PLANTSCAPE SERVER

APPLICATION PROGRAMMING TOOLS

TABLE OF CONTENTS

SESSION OBJECTIVES

At the end of this section of the course the student will be able to:

- Compile and Link PlantScape application modules
- Identify coding templates for tasks and utilities
- Issue messages to the \data\loga file from an application
- Apply API routines to read from and write to the Server database
- Issue messages to the Operator Message Zone on a PlantScape Station
- Generate Alarm Summary entries from an application
- Start a task timer
- Start and reset a Watchdog timer entry
- Execute an NT command from an application
- Identify the method of producing a user written scan task

REFERENCES

Knowledge Builder: Application Development Guide

Compiling and Linking

Compilers, Linkers and Editing

The recommended compilers for developing applications are:

- Microsoft Visual C++ Version 4.0 (SCAN 3000 R620.3 only), or
  Version 5 (SCAN 3000 R620.3 or PSc R130),
  Version 6 (PlantScape R200, R300)

- Microsoft FORTRAN Power Station 32 Version 1.0 (SCAN 3000 R620.3 only), or
  Digital Fortran Version 5.0 (PSc R130, R200, R300)

For whichever language you use, the appropriate application should be installed and the environment variables setup.
For example, if Visual C++ is installed in C:\MS then the path environment variable should include C:\MS\BIN, INCLUDE environment variable include C:\MS\INCLUDE and the LIB environment variable include C:\MS\LIB.

The Windows NT username "engr" should be used for application development so that applications created have permission to access the PSc Server database.

Editing of source files can be achieved using any of the editors that come with Windows NT (for example, Notepad) or the programs listed above.

Compiling

After a source code file has been created, it can be checked, compiled and linked with the application library.
To compile a FORTRAN source code file:

`fapi` *`name`* `[`*`options`*`]`

To compile a C source code file:

`capi` *`name`* `[`*`options`*`]`

where:

name is the name of the FORTRAN or C source code file without the ".f" or ".c" extension

options are the standard compiler options defined in your compiler reference manual.

In either case, if the compilation is successful an object file, *name*.obj, will be created.

Compiling and Linking…..continued

Linking

To link the object file created from either FORTRAN or C source code with the application library:

`lapi` *`name`* `[`*`dll option`*`] [`*`options`*`]`

This will result in the generation of an executable file called “name.exe”.

where:

name is the name of your application without the “.exe”.

dll option is:

for a C program use /cdll, or
for a Fortran program use /fdll
This may be required depending on when your system was delivered.
Contact GTAC for advice on this assistance on this.

options are the standard linker options.

Attention
The resulting executable file must be copied to the honeywell\server\run directory.

Tasks and Utilities

Tasks

A task is a memory resident program that can pause execution and wait to be requested.
They each have an associated LRN (Logical Resource Number, a PSc Server reference) and a PID (Process ID, a WinNT reference).
Tasks that have been started can be listed with the **ps** command and in **Task Manager**.

Once a task executable has been created (by compiling and linking) it can be started in either of three ways:

- Using the **addtsk** command:

  addtsk *name lrn* [*priority*]

- Using the **ct** command:

  ct *lrn priority* -efn *name*

- Choose **Configure→Applications→Summary** on Station and enter the tasks and LRNs accordingly. Next time the PSc Server is started the applications will be started automatically.

Once a task has been started it can be activated in many ways, for example a push button on a custom graphic page.

Tasks can be deleted (assuming thay include no bugs that prevent them from terminating; see page 9) in one of two ways:

- Using the **remtsk** command:

  remtsk *lrn*

- Using the **dt** command:

  dt *lrn*

Tasks that will not terminate and, thus, cannot be deleted by either of the above two methods, can only be “deleted” by:

- using the **EndProcess** function in **Task Manager**

- by using the command:
  kill *PID*

  where PID for the required task can be obtained from **Task Manager**, or from the **ps** command.

continued on next page

Tasks and Utilities…..continued

Utilities

A utility is a program executed from the command prompt, for example, **lisscn** and **fileio**.
They do not have associated LRNs.
Utilities do not terminate and stay memory resident.
What reasons can you give for coding an application as a utility and not a task?

