HITACHI
Inspire the Next

Programming ivy - reference

July 6, 2016

Allart Ian Vogelesang ian.vogelesang@hds.com +1 408 396 6511

Hitachi Data Systems

Two sections in this material

HITACHI
Inspire the Next

1. ivyscript programming language wrapper – for scripting test workflow
   - Similar to a subset of C/C++, with some minor differences.
   - Extensible - parser auto-generated from language grammar. (flex+bison)
   - Longer term - examine pros/cons of making part 2 a CLI that you could use in your favourite scripting language, if so, the current ivyscript programming language wrapper would be dropped.
     - Current idea for this is to transform ivy engine control statements into RESTful API calls in the context of "sessions".
2. ivy engine control statements
   - Operating the underlying ivy engine

HITACHI
Inspire the Next

The ivyscript programming language wrapper

Hitachi Data Systems

ivyscript programming language

HITACHI
Inspire the Next

- Statements in the programming language end with a semi-colon, like C / C++ / Java.
- C style comments are supported
  - The part from /* to */ is ignored
- C++ style comments are supported
  - From // to the end of the line is ignored.
- # style comments are supported
  - From # to the end of the line is ignored.

Make .ivyscript programs executable

HITACHI
Inspire the Next

- So-called "sha-bang" lines work.
- A sha-bang line is when you start a script with a line that specifies a path to the program used to interpret the script.
- For example, as the first line of an .ivyscript program:

```
#!/path_to_ivy_executables/ivy
```

(followed by remainder of ivyscript program)

- Then you can invoke the .ivyscript file as a program itself.

Nested blocks

HITACHI
Inspire the Next

- Anywhere you can put a statement, you can put a nested block, which starts with "`{`" and ends with "`}`".
- Any variable or function declarations made inside a nested block are not "visible" to code outside the nested block.
- Nested blocks are typically used in `if` statements, looping constructs, etc.

Types

HITACHI
Inspire the Next

- There are 3 types: `int`, `double`, and `string`.
- Examples of constants, also called literals:

```
int:      0         -5     12345
double:   5.        .5     5.5      5E-2     5%   5.5%
string:   "house"          ""
```

  - There is also a hex form of `int` literal (constant)
    - `0x0` to `0x7FFFFFFF`
    - The hex form of `int` literal only supports non-negative values
  - `5%` means the same thing as `0.05`.

More on string literals

HITACHI
Inspire the Next

- To include a double quote character in a string constant, escape it with a backslash:
  - `"the word \"house\" is double-quoted"`
- Other escaped characters: `\r, \n, \t`
- An escaped octal character value has 3 digits, e.g. `\001`
- An escaped hex character value has one or two digits, e.g. `\xf` or `\x0f`

Identifiers

HITACHI
Inspire the Next

- "identifiers" are eligible to serve as the name of a variable or function.
- An identifier begins with an alphabetic character (a letter), and continues with letters, digits, and underscore _ characters.

