Honeywell

Lesson 4

4410S

TotalPlant

Notices and Trademarks

Honeywell Inc.

Industrial Automation and Control

Automation College

2820 West Kelton Lane

Phoenix, AZ 85053-3028

1-800 852-3211

Lesson 4 - Table of Contents

Introduction

LESSON OVERVIEW

Lesson Objectives

Upon completion of this lesson, you will have an understanding of CL/AM

- Logical Functions
- Miscellaneous Functions
- Arithmetic functions
- Array Functions
- System Subroutines
- Custom Subroutines
- CL Runtime Extensions and how to implement
- FILE I/O CL Extension
- AMCL01 CDS/Multiple Parameter Move
- AMCL02 Matrix Math Library
- AMCL03 Continuous History Access
- AMCL04 Fast External CDS Fetches
- AMCL05 Off Node Access

Lesson Outline

INTRODUCTION

LESSON OVERVIEW

System Functions

LOGICAL FUNCTIONS
MISCELLANEOUS FUNCTIONS
ARITHMETIC FUNCTIONS
ARRAY FUNCTIONS

Subroutines

SYSTEM
CUSTOM

CL Runtime Extensions

WHAT ARE THEY?
NCF REFERENCE
&CUS AND &CLX FILES
%INCLUDE_SET DIRECTIVE

File I/O CL Extensions

SUBROUTINES IN FILE I/O EXTENSIONS
File$Open_File Subroutine
MATRIX MATH LIBRARY

AMCL02

MATRIX MATH LIBRARY

AMCL03

CONTINUOUS HISTORY ACCESS

AMCL01

CDX_MOVE AND MULTIPLE_MOVE_PARAMTER

AMCL04/05

AM EXTENSION FOR FAST EXTERNAL CDS FETCH
AM EXTENSION FOR OFF-LOGICAL-NODE ACCESS

System Functions

LOGICAL FUNCTIONS

- Logical functions are used to test a condition and return a Logical value if true or false.
- **Comm_Error (x)** accepts an argument that is an unsubscripted data point/parameter of any data type. It returns ON if there is a communications error when this parameter is prefetched, OFF otherwise.

Example:

```
If Comm_Error(A100.PV) then send: "A100.PV Comm Error"
```

- **Equal_Null_Point_Id (p)** determines if a parameter of type Point_Id (may be subscripted) is null. It returns ON if value is null otherwise OFF.

Example:

```
Parameter Tin: $Reg_Ctl
If Equal_Null_Point_Id(Tin)then send : " Parameter Tin is null
"
```

- **Equal_Point_Id (p1,p2)** compares two CDS parameters of type Point_Id (may be subscripted) and returns ON if equal, otherwise returns OFF.

Example:

```
Parameter Tin: $Reg_Ctl
Parameter Tlist: $Reg_Ctl Array(1..10)
If Equal_Point_Id (Tin, Tlist(4)) then send: " Tin =
Tlist(4)"
```

System Functions

LOGICAL FUNCTIONS

- **Exists(p)** accepts an argument that is an unsubscripted datapoint.parameter reference of any data type. It returns ON if the point and parameter exist, otherwise OFF.

Example:

```
If Exists(A100.pv) then send : "A100.pv exists"
Else send: "A100.pv does not exist"
```

- **Bkg_Switchover_Restart** is used to detect a switchover and program restart in the backup AM. This function returns ON only if called during the first execution of a CL block following a switchover.

Example:

```
If Bkg_Switchover_Restart then
&   send: "Background CL ABC restarted"
```

System Functions

MISCELLANEOUS FUNCTIONS

- **Date_Time** returns absolute TDC3000 date/time since midnight of 1 Jan 1979.

Example:

```
Block Time(Generic;at General)
parameter maint : time
parameter count
If count = 0 then (set maint = (Date_Time + 30
days);
& set count = 1; goto finish)

If maint <= Date_Time then
& (send: "Time for maintenance"; set count = 0)
finish:exit
End Time
```

- **Now** returns wall-clock time (seconds since midnight of the current day). It's use is similar to Date_Time example except it only returns current time.

- **LEN(s)** returns the length of the string value of the designated string parameter.

Example:

```
Local name_length
Set name_length = Len(A100.name)
```

System Functions

MISCELLANEOUS FUNCTIONS

- **NUMBER(t)** is used to convert a time expression (t) to a number value representing number of seconds represented by the time expression.

Example:

```
local absolute_time
Set absolute_time = Number(Date_Time)
-- absolute_time would equal the number
-- of seconds since 1 Jan 1979
```

- **ORD(e)** returns the current state, not state name, of a system enumeration, custom enumeration, or self-defining enumeration.

Example:

```
If Ord(A100.mode) = 0 then send: " A100 is in Manual"
```

- **SELF** returns the slot number of the currently executing CL/AM block. A data point may have many CL/AM blocks; therefore , all parameters of the CL/AM segment that deal with individual blocks, like Clbkerr(n) or Clerrloc(n) are arrays indexed by a slot number.

Example:

```
If Clblkerr(self) <> noerror then
&    send: "Error last time I ran"
```

System Functions

ARITHMETIC FUNCTIONS

- All arithmetic functions listed below accept arguments of type number and return number results. The trigonometric functions must be specified in radians, not degrees.

- **Abs(x)** -- absolute value

  Avg(x,y,...) -- average

  Exp(x) -- exponential, e^x

  Int(x) -- truncate to integer

  Ln(x) -- natural logarithm

  Log10(x) -- common logarithm

  Max(x,y,..) -- maximum

  Min(x,y,..) -- minimum

  Round(x) -- round to integer

  Sqrt(x) -- square root

  Sum(x,y,..) -- sum

  Sin(x) -- sine

  Cos(x) -- cosine

  Tan(x) -- tangent

  Atan(x) -- arc tangent

Example:

```
local Sum_of_numbers
set Sum_of_numbers = Sum(A100.pv, A200.pv, 55)
```

System Functions

ARRAY FUNCTIONS

- Array functions require an array identifier argument and returns a number. The array must be local. The number of elements in the array or number of dimensions does not matter.

- **Avg(x)** -- returns the average of the array x
- **Max(x)** -- returns the largest number in the array x
- **Min(x)** -- returns the smallest number in the array x
- **Sum(x)** -- returns the sum of all the elements in array x.

Example:

```
Local array_of_numbers: array (1..50, 1..50)
Local maximum
Local minimum
Local average
Set maximum = Max(array_of_numbers)
Set minimum = Min(array_of_numbers)
Set average = Avg(array_of_numbers)
```

Subroutines

SYSTEM

- **Allow_Bad (x, y)** stores the value of y into the numeric x. If x is a parameter of a data point and y is a bad value, the bad value is stored but, unlike a Set command, the program will not abort.

