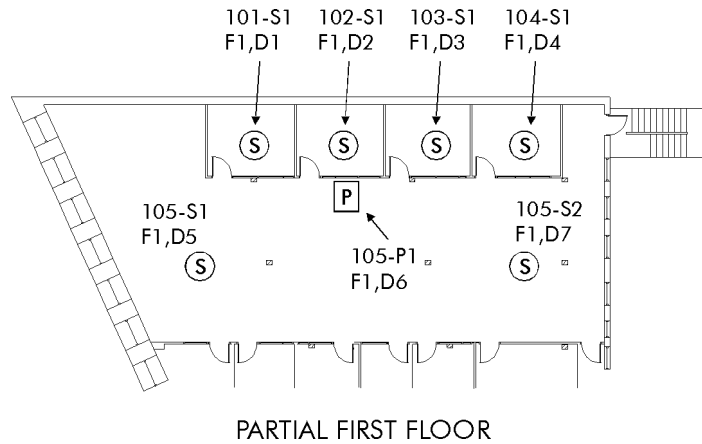
?_M>C_Fxx,Dyy_Tn

where: M signifies Meltov & Bernawai
C indicates Walker
F will be the Floor indicator, xx indicates which floor, from 1 to 3
D will be the Device indicator, yy indicates the device, from 1 to 51

T will be the character indicator, n indicates the character

Dimensional limits for the above string are F3,D51, as there is a total of 3 floors and 51 devices per floor. The interface will contain a total of 153 points (3 x 51).

In order for the Meltov & Bernawai administrators to organize their data, it is necessary to assign a unique label or number to each device in the system. This is also required for the interface to Walker devices. It is recommended that a table is drafted containing the unique number for each device together with its unique ASCII dimension. Labels for the M & B devices are shown in this partial first floor plan:



In the above plan, the M & B device labels are shown, together with the dimensions that will be used for that device in the interface to Walker. The table recommended above will look like this:

POINT	DIMENSION	DESCRIPTION
101-S1	F1,D1 (FLOOR 1, DEVICE 1)	Rm 101 SD
102-S1	F1,D2	Rm 102 SD
103-S1	F1,D3	Rm 103 SD
104-S1	F1,D4	Rm 104 SD
105-S1	F1,D5	Corr 105 SD1
105-P1	F1,D6	Corr 105 PULL
105-S2	F1,D7	Corr 105 SD2

Generating the Program

Once the above data has been compiled, the remaining step for M & B administrators is to construct the program that will dump the ASCII strings to the local Walker device. This part of the procedure will presumably be different for every application and every external system vendor.

In the case of Meltov & Bernawai, their panels recognize each device in the system based on addresses physically set on the hardware. The program they generate will somehow have to attach the address of a given device to the dimensional information created in the above steps.

The program must also attach an ASCII character to the value the device returns when polled. As the devices are digital in nature, this step is comparatively simple. The values chosen to indicate device conditions must be documented and conveyed to Walker technicians, as the destination system must be programmed to understand what it is receiving.

Pseudo-code representative of the required program looks like this:

```
[ GET STATUS FROM SMOKE DETECTOR 101-SD1 ]
POLL 101-SD1 AND GET VALUE

IF VALUE=ON THEN VARIABLE=Y
ELSE VARIABLE=N           ;sets ASCII value

PRINT "?_M>C_F1,D1_T(VARIABLE)" TO SERIAL PORT
```

A program such as the above would have to be generated for each controlled point, with the appropriate dimensions in the F and D field substituted.

Note that in this part of the example there is no transmission verification techniques used.

Programming Walker Devices

At this point all the required information has been obtained from Meltov & Bernawai. The AP point can now be built up in the WS1616 panel that will be used.

The Walker controls system in the building is used such that there are no restrictions on the VB and VC variables that can be used for the interface. It has been identified that the incoming data can be mapped at VB1 through VB153 in the WS1616 at address 102. The AP point, 102-AP1, will resemble the following:

```
Point define --- PA101-AP1

Enable          1/0 (or GCL input)          ? 1
PROTOCOL STRING Format- ?_DIRECTION_LOCATION_VALUE_CHECKSUM

DIRECTION FIELD Format- OUTPUT string eg. C>P          ? C>M
                   INPUT string eg. P>C              ? M>C
LOCATION FIELD   Format- Pxx_Qyy_Rzz
where - P,Q,R are LETTERS denoting dimension fields ( CR if unused )
and   - xx,yy,zz are dimensions ( CR if 0 )
Enter LETTER for 'P' field                          ? F
Minimum value   for xx (0-2000)                     ? 1
Maximum value   for xx (0-2000)                     ? 3
Enter LETTER for 'Q' field                          ? 0
Minimum value   for yy (0-2000)                     ? 1
Maximum value   for yy (0-2000)                     ? 51
Enter LETTER for 'R' field                          ? 0
Minimum value   for zz (0-2000)                     ? 0
Maximum value   for zz (0-2000)                     ? 0
First destination for mapping                       ? VB1
VALUE FIELD    Format- T = character S = character string
                   E = value      F = value string ? T

Build  Add  Copy  Revise  Erase  Download  Group  Select
```

Data Mapping Locations

It is simple to identify where the current value for a given device will reside inside a panel's database. To locate a particular device in Walker memory, multiply the values in the dimension fields of the ASCII string and add the value of the first destination for mapping minus 1:

If we represent the string thus:

$$?_M>C_F_x,D_y_TN_$$

and our variables points used for mapping are represented as VBz (or VCz if VC variables are used), then the location for any point is given by:

$$xy + (z-1)$$

In our example, VB1 is the first destination for mapping, so the device located by the string `?_M>C_F2,D28_TN` will map into the variable location:

$$[(2)(28)] + (1-1) = 56 = VB56.$$

VB56 will always hold the value for the device located at F2,D28. With this information you always know where every point's data resides.