Statement – variable declaration

HITACHI
Inspire the Next

- <type> <list of identifiers with optional = <initializer expression>>;
- Examples:
- ```
  int i, j;
  int k = -1;
  double c;
  double d = 1.5;
  string s;
  string name = "bert";
  ```
  ```

Expressions

HITACHI
Inspire the Next

- A constant (a literal of one of the types) is an expression
  - e.g. `"constant"`
- A variable reference is an expression
  - e.g. `x`
- Expressions may be combined together with operators, which operate the same as in C/C++:
  - +, -, *, /, %, >, <, >=, <=, ==, !=, =, |, &, ^, &&, ||
- Expressions have a type, `int`, `double`, or `string`

Converting an expression to a different type

HITACHI
Inspire the Next

- int(<expression>)
  double(<expression>)
  string(<expression>)
- Some times called a "cast".
- The expression is evaluated and the result is converted to the target type.
- Can result in a run time error
  - E.g. evaluating `int("cow")` would cause a run-time error.

Operators - arithmetic

HITACHI
Inspire the Next

- \+ plus – for numbers, adds, for strings, concatenates
  \- minus
  * multiply
  / divide
  % remainder from integer division

Logical operators - comparison

HITACHI
Inspire the Next

- `>` greater than
  `<` less than
  `>=` greater than or equal to
  `<=` less than or equal to
  `==` equal to
  `!=` not equal to

- There is no true/false type. Logical operators evaluate to an integer value just like the old C language before there was a `bool` type.
  - An `int` or a `double` value used as a logical expression means "false" if the numeric value is zero, and means "true" for any non-zero value.
  - Use of `int` logical values is transparent – it works the way you expect it to.

Bitwise or, bitwise and, bitwise exclusive or

HITACHI
Inspire the Next

- The bitwise operators operate on the individual bits in an `int` value, exactly like in C/C++.

- `|` bitwise or
  `&` bitwise and
  `^` bitwise exclusive or

Logical or, and, not

HITACHI
Inspire the Next

- These operate on logical expressions, which evaluate to an `int` interpreted to mean "false" if the `int` value is zero, "true" otherwise.
  - Like in C/C++ the second expression after the operator is not evaluated if the result is known from evaluating the first expression before the operator.
- || logical or
  - Evaluates the first expression, and if true, returns true. Otherwise, it evaluates the second expression and returns its true/false value.
- && logical and
  - Evaluates the first expression, and if false, returns false. Otherwise it evaluates the second expression and returns its true/false value.
- ! not
  - Evaluates a logical expression and returns the opposite.

Assignment expression

HITACHI
Inspire the Next

- <identifier> = <expression>
- The identifier is looked up in the symbol table at compile time, and if it's valid, at execution the expression is evaluated and the variable is set to that value.
- If the expression is not of the same type as the variable, the value may be coerced / converted, or in some cases a compile time error occurs.

Function call expression

HITACHI
Inspire the Next

- <identifier> (<comma separated list of zero or more expressions>)
  - E.g. `sin(.5)`
- Identifier and parameter list signature are looked up at compile time to and if valid, a function call is built.
- At run time, the expressions are evaluated and the resulting parameter values are passed to the function, the function is executed, and the result is returned.

Operator precedence

HITACHI
Inspire the Next

- Same as C/C++
  - `if ( 3*4+5*6 == fred || ! Person == Nancy = 4)`
  - Means
  - `if ((((3*4)+(5*6))==fred)||(!(Person==(Nancy=4))))`
- If you are not sure, group with parentheses ().

User-defined functions

HITACHI
Inspire the Next

- E.g.

```
int add_three( int i )
{
    return i+3;
};
```

Semicolon needed for function definitions, unlike C/C++

- Functions have a type, which is the type of the object they return to the caller.
- Functions can be "declared" without being defined yet:

```
int add_three(int i);
```

Function overloading

HITACHI
Inspire the Next

- It’s OK to have different functions with the same name as long as the sequence of types of the parameters is different so the compiler can tell them apart.

- ```
  int    addtwo(int i)    { return i+2;}
  string addtwo(string s) { return s + "two";}
  ```
  ```

Ivy-specific builtin functions

HITACHI
Inspire the Next

- `string outputFolderRoot();` — from `[OutputFolderRoot]` statement – default `"."`
- `string testName();` — root part of ivyscript file without `.ivyscript` suffix
- `string masterlogfile();` — you can `log(masterlogfile(), "message\n");`
- `string testFolder();` — root folder for output from this run
- `string stepNNNN();` — from most recent [Go!], e.g. step0002
- `string stepName();` — from most recent [Go]
- `string stepFolder();` — subfolder for most recent [go] within testFolder()
- `string last_result();` — for most recent [Go], returns `"success"` or `"failure"`
- `string show_rollup_structure();` — shows type / instance / workload thread hierarchy.