Example:

```
Block ABC (Point A100; at Pv_Alg)
External A200, A300

Set Pv = ( A200.Pv + A300.Pv ) / 2
-- program will abort here if A200.Pv or A300.Pv is not-a-number

Call Allow_Bad (Pv, ( A200.Pv + A300.Pv ) /2)
-- will not abort if A200.Pv and/or A300.Pv is not-a-number
```

- **Bkg_Change_Priority (p)** used to change priority of a background CL program. See subject Background CL Priorities for more detail.This subroutine may only be called by a Background CL program.

Example:

```
Block ABC (Generic; at Backgrnd)
Local prior: $Bkgprty
Call Bkg_Change_Priority (Low)
----------
----------
Set prior = High
Call Bkg_Change_Priority (prior)
End ABC
```

Subroutines

SYSTEM

- **Bkg_Delay (t)** used to delay execution of Background CL program. See subject Background CL Priorities in previous lesson for more detail. This subroutine may only be called by a Background CL program.

Example:

```
Block ABC(Generic;at Backgrnd)
  Local tconst : Time
  Set tconst = 1 Mins 5 Secs
  Call Bkg_Delay (2 Mins)
  Call Bkg_Delay (20 Secs)
  Call Bkg_Delay (tconst)
End ABC
```

- **CDS_Read (s, fm, p, n)** reads a Custom Data Segment's values from a History Module file and overwrites an existing CDS's values on the Bound Data point. It can only be called by a Background CL program.

  Where **s** will contain the return status of the request

  fm will contain additional File Manager status information

  p is the pathname of the History Module file

  n is the name of the CDS on the Bound Data point

Example:

```
Local stat : $clfstat
Local fmstat : Number
Local path, cds_name : String
Set path = "NET>BTCH>RECIPE34.X"     -- Upper Case
Set cds_name = "RECIPE"              -- Upper Case
Call Cds_Read ( stat, fmstat, path, cds_name )
```

Subroutines

SYSTEM

- **CDS_Write (s, fm, p, n, o)** writes an existing CDS to a History Module file. It can only be called by a Background CL program.

  Where **s** will contain the return status of the request

  fm will contain additional File Manager status information

  p is the pathname of the History Module file

  n is the name of the CDS on the Bound Data point

  o indicates whether the file should be overwritten if it already exists.

  o = Off will create file if file does not exist, otherwise returns error.

  o = On will either create a file or overwrite existing file

Example:

```
Local stat : $clfstat
Local fmstat : Number
Local path, cds_name : String
Set path = "NET>BTCH>RECIPE34.X"   -- Upper Case
Set cds_name = "RECIPE"            -- Upper Case

Call Cds_Write ( stat, fmstat, path, cds_name,
Off)
```

Subroutines

SYSTEM

- **Delete_File (s, fm, p)** subroutine deletes the named History Module file (unless it is protected). It can only be called by a Background CL program.

  Where **s** will contain return status

  fm will contain additional File Manager status information

  p is the pathname of the History Module file to be deleted.

Example:

```
Local stat : $clfstat
Local fmstat : Number
Local path: String
Set path = "NET>BTCH>RECIPE34.X"   -- Upper Case

Call Delete_File ( stat, fmstat, path )
```

Subroutines

SYSTEM

- The enumeration **$clfstat** is the return status used for CDS_Read,CDS_Write, and Delete_File subroutines. The states are as follows:

  0 OK -- Successful store of CDS values

  1 Badpath -- invalid file path was specified

  2 Badfile -- specified file does not contain a CDS

  3 Mismatch -- structure of existing file and CDS do not match

  4 NoCDS -- specified CDS does not exist

  5 Corrupt -- write of CDS failed after the initial record was written to the file. The file exists, but it does not contain a complete CDS.

  6 Fmpviol -- file access privilege violation.

  7 Fmexist -- file exists, but Overwrite option was not specified.

  8 Fmnofil -- specified file does not exist.

  9 Fmhard -- device on which the file is located reported a physical error condition.

  10 Fmfiles -- no more files can be created on the specified volume.

  11 Fmspace -- insufficient space available on the medium to perform the requested action.

  12 Fmnovol -- the specified volume does not exist.

  13 Fmmount -- specified volume is not mounted.

  14 Fmvfull -- specified volume is full.

  15 Fmother -- another type of File Manager error has occured; the "FM" status argument contains a supplemental status code.

Subroutines

SYSTEM

- **Get_CL_Slot (s, i)** takes a string input, compares it with the names of all CL blocks on the point and returns a real number index value if a match is found, otherwise returns a -1.0 if no match found.

  Where **s** is the string input

  i is the index value

Example:

```
Block getslt ( Generic; at General )
Local i
Call Get_CL_Slot ("GETSLT", i)
if i <> -1.0 then send:  "getslt is CL slot", i
else send: "getslt not found"
```

note: Only the first eight characters of the string are used for the comparison.

The string must be in upper case or a -1.0 will be returned.

Subroutines

SYSTEM

- **Modify_String (s, ts, tp, n, ss, sp)** subroutine changes the value of a substring in a target string to the value of a substring copied from a source string. It may be called by either Background or Foreground CL blocks.

  Where **s** is a $Modstr enumeration value expressing return value states of OK or Failed.

  t s is the target string to be modified (in/out)

  t p is the index to the first character of the target string to be modified

  n is the number of characters to move from the source string to the target string

  s s is the source string

  s p is the index to the first character to be fetched from the source string

Example:

```
Local stat :  $Modstr
Local target, source: string
Local tindex, sindex, chars

Set target  =  "net>cust>recipenn.xx "
Set tindex  =  16.0
Set chars   =  2.0
Set source  =  "ab"
Set sindex  =  1.0

Call Modify_String ( stat, target, tindex, chars, source, sindex )

-- Result is " net>cust>recipeab.xx "
```

Subroutines

SYSTEM

- **Enum_Value_Store (e, o, s)** takes a real number, converts it to an integer, stores that value as the state of an enumeration and returns a successful/fail status value.

  Where **e** is a reference to a standard, custom, or self-defined enumeration to be stored to. Enumeration may be a local variable, a parameter on the bound point, or a parameter on another point.

  o is a Number value that represents the desired ordinal state of the referenced enumeration.

  s is the Clerrsts enumeration value denoting return status.

Example:

```
External A100
Local status: Clerrsts
Local m  : Ptexecst

-- Set local variable active
Call Enum_Value_Store  ( m, 1.0, status )

-- set external point active
Call Enum_Value_Store  ( A100.Ptexecst, ord(m), status )

-- set Bound Data point's modattr parameter
Call Enum_Value_Store  ( modattr, ord(m), status )
```

note: The Enum_Value_Store is used in place of a Set statement when verification of a success/fail return status is desired.

Subroutines

SYSTEM

- **Move_Parameter (x, y, s)** subroutine copies a source parameter value to a target parameter value and returns a success/fail status value.