Essential API Routines

gbload	This is the API routine which attaches the Server database to the data area of an application. It's successful execution is essential and need only be called once.
trmtsk	This routine terminates a task or utility completely.
trm04	This is the terminate - stay resident call. It is essential for a task and cannot be used in a utility.
getreq	This is the routine to get values from the parameter block. Parameter block values can be specified from a display pusbutton and other techniques. Although you may not wish for parameters to be passed to your application it is still essential for a task.

General form of task

```
if (c_gbload() == HSC_ERROR)

{
        c_logmsg(progname, "205", "common load error - %x", errno);
        c_trmtsk(ZERO_STATUS);    /* error - terminate */
}

while (1)

      {

       c_getreq(&prmblk);
       if (prmblk.param1==0)

       {
        c_trm04(0);
        }

      else

       {
        **** Perform some Function ****
       }

    }

exit(0);
```

Sending Messages to \data\loga

logmsg

This is the routine to send messages or diagnostics to the loga file.

Exercise

Now that we have the essential routines and an output routine, let's do an exercise.

Step	Action
1	Make a copy of file ex1.c with filename ex1#.c
2	Edit ex1#.c to output a log file message which identifies your team number.
3	Edit the program identifiers to **`ex1#.c`**
4	Compile and link the code in file ex1#.c
5	Copy the executable into the \run directory
6	Assign LRN 12#
7	Start the task
8	Add a pushbutton to team#.dsp to request the task
9	Monitor the log file to ensure: 1. no gbload error 2. correct log file message produced

Reading and Writing Point Values

getpnt	Gets the point type and number from the point name.
getval	This family of routines get point values. A getpnt call is usually used to obtain the point type/number. **int c_getval_numb(point, param, value)** **int c_getval_asci(point, param, string, stringlen)** **int c_getval_hist(point, param, offset, value)**
putval	This is the corresponding family of routines to output to a point. In general they are used to send new values to point parameters with destination addresses located within a controller. However, a PV can be changed if necessary. What would be a prerequisite for this ?
ppv	The **ppv** routine can be used to send a value to a PV and invoke point processing (since, in this case, the point does not need to be off scan). This should be used if there is an algo attached to the point.
CDA points	The routines **hsc_point_number**, **hsc_param_number**, **hsc_param_values**, and **hsc_param_value_put** must be used in place of **getpnt**, **getval** and **putval** for CDA points. These functions are equivalent except **hsc_param_number** which gets the parameter number of the SP, OP, and so on.. When using **getval** and **putval** these numbers are specified by the include file parameters.
Exercise:	Copy ex2.c to ex2#.c Edit ex2#.c to modify the point_IDs to LT10# and U#RECIPE Assign it to LRN 13# Add a pushbutton to team#.dsp to request the task What conditions will cause the task to produce an error message?

Reading and Writing Server database records

dataio

This is the family of routines to access the database by logical file and record number.
The specific routines and their functions are as follows:

c_dataio_open
opens a server file.

c_dataio_close
closes a server file.

c_dataio_read and c_dataio_write
operate on data in a specified location.
Location values are constructed by OR-ing flags from the following list:

LOC_ALL, LOC_MEMORY, LOC_DISK, LOC_LINK1, LOC_LINK_2.
The recommended configuration to use is LOC_ALL.

c_dataio_size
returns the size of a server file.

c_dataio_read
reads a record from a RELATIVE server file into an integer buffer.

c_dataio_write
writes a record from an integer buffer to a server file.

c_dataio_read_blk
reads a number of records from a RELATIVE server file into an integer buffer.

c_dataio_write_blk
writes a number of records from an integer buffer to a server file.

c_dataio_queue
queues (writes plus increment pointers) a record from an integer buffer to the top (newest record) of a CIRCULAR server file.

c_dataio_dequeue
dequeues (reads plus decrement pointers) a record from the bottom (oldest record) of a CIRCULAR server file into an integer buffer.

c_dataio_read_newest
reads a record relative to the top of a CIRCULAR server file into an integer buffer and returns the relative record number for use in subsequent writes. This call is equivalent to c_dataio_read.