Math builtin functions – same as C/C++

HITACHI
Inspire the Next

- `double sin(double), double cos(double), double tan(double)`
- `double sinh(double), double cosh(double), double tanh(double)`
- `double asin(double), double acos(double),`
  `double atan(double), double atan2(double,double)`
- `double log(double), log10(double),`
  `double exp(double), double pow(double,double)`
- `double sqrt(double)`
- `int abs(int)` - absolute value
- `double pi(), double e()`

String builtin functions

HITACHI
Inspire the Next

- `string substring(string s, int begin_index_from_zero, int number_of_chars);`
- `string left(string s, int n);` like in BASIC, gives you leftmost / rightmost characters
  `string right(string s, int n);`
- `string trim(string s);` removes leading / trailing whitespace
- `string to_lower(string s);`
  `string to_upper(string s);`
- `int stringCaseInsensitiveEquality(string s1, string s2);`
- `string int_to_ldev(int n);` `int_to_ldev(0xFF)` returns `"00:FF"`
- `string to_string_with_decimal_places(double x, int n);`
  `to_string_decimal_places(3.1415,2)` returns `"3.14"`

regex builtin functions

HITACHI
Inspire the Next

- ivy uses the default flavour of C++ std::regex, which I think uses the ECMAscript dialect

- `int regex_match(std::string s, string regex);`
  E.g. `if ( regex_match("horse","(horse)|(cow)") ) then print("animal\n");`

- `int regex_sub_match_count(string s, string regex);`

- `string regex_sub_match(string s, string regex, int n);`
  `n` must be less than `regex_sub_match_count(s, regex)`

- `int matches_digits(string s);`
  `int matches_float_number(string s);`
  `int matches_float_number_optional_trailing_percent(string s);`
  some ivy parameters can be set to these
  `int matches_identifier(string s);`
  alphabetic, continued with alphanumeric and underscores _
  `int matches_IPv4_dotted_quad(string s);`

Accessing csv files – row and column

HITACHI
Inspire the Next

Header row is row -1

Row 0 is test step 0 or subinterval 0

Use column number from 0, or say "Overall IOPS"

Test step csv files (not shown) have one line per subinterval (both host & subsystem data)

Summary csv files like this one have one line per test step.

	A	B	C	Q	R	S	AW	AX
1	Test Name	Step Number	Step Name	iogenerator type	blocksize	maxTags	Overall IOPS	Overall Decimal MB/s
2	demo9	step0000	iops_max	random_independent	4 KiB	32	2597.16	10.638
3	demo9	step0001	baseline_service_time	random_independent	4 KiB	32	25.81	0.105718
4	demo9	step0002	1.125_x_baseline	random_independent	4 KiB	32	555.849	2.27676
5	demo9	step0003	1.25_x_baseline	random_independent	4 KiB	32	1097.86	4.49682
6	demo9	step0004	1.5_x_baseline	random_independent	4 KiB	32	1486.01	6.08671
7	demo9	step0005	1.75_x_baseline	random_independent	4 KiB	32	1722.35	7.05476
8	demo9	step0006	2_x_baseline	random_independent	4 KiB	32	1898.07	7.77448
9	demo9	step0007	3_x_baseline	random_independent	4 KiB	32	2353.68	9.64067
10	demo9	step0008	4_x_baseline	random_independent	4 KiB	32	2602.77	10.6609
11	demo9	step0009	5_x_baseline	random_independent	4 KiB	32	2601.7	10.6566

AY	AZ	BA
Overall Average Blocksize (KiB)	Overall Little's Law Avg Q	Overall Average Service Time (ms)
4	64.0024	24.6432
4	0.164588	6.37691
4	4.08014	7.34037
4	8.74646	7.96686
4	14.2187	9.56836
4	19.2298	11.1649
4	24.2199	12.7603
4	45.0346	19.1337
4	63.9994	24.589
4	63.9962	24.5978

Csv file builtin functions 1/3

HITACHI
Inspire the Next

- `set_csvfile(string filename);`
  - Loads csv file into a kind of spreadsheet object, if it's not already loaded into memory.
  - You can load multiple csv files and switch back and forth.
  - All subsequent csvfile calls refer to the currently set csvfile.
- `drop_csvfile(string filename);`
  - If you are done with it and you would like to release the space.
- `int csvfile_rows();`
  - Number of rows following the header row.
  - Returns -1 if invalid file or file empty. Returns 0 if there was only a header row.
- `int csvfile_columns_in_row(int row);`
  `int csvfile_header_columns();` same as `csvfile_columns_in_row(-1)`

Csv file builtin functions 2/3 – individual cells

HITACHI
Inspire the Next

- `string csvfile_cell_value(int row, int column);`
  `string csvfile_cell_value(int row, string column_header_text);`
  - You can refer to a column using an int, the column index from zero.
  - You can refer to a column using a string, the column header text.