  Where **x** identifies the target parameter

  y identifies the source parameter

  s is a Clerrsts enumeration returning either NoError or a specific error e.g., BadValst, ComErr, Abort, etc.

Example:

```
Block ABC ( Generic; at General )
Parameter Tin     : $Reg_Ctl  Array (1..10)
Parameter Temp    : $Reg_Ctl  Array (1..10)

Local status      : Clerrsts
Local p

Set p = 1

Call Move_Parameter ( Tin(p), Temp(p), status )
-- Moves entity_id in Temp(1) to the entity parameter
Tin(1)

Call Move_Parameter (Tin, Temp, status )
-- Moves all values in array Temp to array Tin
```

Note : The transfer of Point-Id type data forces a relink of all CL using the target point_id parameter for indirection.

Subroutines

SYSTEM

- **Set_Null_Point_id (p, s)** subroutine stores a null point identifier to a CDS parameter of type entity. The CDS parameter may be either a single parameter or a subscripted element of an array.

  Where **p** defines the parameter

  s is the return success/fail status enumeration Clerrsts

Example:

```
Package
Custom
   Parameter pt1 : $Reg_Ctl
   Value A100
   Parameter pt2 : $Reg_Ctl  Array ( 1..5 )
   Value ( A101, A201, A301, A401, A501)
End Custom

Block Setnul (Generic; at General)
Local status  : Clerrsts

-- change A100 on Bound Data point to null value

Call Set_Null_Point_Id ( pt1, status )
If status <> Noerror then Send: " Pt1 Error", Ord (status)

-- change A301 on Bound Data point array to null value

Call Set_Null_Point_Id ( pt2(3), status )
If status <> Noerror then Send: " Pt2(3) Error", Ord
   (status)

End Setnul
End Package
```

Subroutines

SYSTEM

- The enumeration **Clerrsts** is the return status used for Enum_Value_Store, Move_Parameter, and Set_Null_Point_Id subroutines. The states are as follows:

  0 NoError -- No error detected

  1 Error3 -- N/A - Notify Honeywell Tac

  2 LimViol -- Limit Violation (minor error)

  3 Rights -- Access Rights Violation (minor error)

  4 CommErr -- Communications error

  5 Error2 -- N/A - Notify Honeywell Tac

  6 BadValst -- Bad value store (CL Failure condition) condition

  8 Abort -- Abort statement executed (CL Failure condition)

  9 Error1 -- N/A - Notify Honeywell Tac

  10 BranchV -- Backward Branch Violation (CL Error condition)

  11 ArithErr -- Arithmetic Error (CL Error condition)

  12 ArrayLim -- Array Limit Violation (CL Error condition)

  13 Range -- Range Limit Violation (CL Error condition)

  14 ProgErr -- Programming Error (CL Error condition)

  15 KeyLevel -- Key Level Restriction (CL Error condition)

  16 CnfErr -- Configuration Error (CL Error condition)

Subroutines

SYSTEM

- **Number_to_String (s, str, i, f)** subroutine creates a string characterization of a real number. This subroutine is callable from either Background or Foreground CL programs.

  Where **s** will contain the returned status **$convrs** enumeration states of OK, Badform, or Badvalu.

  s t r will contain the string Characterization of the real number.

  i identifies the real number to be converted

  f is a format specification describing the desired character representation of the converted number e.g., "Real" or "Unknown".

Example:

```
Local i
Local str   : string
Local fmt   : string
Local stat  : $convrs

Set fmt = "G99999"  -- unknown dummy format
Set  i  = 23.44

Call Number_to_String ( stat, str, i, fmt )
Send: " The number is = ", str

-- The example will produce the following operator
message -- display: The number is = 23.44
```

Subroutines

SYSTEM

- **Set_Bad (x)** subroutine is similar to the Set_Null_Point_Id subroutine except that Set_Bad is used to null a parameter of the type number.

  Where **x** is the numerical parameter to be given a bad bit pattern (NaN) e.g., " ---------".

Example:

```
Block ABC ( Point A100; at General )

Call Set_Bad ( PVHHTP )

--setting PVHHTP to NaN disables the alarm.
```

Subroutines

CUSTOM

- AM/CL allows for user created subroutines. Unlike the system-supplied subroutines, the custom subroutines must be defined and compiled along with the CL Block calling it.

- Custom Subroutines require adherence to the following syntax.

```
SUBROUTINE  <subroutine_Id> <(subroutine arguments)>
--
--    subroutine code
--
END  <subroutine_Id>
```

- Arguments must be defined as to IN (default), OUT, or INOUT. They must also be defined as to expected data type (Number is the default).

Example:

```
Block Store (Generic;at General)
   External Pointx
   Local Index1, Index2
   Set Index1 = 1

   Call Sub1 (Pointx.amfl(index1).pv, index2)

End Store

Subroutine Sub1 (value1:Out Number; value2:Out Number)
   Set value1 = 44
   Set value2 = 3
End Sub1

-- Result: Pointx.amfl(1).pv = 44 and Index2 = 3
```

note*:* Subroutine must immediately follow CL Block in source code.

Subroutines

CUSTOM

- Stores to OUT arguments are done in backwards order of their appearance in the arguments list.

Example:

```
Block Store (Generic; at General)
   External Pointx
   Local Index
   Set Index = 1

   Call Sub1 (Pointx.amfl(Index).pv , Index)

End Store

Subroutine Sub1 (value1:Out ; value2:Out )
   Set value1 = 44
   Set value2 = 3
End Sub1

-- Results: Pointx.amfl(3).pv = 44
```

Example: -- **Arguments reversed**

```
Block Store (Generic; at General)
     External Pointx
     Local Index
     Set Index = 1

     Call Sub1 (Index , Pointx.amfl(Index).pv)

End Store

Subroutine Sub1 (value2:Out ; value1:Out )
     Set value1 = 44
     Set value2 = 3
End Sub1

-- Results: Pointx.amfl(1).pv = 44
```

Subroutines

CUSTOM

- Local variables and functions can be defined in subroutines, but Parameters and External declarations are not permitted.

- A subroutine may access all functions and variables visible from the calling program with which it is compiled; however, the calling program cannot access variables declared within the subroutine.

- Whole arrays and parameters of data type Point_Id cannot be passed to custom subroutines.

- Be careful when using INOUT when only an IN argument is required. The INOUT may change the value of the main CL parameter used in the argument.

Example:

```
Block Store (Generic; at General)
   External Pointx
   Local var

   Call Sub1 (Pointx.amfl.pv , var)

End Store

Subroutine Sub1 (value1:InOut ; value2:Out )
   Local factor
   Set factor = 0.25
   If value1< 50 then set value1= 50
   Set value2 = value1* factor
End Sub1

-- if Pointx.amfl.pv was less than 50 then it will be
   modified by the subroutine.
```

CL Runtime Extensions

WHAT ARE THEY?