Reading and Writing Server database records…..continued

dataio **…continued**	**c_dataio_read_oldest** reads a record relative to the bottom of a CIRCULAR server file into an integer buffer and returns the relative record number for use in subsequent writes. **c_dataio_write_newest** writes a record from an integer buffer to a CIRCULAR server file record relative to the top.
dblock	This routine provides advisory locking. Advisory locking means that the tasks that use the file take responsibility for setting and removing locks as needed. Record locking can only be performed on memory resident relative files which have been setup to enable record locking.
dbunlock	This routine clears the lock advice.
Exercise:	Copy ex3.c to ex3#.c Edit ex3#.c to modify the User Table number 1#. Assign to LRN 14#. Add alphanumeric object(s) to team#.dsp to display: • UTBL1#, Rec 3, Word 1 (data entry allowed), and • UTBL1#, Rec 4, Word 1. What is the best way to do this? Use REAL2 format. Why? Display with two decimals. Add a pushbutton to team#.dsp to request the task. Enter a value into Rec 3 and request the task. Ensure that the correct results are seen in Rec 4 and the log file.

Sending Operator Messages

oprstr

Send a string to the message zone.
The general forms are:

int c_oprstr_info(crt, message)

int c_oprstr_message(crt, message)

int c_oprstr_prompt(crt, message, param1)

char *c_oprstr_response(crt, prmblk)

---

c_oprstr_message
outputs an invalid request message that is cleared after a TIME_RQUEST seconds (defined in system).

c_oprstr_prompt
outputs an operator prompt and sets the <Enter> key to notify the calling task with the specified parameter 1.
After calling this routine the task should branch back to its GETREQ call to service its next request, and if none exists, the terminate with a TRM04.
When the operator types a response and presses <Enter>, the task is requested with the specified parameter 1, and should branch to code that calls **c_oprstr_response** to fetch the response.

c_oprstr_response
reads the entered data and clears the prompt.
Upon successful completion c_oprstr_response will return a pointer to a null-terminated string containing the response from the operator.
Upon successful completion c_oprstr_info, c_oprstr_message and c_oprstr_prompt will return zero.
Otherwise, a NULL pointer or -1 will be returned, and **errno** is set to the following error code:

[M4_INVALID_CRT] An invalid Station number was specified.

Exercise:

Copy ex4float.c to ex4#float.c

Edit ex4#float.c to request and enter a new SP value for LT10#.

Assign to LRN 17#.

Copy ex4text.c to ex4#text.c

Edit ex4#text.c to request and verify a Point Name.

Assign to LRN 18#.

Add pushbuttons to team#.dsp to request the tasks, and test them.

Alarm Summary entries

almmsg

ALMMSG is used to send the specified text to the alarm system for storage into the alarm and/or event file, and printing on all printers.

c_almmsg_event will send the text to all printers and log the text to the event file.

c_almmsg_alarm will send the text to all printers and log the text to the alarm list and event file.
It also sets the first character of the level field of the alarm to either 'L', 'H' or 'U' depending on the value of priority.

The priority of the alarm is defined as follows:

ALMMSG_LOW	Low priority
ALMMSG_HIGH	High priority
ALMMSG_URGENT	Urgent priority

c_almmsg_format
will format up an alarm message given all the relevant fields.
It returns a pointer to a null-terminated string that can then be passed onto **c_almmsg_event** or **c_almmsg_alarm**.

The text string can be broken up into six fields.
The starting character of each field is defined by the following identifiers:

ALMMSG_NAME
Alarm name (equals 0)

ALMMSG_ID
Alarm ID (e.g. PVHI, SVCHG)

ALMMSG_LEVEL
Alarm level (e.g. L, U, H, STN01)

ALMMSG_DESCR
Alarm description.

ALMMSG_VALUE
Alarm value

ALMMSG_UNITS
Alarm units.