- `string csvfile_raw_cell_value(int row, int column);`
  `string csvfile_raw_cell_value(int row, string column_header_text);`
  - ivy "wraps" text fields as a formula with a string constant, e.g. `="horse"`
    - This stops Excel from interpreting 1-1 as January 1st, and 00:00 from interpreting as a time.
  - The csv file functions normally "unwrap" csv column values, removing this kind of wrapper or removing simple double quotes surrounding a value, to treat `="horse"`, `"horse"` and `horse` the same
  - Retrieving the raw value give you exactly what was between the commas in the csv file.

Csv file builtin functions 3/3 – headers & slices

HITACHI
Inspire the Next

- `string csvfile_column_header(int col);`
  - Give you the text of the column header

- `string csvfile_column(int col);`
  `string csvfile_column(string column_header);`
  - Gives you a "column slice" of the spreadsheet showing "raw" values.
  - E.g. `"IOPS,55,66,55,44"`
  - Demo number 8 shows iterating through the column slices to write out the transpose of a csv file.

- `string csvfile_row(int row);`
  - Gives you a "row slice" of the spreadsheet showing the "raw" values.
  - E.g. `="random_independent",="4 KiB",32,2601.7`

utility functions

HITACHI
Inspire the Next

- `string print(string), double print(double), int print(int)`
  - Prints the specified value to stdout and then returns that value.
- `int fileappend(string filename, string s)`
  - One way to write output. Does not append a newline to `s`.
- `int log(string filename, string s)`
  - Writes a timestamp prefix before the string, and adds terminating newline if the last line in `s` doesn't already have one.
  - E.g. `log(masterlogfile(),"message");`
- `trace_evaluate(int)`
  - Turns execution tracing on/off. Zero means off, otherwise on.

Builtin functions – `shell_command`

HITACHI
Inspire the Next

- `string shell_command(string)`
  - Executes the shell command and returns its output.
    - Runs as **`root`**. You have been warned.
    - Ivy runs as `root` in our lab because ivy uses ssh to fire up ivyslave and ivy_cmddev on test hosts, and "`root`" has been set up to not require a password to ssh. Ivy may also need to run as `root` to do I/O to raw LUNs – not sure.
    - The only ivy component that definitely requires to run as `root` is the SCSI Inquiry tool, which has the executable that issues "SCSI Inquiry" marked `setuid` as `root`, and thus works for any user.
  - Use `shell_command()` to do almost anything
    - `grep` in an ivy output folder to find a csv file name
    - Get a time or date stamp

Builtin functions – `exit()`

HITACHI
Inspire the Next

- As in
  - ```
    if ( last_result() != "success" )
    {
        print "timed out without making a valid measurement.\n";
        exit();
    }
    ```
    ```

Statements: expression statement

HITACHI
Inspire the Next

- <expression> ;
- Executes the expression and discards the result.

Statements – if / then / else

HITACHI
Inspire the Next

- `if ( <logical expression> ) <statement>`
- `if ( <logical expression> ) <statement> else <statement>`
- <statement> can be a single statement, or it can be a nested block starting with { and ending with }.

```
int x = 1;

if ( x >= 0 ) print( "x is greater than or equal to zero.\n");
else          print( "x is less than zero.\n");

if ( x >= 0 ) { print( "x is greater than or equal to zero.\n"); x = x + 1; }
else          { print( "x is less than zero.\n");                x = x - 1; }
```

Statements – traditional C style for loop

HITACHI
Inspire the Next

- `for ( <initializer expression> ; <logical expression>; <epilogue expression> )`
  `<loop body statement>`

- The initializer expression is run.

- Then the logical expression is evaluated, if false, execution of the statement is complete.

- Otherwise, the loop body statement is run, then the epilogue expression is run, then we loop back to where we will evaluate the logical expression.

Example of traditional for loop

HITACHI
Inspire the Next

```
int i;
for ( i=0; i<10; i=i+1 )
{
      print( "i = " + string(i) + "\n");
}
```

- Note that it's not `for (int i=0; i<10; i++)`
  1. The initializer is an expression, not a statement, so can't declare `i` to be an `int`.
  2. There is no C++ increment operator ++.

Statement – list-style for loop

HITACHI
Inspire the Next

- For <identifier> = { <list of expressions> } statement
- E.g.