- CL Runtime Extensions are optional applications oriented subroutines and functions that are callable by CL/AM programs.

- These subroutines and files are packaged in files called Include Sets. These optional files are loaded into the AM during system startup and configuration

- Although intended to be used for optional Subroutines and functions that may be purchased from Honeywell, the following list of options are included with every system.

  File I/O CL extensions (R300) enable CL/AM programs to both read and write data from ASCII text files on the History Module and to create and access volumes, directories, and ASCII text files on removable media.

  AMCL01: CDS MOVE AND MULTIPLE MOVE PARAMETER(R401) load module adds ability to transfer large amounts of parameter data either via complete CDS transfers or use of parameter transfer lists.

  AMCL02: Math Library (R400) load module adds higher level mathematics subroutines and functions to CL/AM such as matrix math operations.

  AMCL03: Continuous History Data Access (R400) load module adds the ability for CL/AM to retrieve continuous history data from the History Module.

CL Runtime Extensions

NCF REFERENCE

- CL Runtime Extensions are loaded into the AM via External Load Modules under the direction of the NCF.

- On page #3 of the AM node NCF, External Load Module names are entered along with associated personality type.

Example:

```
                                                    06 Dec 93 08:59:58  3
APPLICATION MODULE NODE                                PAGE  3 OF  3  MOD AM  20

  NODE      20

ENTER EXTERNAL LOAD MODULE NAMES & ASSOCIATED PERSONALITY-TYPES:
  NAME --- PERS.      NAME --- PERS.      NAME --- PERS.      NAME --- PERS.
FILE      AMO       AMCL05    AMO
CONV      AMO
AMCL01    AMO
AMCL02    AMO
AMCL03    AMO
AMCL04    AMO


USE DEFAULT PERSONALITY TYPE?                 YES NO

                                                AMO
ADDITIONAL MODULE MEMORY (WORDS)

TOTAL (MODULES PLUS ADDITIONAL MEMORY)      333824


FURTHER EXTERNAL DIRECTIVES?                  YES NO
F1=CHECK     F3=SET OFFLINE   F5=ABORT        F7=NEXT ITEM      F9=PACK NCF
F2=INSTALL   F4=PRINT
```

- These entries effectively direct the AM to load these modules in addition to the normal AM personality.

Note: Reference Section 8 of *AM Implementation Guidelines*

CL Runtime Extensions

&CUS AND &CLX FILES

- &CUS and &CLX directories contain the files required to support the External Load Modules and the CL extensions.

- &CUS contains files like FILE.LO,AMCL01.LO, AMCL02.LO, and AMCL03.LO. which contain the subroutines and functions to be loaded for these External Load Modules.

- &CLX contains the set-definition files FILE.SF, AMCL01SF, AMCL02.SF and AMCL03.SF which describe the added CL routines to the CL compiler .

- &CUS and &CLX can be on a History Module or on removable media.

- To use CL Extensions "NET>&CUS" must be entered in the EXT LOAD MODULE port on the Engineering Personality's Modify Default Volume Pathnames display.

- As future External Load Modules become available, directions will accompany explaining the loading into the &CUS and &CLX files. It will also have instructions on adding the External Load Module to the NCF.

CL Runtime Extensions

%INCLUDE_SET DIRECTIVE

- CL/AM programs that will use CL Runtime Extensions must use the Include directive to specify which custom subroutines and/or functions are to be used.

Example:

```
Block ABC (Point A100; at Backgrnd)
--
Local  -------
--
%Include_Set File
--
Set  ----------

Call File$Open_File (Status, FM_Status, File_Id, Num_Records,
&                    Buffer_Size, Path, 0)
```

- For the other three mentioned runtime extensions, the Include Directive would look like:

```
%Include_Set AMCL01
%Include_Set AMCL02
%Include_Set AMCL03
```

- The Include Directive need only appear once in the Program to allow use of all subroutines included in that external Load Module. The Include directive must begin on the first column and must precede the first executable line of code.

File I/O CL Extensions

- The CL Extensions for FILE I/O consists of a set of subroutines that enable a CL/AM program to both write data to and read data from ASCII text files on the History Module or removable media (floppies or cartridge disks).

- FILE I/O provides the following input/output functions:

  Create volumes and directories and Delete directories from cartridge or floppy media

  Create, Open, Close, and Delete ASCII text files on the HM or on a floppy/cartridge

  Convert CL internal data types to external values, store them into fields, combine the fields into records, and write the records into files

  Read records from files and convert fields in the record to any appropriate CL internal data type

  Convert string characters to upper case (for use in comparisons)

  Rename files

  Protect and unprotect files

  Copy files

  Copy individual records from one file to another

  Determine volume size and free space

  Determine if file exists

  Search for files with certain name characteristics; get file attributes for each of those files

File I/O CL Extensions

SUBROUTINES IN FILE I/O EXTENSIONS

Subroutine Name	Subroutine Function	Section
File$Create_File	Create a new file	C.6.1
File$Open_File	Open existing file	C.6.2
File$Put_Field	Write values to fields in the file's I/O buffer	C.6.3
File$Put_Field_S	Write values to fields in a string variable*	C.6.4
File$Write_Record	Write record in the file's I/O buffer to the file	C.6.5
File$Read_Record	Read record from HM to the file's I/O buffer	C.6.6
File$Get_Field	Read values from fields in the file's I/O buffer	C.6.7
File$Get_Field_S	Read values from a string variable*	C.6.8
File$Close_File	Close a previously opened file	C.6.9
File$Copy_Record	Copy a record from one I/O buffer to another	C.6.10
File$Copy_File	Make copy of an existing file	C.6.11
File$Exists	Determine if a given file exists	C.6.12
File$Find_Field_Pointer	Find a field within the file's I/O buffer	C.6.13
File$Find_Field_Pointer_S	Find a field within a string variable*	C.6.14
File$Set_Null	Set length of the file's I/O buffer to zero	C.6.15
File$Blank_Fill	Place blank characters into the file's I/O buffer	C.6.16
File$Blank_Fill_S	Place blank characters into a string variable*	C.6.17
File$Rename	Change the name of an existing file	C.6.18
File$Safe_Rename	Give the name of one file to another file	C.6.19
File$Protect_File	Protect/unprotect file from deletion	C.6.20
File$Convert_SDE_to_Internal	Convert SDE to internal form	C.6.21
File$Convert_SDE_to_ASCII	Convert SDE to external form	C.6.22
File$Get_Volume_Statistics	Get space use information for volume	C.6.23
File$Create_Character	Convert number to ASCII character*	C.6.24
File$Convert_FM_Status	Convert FM status value to string	C.6.25
File$Strip_Blanks	Enable/disable stripping by File$Get_Field	C.6.26
File$Upper_Case	Convert string characters to upper case*	C.6.27
File$Delete_File	Delete file from fixed or removable media	C.6.28
File$Create_Volume	Create a volume on a floppy/cartridge disk	C.6.29
File$Create_Directory	Create a directory on a floppy/cartridge disk	C.6.30
File$Delete_Directory	Delete a directory from a floppy/cartridge disk	C.6.31
File$Read_Directory	List all files that match specified search criteria	C.6.32
File$Get_File_Attributes	Get file data from files in Read_Directory list	C.6.33
File$Read_Directory_Complete	Deallocate space used by Read_Directory list	C.6.34
File$Get_Volume_Directories	List all directories on the specified volume	C.6.35
File$Create_Volume_With_Descriptors	Create a volume with descriptors on a floppy/cartridge disk	C.6.36
File$Modify_Volume_ID_String	Modify a volume ID string	C.6.37
File$Modify_File_Descriptor	Add, modify, or delete a file's descriptor	C.6.38
File$Modify_Directory_Descriptor	Add, modify, or delete a directory descriptor	C.6.39
File$Get_File_Descriptor	Get the descriptor of a file previously created	C.6.40
File$Get_Volume_Directories_D	Get list of directories and descriptors	C.6.41