Exporting from Walker Devices

The final stage of the interface is transmitting the data obtained from the fire suppression system out of Walker to the building's LAN administrator. As mentioned earlier, the LAN administrator is going to write his own program to use the data from Walker for special displays on the computer stations throughout the building.

To send the information to the LAN we will use the GCL MESSAGE command. We will also use the optional transmission verification provided by Walker comvars.

Unlike the previous interface to the fire system, the construction of an AP point is not required to keep track of the data exchange rules when data is sent out from Walker controls to the office LAN system. All of the data that will be sent to the LAN is already contained in the SAC database, so no rules are required to manage the interface. This is the case when data is output from Walker; the information is just sent straight out. AP points are only required when data is incoming.

The LAN administrator has been briefed on ASCII Protocol procedures, and he has a section of memory allocated in his workstation for the incoming data. He has constructed a program that will read and interpret the standard Walker ASCII strings and order the incoming data in his custom database accordingly. As a memory aid, we will represent the LAN with the letter L in the strings we send out.

The GCL required for this part of the interface will obtain the value from the VB points in the SAC database and send out one of two possible messages (one message for device OFF and one for device ON). GCL recognizes Ascii characters as hex Information required to construct the appropriate GCL is obtained from previous work in this example:

- The ASCII value Y indicates the M & B device is ON (activated by fire condition).
- The ASCII value N indicates the device is OFF (normal condition).

GCL will also be used to enable comvar checking to verify transmission integrity. To utilize the ASCII Characters such as Y and N as arguments in GCL, they must be in the Ascii Hex form (i.e. in GCL Y would be \89 and N would be \78)

The GCL required to send the data for the first device (the smoke detector located at F1_D1) looks like this:

```
IF ( VB1 >< VA1 ) OR ( VA3 OFF-FOR 1S ) THEN SEND_STRING
VA1 = VB1
IF VA2 ON-FOR 10S THEN CHECK
```

```
:SEND_STRING
U = 2
VA2 = 1
VA3 = 1
```

```
IF VB1 = \89 THEN MESSAGE ( "APCOM2:" U ?_C>L_F1,D1_TY_
), /C
VA2 = 1 ELSE MESSAGE ( "APCOM2:" U ?_C>L_F1,D1_TN_ )
RET
```

```
:CHECK
VA2 = 0
IF U = ( "COMPLETED" ) THEN VA3 = 1 ELSE VA3 = 0
RET
```

The GCL is written to send out a string on a change of state on VB1. Two seconds after sending the string the transmission will be checked to see if it was completed. If it failed the code will retransmit the string.

The code would be required for each point sent to the LAN. Further, the LAN administrator must be able to interpret the values Y and N for their conditions.

There will be several ways to implement an interface such as the one in this example. The GCL above represents one simple way to get the information sent and checked, however, this is probably not the most elegant solution possible. In all applications of the Walker ASCII Protocol, both Walker technicians and external system administrators are encouraged to contact walker Systems' Support for engineering assistance.

PROGRAMMING TECHNIQUE: OVERLAPPING MEMORY

Multiple AP points can overlap the memory that will be assigned for VB or VC variable mapping destinations. It may appear at first that overlapping memory areas will cause a problem, but an overlap can often be constructed to save memory space. Several AP points can exist in one SAC's database, and they can be set up to reference the same memory locations. There are many applications that can be made more efficient using these techniques.

Large single-dimension applications can be separated into two dimensions denoted by the same ASCII mnemonic. This will result in the first field representing the hundreds space of the total number of points, and the second field representing the tens space.

For example, if you had a building with 25 floors and 25 rooms per floor, you might build a single dimension starting at the first room number, 101, and ending at the last room number 2525. This would fill 2425 variable spaces in the SAC database, but only 25% of these spaces would actually contain useful data. There are two potential uses of memory overlapping that will streamline this application.

The simplest use of overlapping is to build the AP using all 2425 memory locations, then building other APs to use the sections of memory that will not be used. This is only useful if there are other interfaces to the SAC that will be served by the spaces made available (for example, the space from VB126 to VB200 or from VB226 to VB300, et cetera).

A second overlapping technique allows you to build an AP point with the first two fields denoted by the same character (in this case we will use R, for Room). The range for both the first and second dimension fields would be set as 1 to 25. When the string is received by Walker, the R dimension is split into two fields; one for the tens and one for the hundreds. This way you will use up only 625 spaces instead of 2425.

To avoid confusion, just keep track of how you have used the Protocol. In the above example, room 1215 would be mapped into variable space $12 \times 15 = 180 = VB180$ instead of VB1215.

There are many ways to plan and construct efficient database use. Any time you can conserve memory you will benefit from faster scan rates and better system response. You can use overlapping in many applications.