```
int i;
for i = { 0, 1, 2 }
     print("i = " + string(i) + "\n");

string s;
for s = { "cat", "dog", "mouse" }
{
     print ( "A " + s + " has four legs.\n");
}
```

Statement – while loop

HITACHI
Inspire the Next

- `while` (<logical expression>) <loop body statement>
- The logical expression is evaluated, and if false, execution of the statement is complete.
- Otherwise, the loop body statement is executed and then we loop back to evaluating the logical expression again.

Statement – do - while loop

HITACHI
Inspire the Next

- `do` <loop body statement> `while` (<logical expression>) ;
- The loop body statement is executed, and then the logical expression is evaluated, and if the result was "false", execution of the statement is complete.
- Otherwise, and then we loop back to running the loop body statement again.

HITACHI
Inspire the Next

Operating the ivy engine

Hitachi Data Systems

The "test name"

HITACHI
Inspire the Next

- When ivy is invoked on the command line like
  - `ivy some/path/henri.ivyscript`
- The **`henri`** in `some/path/henri.ivyscript`, the part of the ivyscript filename discarding the path and the .ivyscript suffix, is called the "**test name**".
- It's used as the subfolder name off of the `[OutputFolderRoot]` folder.

"test name" – used in output filename prefixes

HITACHI
Inspire the Next

- The test name is also used as part of the prefix of ivy output filenames.
  - So that you can combine together in one folder any files from multiple ivy runs and there wouldn't be name collisions as long as the test names were different.
  - So that if you threw all the output files in one folder they would sort on nicely on filename, grouping like they were grouped in the original folders.
  - So that if all you get is the file, you still knew which folder it came from.

Statements – [OutputFolderRoot]

HITACHI
Inspire the Next

- `[OutputFolderRoot] <string literal>;`
  - Specifies a root folder which must already exist.
  - The default is `"."` (the current folder).
  - Specifies the root folder in which ivy will make a subfolder to record the output from running an `.ivyscript` program.
- A string literal (string constant) is required, because the output root folder name is captured at compile time.
  - This way, the output folder structure and log files can be all in place before the ivyscript program starts running.
  - At most one `[OutputFolderRoot]` statement, anywhere in your program.

Statement – `[Hosts]`

HITACHI
Inspire the Next

- `[Hosts]` <list of hosts> `[Select]` <select spec> ;
- Forms of specifying test hosts:
  - <string expression for ivyscript_hostname>
    - E.g. `"sun159"` [must look like an identifier]
  - <dotted quad - *not in quotes*>
    - E.g. 192.168.1.1
  - <starting hostname> `to` <ending hostname or number>
    - Shorthand for a series of hostnames with numeric suffixes.
    - E.g. `"cb16" to "cb31"` or just `"cb16" to "31"` or even `"cb16" to 31`

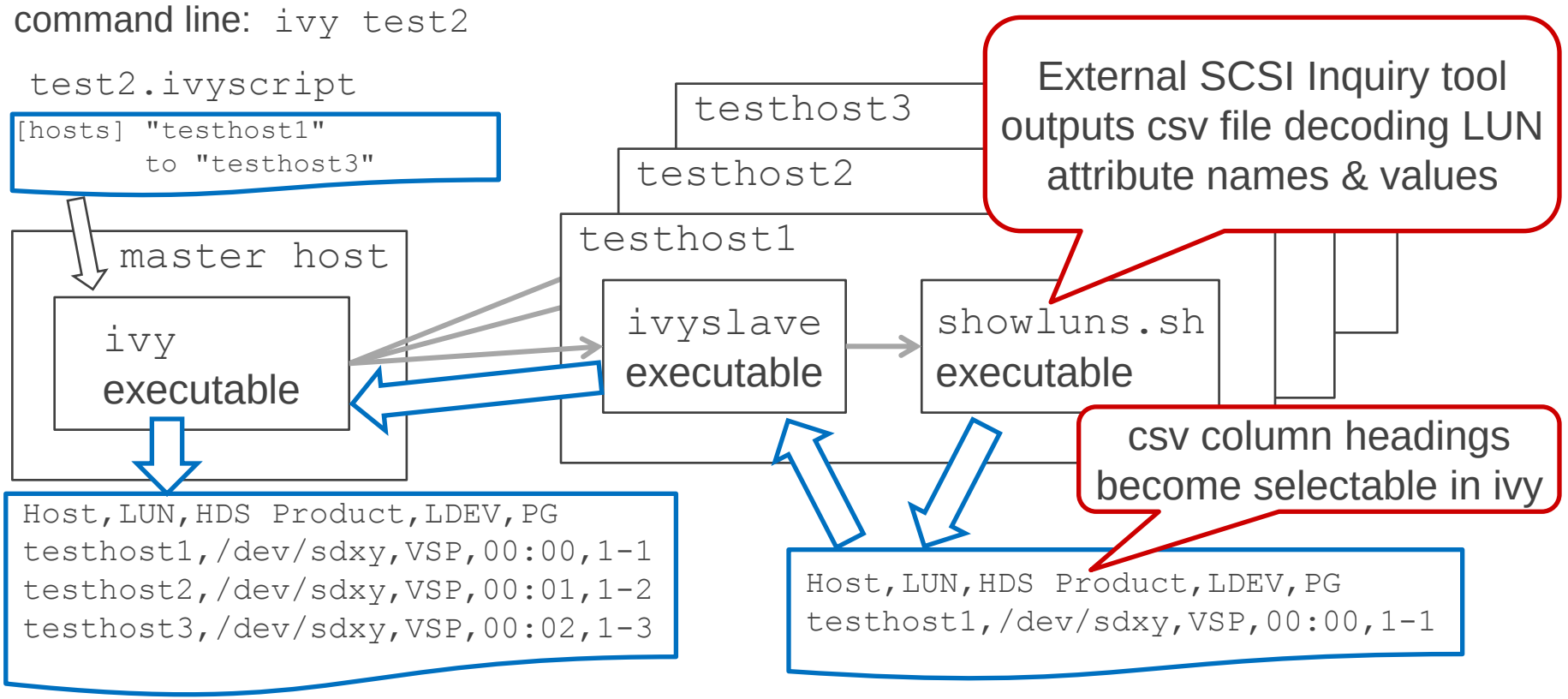
[Hosts] statement starts up ivy on test hosts

HITACHI
Inspire the Next

- On each specified host, the "ivyslave" executable is started using an ssh command.
  - Master host must be set up to ssh to test hosts without a password (that is, using certificate-based authentication.)
  - Have only tried running ivy as root.
    Don't know if a regular user is permitted to do raw LUN I/O.
- Each test host discovers all its storage LUNs, using a SCSI Inquiry-based LUN lister utility program.
  - ivy uses Ian's "showluns.sh" that decodes Hitachi proprietary attributes like subsystem type, serial number, LDEV, Port, PG, CLPR
- The combined list from all the test hosts is "all discovered LUNs"

Vendor-independent LUN attribute discovery

HITACHI
Inspire the Next



Sample attribute values from LUN lister tool

HITACHI
Inspire the Next