* File I/O Extension subroutines that can be called by non-Background CL programs.

File I/O CL Extensions

File$Open_File Subroutine

- **File$Open_File (s, fm, fn, nr, bs, p , ap)** subroutine is used to open an existing file on the History Module.

  Where **s** contains returned error status (out)
  fm contains returned file manager error status (out)
  fn contains file_id used on subsequent call (out)
  nr contains number of records currently in file (out)
  bs contains maximum characters per record (in)
  p contains full pathname of file (in)
  ap contains access privilege type (in):
  0 = exclusive access
  1 = Shared read - No write

Example:

```
Block ABC (Point A100; at Backgrnd)

   Local Status, FM_Status, File_id, Num_Records, Buffer_Size,
&    Acc_Priv
   Local Path: String

%Include_Set File

   Set Path = "Net>S307>Temp307.xx"
   Set Buffer_Size = 132
   Set Acc_Priv = 0

Call File$Open_File (Status, FM_Status, File_Id, Num_Records,
&              Buffer_Size, Path, Acc_Priv)

If Status <> 0 then (send: " File not opened error #",
   Status;
&    goto Finish)

Finish: Exit

End ABC
```

File I/O CL Extensions

File$Open_File Subroutine

- The possible return status values for this function:

  0.0 Good Status

  1.0 Invalid file pathname

  2.0 File exists; but is of wrong type

  3.0 File privilege violation

  5.0 File does not exist

  6.0 Device error while reading file

  9.0 Volume does not exist

  12.0 Other file manager error; check File Manager status

  22.0 Maximum number of files already opened (Max 10)

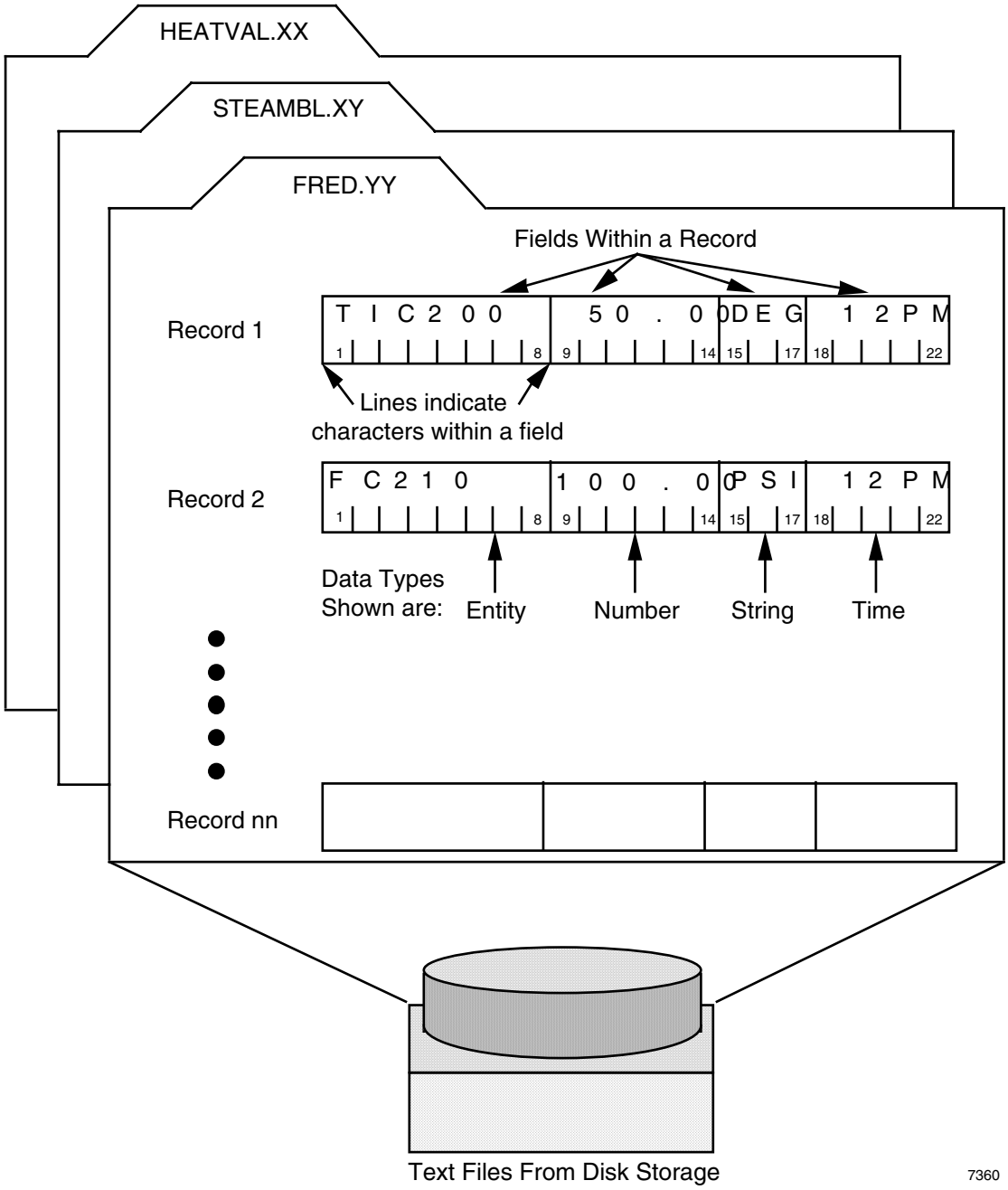
  23.0 Memory limit exceeded

  29.0 File reserved for another operation

  37.0 Invalid file access code

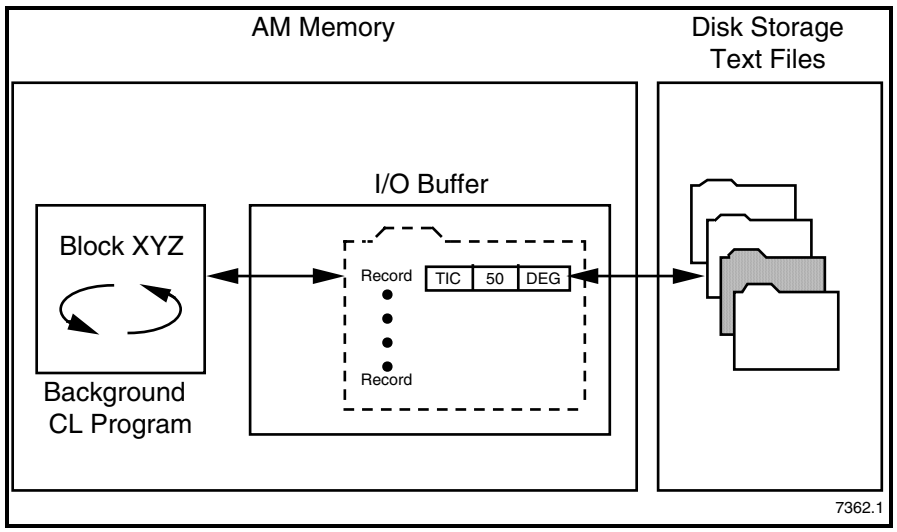
  38.0 Invalid record length

File I/O CL Extensions



File I/O CL Extensions

- A file must be opened before it can be read from or written to. Each time a file is opened, I/O buffer space is allocated for that file in memory.
- Records to be written to a file are created in the files I/O buffer using the File$Put_Field subroutine for each field to be written. Once all fields are in the I/O buffer, a File$Write_Record subroutine writes the record to the file.
- Records to be read from a file are written to the I/O buffer using the File$Read_Record subroutine. Once the Record is in the I/O buffer, a File$Get_Field subroutine reads each field from the I/O buffer into the CL background program.



- Only 10 Files (I/O Buffers) may be opened at any one time. Files should be opened using the File$Open_File subroutine. Once work on the file is completed, the file should be closed with a File$Close_File subroutine.

AMCL02

MATRIX MATH LIBRARY

- The AMCL02 Math Library extension consists of a set of subroutines that are loaded into the AM as external load modules. The Math Library subroutines require that the AM have floating point hardware to perform double precision calculations. This means the AM must use an HMPU processor board.

- The Math Library can be used to:

  Calculate the standard deviation of an array

  Generate random numbers with uniform (0-1) distribution, or with a psuedo-Gaussian (normal) distribution

  Perform matrix operations, such as the following:

  -- Create a two dimension local matrix from a single dimension CDS array

  -- Create a single dimension CDS array from a two dimension local matrix

  -- perform matrix multiply, add, subtract, and transpose operations

  -- Perform matrix inversion and solution with right-hand/left-hand division options

  -- Create, reserve, initialize, and release matrix work areas

  Determine CPU usage of a CL block in milliseconds

AMCL02

MATRIX MATH LIBRARY

- The Matrix Math Library contains the following subroutines:

Subroutine Name	CL/AM Reference Subsection