Exercise:

Copy ex5.c to ex5#.c

Edit ex5#.c to identify your alarms with your team number.

Assign to LRN 19#.

Add a pushbutton to team#.dsp to request the task.

Request the task and monitor Station for the three alarms.

Timers

tmstrt	TMSTRT starts a timer to request the calling task every cycle seconds. This is equivalent to calling RQTSKB every interval. The arguments param1 and param2 are passed as words 2 and 3 of the ten word parameter block, to the task each interval. These parameters can be accessed by calling GETREQ. **c_tmstrt_single (cycle, param1,param2)** requests the specified task only once. **c_tmstrt_cycle (cycle, param1,param2)** requests the specified task continuously every cycle.
tmstop	TMSTOP stops the timer specified by the argument tmridx. This index corresponds to the return value of TMSTRT.
Question	Why would you use timers rather than algorithm 16?

Watchdog Timer

wdstrt

WDSTRT enables a watch dog timer for the calling task.

The general form is:

c_wdstrt(timeout, mode)

Calling this routine allocates an entry in the WDTTBL.
Each second, WDT decrements the timer entry.
If the timer becomes zero then the action defined by the mode will be taken.
The modes available are the following:

WDT_MONITOR
monitor timer entry only.

WDT_ALARM_ONCE
generate an alarm on first failure only.

WDT_ALARM
generate an alarm on each failure.

WDT_RESTART_TASK
restart task on first failure, reboot the server system on second failure.

WDT_RESTART_SYS
restart the server system on failure.

wdon

WDON is used to prevent the watch dog timer from timing out the calling task.

The general form is:

c_wdon(wdtidx)

where wdtidx is the watchdog timer index.

Exercise:

Edit ex6.c to modify the Point ID to LT10#.

Assign the task to LRN20#.

Do you need to add a pushbutton the team#.dsp to request the task?

Monitor the log file for the correct output.
What do you expect to see?

Display the Applications Timer Table and set the Reset Period for LRN20# to 0.
This will cause the task to NOT be requested again by the Task Timer.
Monitor the Applications Watchdog Timers and note that the Watchdog Timer for LRN20# will not reset when it reaches 60, and an error will be produced

when it reaches 0.

Executing an NT command

ex

EX passes a command line string as input to the command line interpreter and executes it as if the command line was entered from a Command Prompt.
If successfully completed a value of 0 is returned.
Otherwise, -1 is returned and **errno** is set to an error code depending on the command line executed.

The **exscript** application which you have used employs this routine.
It also uses a routine called **getlrn** to determine its own LRN.

Exercise:

Delete LRN 12#

Modify ex1#.c to delete files named “rpt0#*”.

> **Attention**
>
> The C language requires the following syntax:
>
> ```
> c_ex("del c:\\honeywell\\server\\report\\rpt0#*");
> ```

Compile and link the modified task.

Re-assign it to LRN 12#.

Request a valid Report in the range 0#1 to 0#9.

Check with Explorer for any files named “rpt0#*”.

Request the task and check with Explorer that there are no files named “rpt0#*”.

User written Scan Tasks

Why a "user-written" scan task ?

To introduce unsupported PLC-like devices into your system, use the User Scan Task option to write an application which provides an interface between the device and the PlantScape system.

The link between the server and the User Scan Task is the PlantScape database User Tables.
The server provides database scanning software, **dbscn.exe**, to scan data from the User Tables into server points.
The User Scan Task reads data from the remote device and writes it into the User Tables.
PlantScape can also send controls to the remote device by way of the User Scan Task.

The option works by using channels defined as "User Scan Task" type.
These channels operate in exactly the same manner as a conventional channel. They interact, however, with "User Scan Task" controllers rather than physical controllers.

Point parameters will be sourced from these database controllers by specifying the word address within the specific record.
Refer to the *Application Programmers Guide* and *Quick Builder online help* for details on configuring this type of channel.

How do you do controls ?

The gdbcnt routine provides for this.
The LRN of the task doing the control must be specified when the channel is built.