```
hostname = cb23
LUN_Name = /dev/sdbu
Hitachi_Product = HM700
HDS_Product = "HUS VM"
Serial_Number = 210030
Port = 1A
LDEV = 00:00
Nickname = ""
LDEV_type = Internal
RAID_level = RAID-1
Parity_Group = 01-01
Pool_ID = ""
CLPR = CLPR0
Max_LBA = 2097151
Size_MB = 1073.741824
Size_MiB = 1024.000000
Size_GB = 1.073742
Size_GiB = 1.000000
Size_TB = 0.001074
Size_TiB = 0.000977
Vendor = HITACHI
Product = OPEN-V
```

LUN attribute matching

HITACHI
Inspire the Next

- The LUN lister tool output csv file header line defines the LUN attribute names:
  - e.g. `"HDS Product,Serial Number, LDEV,..."`
- Internally within the C++ LUN object, ivy takes that column header, trims off any surrounding quote marks and white space, then converts all non-alphabetics/non-digits to underscores _, translates to lower case, and uses that internally within the object as the key.
- Then later on, if you ask if the LUN contains "HDS Product" or you ask for "hds_product", the LUN object's lookup routine does the same thing to the name you ask for before looking it up – either has the same effect.
- Similarly, when you ask if a value matches a LUN, the same normalization of the values is done before deciding if there is a match.

"all discovered LUNs" -> "available test LUNs"

HITACHI
Inspire the Next

- On the `[Hosts]` statement, the `[Select]` clause filters from all discovered LUNs on all the specified test hosts to create the pool of "available test LUNs" upon which you can "[CreateWorkload]".
- Only on the `[hosts]` statement, the `[Select]` clause must specify a non-null value for at least one of "`serial_number`" (which we always have for Hitachi) or "`vendor`" (if we are testing another vendor's equipment).
  - This is designed to prevent accidental annihilation of your boot drive.

[Select] syntax

HITACHI
Inspire the Next

- <expression for attribute name> `is` <expr. for att. value>
  - `"LDEV_type" is "DP-Vol"`
- <expression for attribute name> `is` { <list of attribute value expressions> }
  - `"port" is { "1A", "3A", "5A", "7A" }`
- The Select clause matches against a LUN if that LUN has the specified attribute and the value of that attribute matches.
- Select clauses are parsed by outer programming language, so there are no quotes around the entire `[Select]` expression.

[Select] LDEV, PG

HITACHI
Inspire the Next

- There are a couple of cases of "custom" attribute value matching for Hitachi subsystems. (Other vendors are encouraged to write their own.)
- There is a custom matcher for "`LDEV`" which understands things like "`00:1A-00:3F, 01:FF`"
- There is a custom matcher for "`PG`" which understands
  - "`1-*`" matches PG names starting with 1-
  - "`1-2:4`" matches 1-2, 1-3, 1-4
  - "`1-2:`" matches 1-2, 1-3, …
  - "`1-:2`" matches 1-1, 1-2

[hosts] – use of command devices is automatic

HITACHI
Inspire the Next

- After we have "available test LUNs", (which excludes command devices)
- The [hosts] statement looks through the command devices that were part of "all discovered LUNs", and for each unique subsystem serial number in available test LUNs, for the first command device found that goes to that subsystem, if the Hitachi proprietary command device connector "`ivy_cmddev`" (not part of the ivy open source project) is available, we fire it up remotely on the test host that has the command device, and retrieve the RMLIB API data on the configuration of the subsystem.
- For each available test LUN, if we have RMLIB API configuration data for the LDEV behind that LUN, the RMLIB API LDEV configuration attribute value pairs are merged into the LUN's attributes.
  - That means that if you have a command device, you can select on `drive_type` to create a workload.
- Later, when we run a test step, RMLIB API performance data is collected on the same time boundaries as the test intervals and that data goes in a csv file set for that test step.

Statements - [SetIogeneratorTemplate]

HITACHI
Inspire the Next