AMCL02$Standard_Deviation	E.4.2
AMCL02$Uniform_Random_Number	E.4.3
AMCL02$Normal_Random_Number	E.4.4
AMCL02$Get_Time	E.4.5
AMCL02$Subtract_Time	E.4.6
AMCL02$CDS_Array_to_Local_Matrix	E.4.7
AMCL02$Local_Matrix_to_CDS_Array	E.4.8
AMCL02$Create_Work_Area *	E.4.9
AMCL02$Reserve_Work_Area *	E.4.10
AMCL02$Release_Work_Area *	E.4.11
AMCL02$Init_Matrix_Work_Area *	E.4.12
AMCL02$Get_Work_Area_Info *	E.4.13
AMCL02$Get_Matrix_Work_Area_Info *	E.4.14
AMCL02$Get_Matrix_Info *	E.4.15
AMCL02$Create_Matrix *	E.4.16
AMCL02$Put_Element *	E.4.17
AMCL02$Get_Element *	E.4.18
AMCL02$Add_Matrices	E.4.19
AMCL02$Subtract_Matrices	E.4.20
AMCL02$Multiply_Matrices	E.4.21
AMCL02$Transpose_Matrix	E.4.22
AMCL02$Matrix_Inversion	E.4.23

* Math Library subroutines that can be used only by background CL programs.

AMCL02

MATRIX MATH LIBRARY

- The AMCL02$Standard _Deviation subroutine is used to calculate the Mean and Standard Deviation of an array of numbers. It requires the following arguments:

```
SUBROUTINE AMCL02$Standard_Deviation
  Ret_Status          : OUT NUMBER;    -- Math Library return status
  CL_Error_Status     : OUT CLERRSTS;  -- CL error enumeration
  Mean                : OUT NUMBER;    -- Returned mean
  Standard_Deviation  : OUT NUMBER;    -- Returned standard deviation
  Source_Array        : IN  cl_type;   -- Source array name
  Number_of_Items     : IN  NUMBER;    -- Number of items to use
  Starting_Offset     : IN  NUMBER;    -- Starting element offset
  Bad_Value_Flag      : IN  LOGICAL    -- If TRUE, skip any Bad Values;
                                       -- If FALSE, abort on Bad Value
```

Where **Ret_Status** is used to return the Math Library error status. The returned value corresponds to the following status conditions:

0.0 — Successful
1.0 — Bad value error—a bad value was input to, or generated by, the calculation.
2.0 — Bad CL Reference List number—internal error, call Honeywell TAC.
3.0 — Bad input error—an input value is either bad or out of range.
4.0 — CL error—use the CL_Error_Status return value to determine the reason.
9.0 — Bad standard deviation array—fewer than two useable values were found in the source array.
27.0 — HMPU is required

And **CL_Error_Status** returns the CL Error return status enumeration as follows:

NOERROR — No Error
LIMVIOL — Limit Violation
RIGHTS — Rights Error
COMMERR — Communication Error
BADVALST — Bad Value Status
COMABORT — Communication Abort
ABORT — Abort
ARITHERR — Arithmetic Error
ARRAYLIM — Array Limit Violation
RANGE — Range Error
PROGERR — Program Error
KEYLEVEL — Keylevel Error
CNFERR — Configuration Error

AMCL02

MATRIX MATH LIBRARY

- The following CL code fragment calculates the standard deviation of a five element segment of an eight element local CL array.

```
LOCAL status,                  -- Math Library return status
&     start,                   -- Start element offset from lower bound
&     num_data,                -- Number of data items
&     mean,                    -- Returned mean
&     std_dev   : NUMBER       -- Returned standard deviation
LOCAL cl_status : CLERRSTS     -- CL error status
LOCAL bad_val   : LOGICAL      -- Establish bad/infinite value handling
LOCAL data      : NUMBER ARRAY (5..12) -- Data array

%INCLUDE_SET AMCL02

SET start = 2.0  -- Start at the second element in the array (index = 6)
                 -- Start = index - lower bound + 1 (6 - 5 + 1 = 2)
SET num_data = 5 -- Use five elements in the calculation

CALL AMCL02$Standard_Deviation
& (status, cl_status, mean, std_dev, data, num_data, start, bad_val)
```

Given these initial values for "data": [4.5, 6.0, 2.3, 5.2, 5.4, 3.5, 5.5, 2.1]

The values used in the calculation will be: [6.0, 2.3, 5.2, 5.4, 3.5]

The returned value for "std_dev" will be: 1.531992...

Using the formula for Standard Deviation:

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

where s = standard deviation
x = the elements of the array
n = number of elements in the calculation

AMCL03

CONTINUOUS HISTORY ACCESS

- The AMCL03 CL/AM extension consists of a set of subroutines that are loaded into the AM as an external load module. These subroutines permit the AM access to historical data on the History Module

- The Historical Data Access subroutines are used to:

  Read historized data for a single data point

  Specify whether the historized data is read as:

  an absolute snapshot

  absolute average

  or relative snapshot

  Collect history of the following types:

  snapshots

  hourly averages

  shift averages

  daily averages

  monthly averages

  user defined averages

  Get the collection rate (5, 10, 20, or 60 seconds) for a CL history call

  Specify point.parameter values in ASCII format

  Search 8 hour on-line data only, or all data (historized and on-line)

AMCL03

CONTINUOUS HISTORY ACCESS

- AMCL03 Continuous History Access consists of the following subroutines:

Subroutine Name	Subroutine Function
AMCL03$Get_Collection_Rate	Obtains the collection rate
AMCL03$Abs_History	Retrieves snapshot history (absolute time base)
AMCL03$Rel_History	Retrieves snapshot history (relative time base)
AMCL03$Avg_History	Retrieves averages history data (absolute time base)

- AMCL03 is available on Rel. 400 and later.

- AMCL03 may only be used in Background.

AMCL03

CONTINUOUS HISTORY ACCESS

- The AMCL03$Abs_History subroutine obtains snapshot history for a specified point.parameter on the LCN based on absolute begin and end search data and times. It requires the following arguments:

```
SUBROUTINE AMCL03$Abs_History
 (Ret_Status       : OUT    NUMBER;    -- History access error status
  CL_Error_Status  : OUT    CLERRSTS;  -- CL error enumeration
  Time_Stamp_Array : OUT    cl_type;   --
  Status_Table     : OUT    cl_type;   --
  Values           : OUT    cl_type;   --
  Num_of_Values    : IN OUT NUMBER;    --
  Sample_Rate      : IN     NUMBER;    --
  Scope_of_Search  : IN     NUMBER;    --
  Begin_Date_Time  : IN     TIME;      --
  End_Date_Time    : IN     TIME;      --
  Historized_Param : IN     cl_type)   -- Point.Parameter identification
```

Where **Ret_Status** is used to return the History Access error status. The returned value corresponds to the following status messages:

0.0 — Ok_cl_hstat	— No Errors
1.0 — Param_cl_hstat	— Error in Parameters
2.0 — Complete_With_Item_Errors	— Some items contain errors
3.0 — Busy_cl_hstat	— System busy, request denied
4.0 — Failure_cl_hstat	— Could not complete request
5.0 — Indeterminate_cl_hstat	— An error has occurred
6.0 — Errors_cl_hstat	— An error has occurred
7.0 — DA_cl_hstat	— Data Access Error
8.0 — memory_cl_hstat	— A memory error has occurred
9.0 — Conversion_cl_hstat	— Error Converting String
1000+ — Illogical_error_cl_hstat	— Contact Honeywell TAC

And **`CL_Error_Status`** returns the CL Error return status as follows:

NOERROR	— No Error	LIMVIOL	— Limit Violation
RIGHTS	— Rights Error	COMMERR	— Communication Error
BADVALST	— Bad Value Status	COMABORT	— Communication Abort
ABORT	— Abort	ARITHERR	— Arithmetic Error
ARRAYLIM	— Array Limit Violation	RANGE	— Range Error
PROGERR	— Program Error	KEYLEVEL	— Keylevel Error
CNFERR	— Configuration Error		

- Refer to CL/AM Reference Manual, Appendix D for subroutine detail.

AMCL03

CONTINUOUS HISTORY ACCESS

Time_Stamp_Array — An array of variables of the data type Time that represent the time stamps associated with each history value. Even when the values array contains a Bad Value, there will be an associated time stamp with a status for that element.

Status_Table — An array of status values for each history value. This array should be used to verify the data in the values array and in the time stamp array. Note that a status of Normal Data can be accompanied by a Bad Value (see heading D.5.1) in the corresponding Values array.

0.0 — Normal_Data	— The value is analog
1.0 — Digital_Data	— The value is digital
7.0 — Time_Change	— A time change has occurred
8.0 — Missing	— This record is missing

Values — This parameter presents the returned values in an array. It is possible for a Bad Value to be present in this array. If a Bad Value is present, check the associated status table. A Bad Value will appear any time that the associated status value is greater than 1.0.

Num_of_Values — Specifies the number of values expected. This number allows you to control how much space needs to be set aside for the data. If the actual number of returned values is less than the specified number, the actual number returned will be substituted. If there is more data than allowed for by this number, the excess data is ignored. The value of this parameter can range from 1 to 262.

Sample_Rate — The rate at which Snapshot samples are to be collected (5, 10, 20, and 60 are the only valid inputs). Note that if the snapshots were taken and stored at a slower rate than the sample rate specified, the values array will contain a Bad Value for each missing record.

Scope_of_Search — This number represents the area to search for the requested data.

0.0 — All data
1.0 — On-line data only

Begin_Date_Time — This parameter specifies the start time for the search.

End_Date_Time — This is the end time for the absolute history search.

Historized_Param — Identifies the desired point.parameter. This parameter can be expressed in ASCII form, internal form, or as a CDS variable of type entity.

AMCL03

CONTINUOUS HISTORY ACCESS

Examples:

```
LOCAL status,                              -- return status
&     size,                                -- number of values requested
&     rate      : NUMBER                   -- sample rate
LOCAL cl_status : CLERRSTS                 -- CL error status
LOCAL beg_time,                            -- begin time
&     end_time  : TIME                     -- end time
LOCAL values,                              -- requested values array
&     statuses  : NUMBER ARRAY (1..262)    -- status array
LOCAL times     : TIME ARRAY (1..262)      -- timestamp array

%INCLUDE_SET AMCL03

SET size = 262
SET beg_time = (Date_Time) - 150 HOURS
SET end_time = (Date_Time) - 120 HOURS

CALL AMCL03$Get_Collection_Rate
&          (status, cl_status, rate, point.pv)  -- get the sample rate

CALL AMCL03$Abs_History
&          (status, cl_status, times, statuses, values, size, rate, 0,
&           beg_time, end_time, point.pv)
```

This example will obtain all snapshots between 150 hours ago and 120 hours ago for point.pv at the same rate that the history data was collected.