```
[SetIogeneratorTemplate] "random_steady"
      [parameters] "IOPS = 20, blocksize = 4KiB"
                + ", maxtags = 10, fractionRead = 50%";
```

- Sets the defaults for the specified I/O generator name.
- If you are going to use multiple [CreateWorkload]s with minor variations, you could use [SetIogeneratorTemplate] to set all the things that are in common, and then when you create each workload you only specify what's unique for that workload.
  - Handy if you are going to create a series of sequential workloads each starting at a different point in the LUN or having coverage of a different portion of the LUN. Then when reading the program, it's more clear what's going on if the [CreateWorkload] only sets what's different each time.
- The ivyscript language parser expects a single character string expression for [Parameters], as the string is passed as a whole to the corresponding underlying ivy engine function, which parses it there.

[iogenerator] some common [parameters]

HITACHI
Inspire the Next

- `VolumeCoverageFractionStart` default `0.0` same as `0%`
  `VolumeCoverageFractionEnd` default `1.0` same as `100%`
  - Establishes the "coverage zone" within the LUN. You can layer different workloads in different parts of the same LUN.
- `blocksize` default "`4 KiB`" same as "`4096`" – also supports "`MiB`" units.
- `maxTags` default `1`.
  - The maximum number of I/Os that this workload on this LUN is allowed to ***try*** to issue at one time.
  - OS call to start I/Os may block if underlying HBA/device driver is out of tags. Workloads share LUNs and share the underlying HBA/device driver.
- `IOPS` default `5`
  - `IOPS` = "`max`" - keep starting I/Os trying to keep queue depth at "maxTags".
- `fractionRead` default `1.0` same as `100%`.

`[iogenerator]` random – two types

HITACHI
Inspire the Next

- `random_steady`
  - I/Os are issued to random locations on a steady drumbeat in time.
- `random_independent`
  - I/Os occur at random times as well as to random locations
  - Random independent distributions are easier to model mathematically.
  - The lower the IOPS rate or the shorter the observation period, the more erratic `random_independent` IOPS will appear.
  - In general, `random_independent` I/O patterns will have a slightly higher service time compared to `random_steady` workloads, because scheduled I/O start times are independent and in general can collide (bursty), whereas `random_steady` workloads space out I/O scheduled start times evenly.

[iogenerator] "sequential"

HITACHI
Inspire the Next

- In ivy, a sequential workload must be all reads (`fractionRead=1.0` or `fractionRead=100%`) or all writes (`fractionRead=0%`).
- But, you can use a for loop to create a series of sequential threads starting at different points along the LUN, where each of the threads is either a read thread or a write thread
  - `SeqStartFractionOfCoverage = 0.23`
  - Range is from 0.0 to less than 1.0 - this is relative to the volume coverage zone defined from `VolumeCoverageFractionStart` to `VolumeCoverageFractionEnd`.
  - More commonly use the volume coverage parameters to have sequential threads wrap around in their own areas.

Sequential example – volume coverage

HITACHI
Inspire the Next

- Use a loop to create 10 sequential threads where each of the 10 threads operates within its own 1/10th of the LUN – its own “zone”, so that when it gets to the end of its own zone, it should wrap around to the beginning of that zone.