AMCL01

CDS_MOVE AND MULTIPLE_MOVE_PARAMETER

CDS_MOVE

- The CL subroutine "Custom Data Segment Move" (CDS_Move) provides a method by which a CL/AM program can copy the contents of a CDS (on a source AM point) to another CDS with the same structure (on a destination AM point).

- The source and destination CDS can be on the same or different AMs on the local LCN, and neither is required to be on the Bound Data point.

- The CDS_Move subroutine can be used only by CL Blocks linked to the Backgrnd insertion point.

- The CDS_Move subroutine requires the following arguments:

```
Subroutine AMCL01$CDS_Move
    Return_Status      : OUT Number;     -- CDS_Move return status
    Det_Status         : OUT Number;     -- Detailed return status
    Dest_Entity        : IN CL_Type;     -- Destination point name
    Dest_Package       : IN String;      -- Destination package name
    Source_Entity      : IN CL_Type;     -- Source point name
    Source_Package     : IN String;      -- Source package name
```

AMCL01

CDS_MOVE

Example:

```
Package

Custom

Parameter SRC_PNT : $Reg_cntl       -- Entity Type
Parameter DEST_PNT      : String          -- May be Entity or String Type

End Custom

Block Move_CDS(Point ABC; at Backgrnd)

%Include_Set AMCL01

Local src_pack, dest_pack     : string
Local status, det_status      : number

Set src_pack = "CDS1"
Set dest_pack = "CDS2"
Set DEST_PNT = "A100_LongTag"

Call AMCL01$CDS_Move (status, det_status, DEST_PNT, dest_pack,
&  SRC_PNT, src_pack)

End Move_CDS

End Package
```

RESULTS:

Values of the parameters in "CDS1" (src_pack) on point referenced in SRC_Pnt are copied to "CDS2" (dest_pack) on point A100_LongTag (DEST_PNT).

AMCL01

MULTIPLE_MOVE_PARAMETER

- The Multiple_Move_Parameter subroutine provides the capability of moving a variable number of values in a single call. Additional subroutines provide tools for defining lists of parameters to be moved with one call.

- The significant features of this set are:

  Capability of moving a variable number of values in a single call in a BACKGRND CL/AM block.

  Allows moves from point.parameters to point.parameters

  Allows moves from local CL variables to point.parameters and vice versa.

  Allows moves from/to point.parameters through Network Gateway.

  Ability to move entire arrays and array segments.

  Ability to move string arrays.

  Ability to specify point.parameters in ASCII format (which results in parameter indirection capability).

  Ability to set lists permanent.

  Ability to inactivate list items.

  Ability to define on-line, the maximum size of memory for list structures.

AMCL01

MULTIPLE_MOVE_PARAMETER

Subroutine Name	Function
AMCL01$Allocate_list	Allocate an entry in the List Summary, allocate memory for the list, and initialize the list with empty items.
AMCL01$Get_List_Id	Return the ASCII identifier for a particular list.
AMCL01$Modify_Entry	Add or modify a list item.
AMCL01$Modify_Complex_Entry	Same as Modify_Entry, but adds ability to define array segments and define point.parameters in ASCII format.
AMCL01$Multiple Move Parameter	Execute the move of items in a source list to items in a destination list.
AMCL01$Set_Permanent_List	Set a list "permanent" or "not permanent".
AMCL01$Deallocate_List	Delete a list.
AMCL01$Get_List_Mem_Stats	Obtain maximum amount of memory available for list configuration and the currently used memory.
AMCL01$Change_List_Mem_Size	Change the maximum size of user memory dedicated to list storage.
AMCL01$Set_List_Item_Active	Activate/inactivate a particular list item.
AMCL01$Get_Exec_Time_Stats	Obtain statistics on execution time of Multiple Move function.

MULTIPLE_MOVE_PARAMETER

Example:

```
External PT1   -- source point
External PT2   -- destination point

Local loc_data : NUMBER ARRAY (1..300)
Local error, err_code, L1_nb, L2_nb    : NUMBER
Local da_error : CLERRSTS ARRAY (1..2)

%INCLUDE_SET AMCL01

----------------- allocate source and destination lists ----------------------

Call AMCL01$Allocate_List (err_code,      -- return status
&                          L1_nb,         -- return internal list id
&                          2,             -- number of items
&                          "Source"       -- list id of Source list
If err_code <> 0 then ..........

Call AMCL01$Allocate_List (err_code,      -- return status
&                          L2_nb,         -- return internal list id
&                          2,             -- number of items
&                          "Dest"         -- list id of Source list
If err_code <> 0 then ..........

------------------------ fill in the lists --------------------------------

Call AMCL01$Modify_Entry (err_code, 1, L1_nb, "Source", Pt1.Data)
If err_code <> 0 then ..........

Call AMCL01$Modify_Entry (err_code, 2, L1_nb, "Source", loc_data)
If err_code <> 0 then ..........

Call AMCL01$Modify_Entry (err_code, 1, L2_nb, "Dest", Pt2.Data1)
If err_code <> 0 then ..........

Call AMCL01$Modify_Entry (err_code, 2, L2_nb, "Dest", Pt2.Data2)
If err_code <> 0 then ..........

---------------------------------------------- Multiple move ----------------------------------------------

Call AMCL01$Multiple_Move_Parameter (error, da_error, L1-nb, "Source", L2_nb,
&                                    "Dest")
If error <> 0 then send: "Error #", error, ", Data error #", da_error
```

AMCL04 / AMCLO5

AMCLO4

AM EXTENSION FOR FAST EXTERNAL CDS FETCH

- This extension contains no CL subroutine calls. It is available on R401.
- When loaded in an AM, this extension allows other modules on the LCN to have faster access to Custom Data. This provides for more CPU freetime for Background CL programs.
- Typical access times without this function can run from 0.6 to 3.0 milliseconds. With this function access times are reduced to 0.3 milliseconds.

AMCLO5

AM EXTENSION FOR OFF-LOGICAL-NODE ACCESS

- This extension contains no CL subroutine calls. It is available on R401.
- Allows users, in foreground CL, to fetch string array elements from any on-physical-node point. Without this extension, string arrays may be fetched only from on-logical-node points (points in the same unit).
- An other feature of this extension allows for indirection through off-logical-node points. Off-physical -node references may only occur at the last level of indirection.