```
[hosts] "sun159" [select] "serial_number" is 83011441;

int    zones = 10;
int    zone;
double start, end;

for (zone = 0; zone < zones; zone = zone + 1)
{
        start = double(zone)/double(zones);
        end = double(zone+1)/double(zones);

        [CreateWorkload] "zone" + string(zone)
                    [iogenerator] "sequential"
                    [parameters] "VolumeCoverageFractionStart=" + string(start)
                                + ",VolumeCoverageFractionEnd=" + string(end)
                                + ",IOPS=max, blocksize=64KiB, fractionRead=100%, maxTags=1";
}

[CreateRollup] "workload"; // this is to give us data by zone across all LUNs

[Go] "stepname=separate_zones, measure_seconds = 60";
```

SeqStartFractionOfCoverage

HITACHI
Inspire the Next

- Use a loop to create 10 sequential threads where each of the threads covers the entire LUN, wrapping around from the end of the entire LUN to the beginning of the LUN, but where each thread starts at a different equally spaced point.

```
[hosts] "sun159" [select] "serial_number" is 83011441;

int    zones = 10;
int    zone;

double start;

for (zone = 0; zone < zones; zone = zone + 1)
{
        start = double(zone)/double(zones);

        [CreateWorkload] "zone" + string(zone)
                    [iogenerator] "sequential"
                    [parameters] "SeqStartFractionOfCoverage=" + string(start)
                              + ",IOPS=max, blocksize=64KiB, fractionRead=100%, maxTags=1";
}

[CreateRollup] "workload"; // this is to give us data by zone across all LUNs

[go] "stepname=whole_LUN_staggered_start, measure_seconds = 60";
```

Sequential example – read threads & write threads

HITACHI
Inspire the Next

- Use a loop to create a group of sequential workload threads each operating within its own "zone", and where some threads do writes and some do reads.

```
[hosts] "sun159" [select] "serial_number" is 83011441;

int zones = 12;  int zone; double start, end; double seq_percent_read = 75%;

for (zone = 0; zone < zones; zone = zone + 1)
{
        start = double(zone)/double(zones);
        end = double(zone+1)/double(zones);

        double rw; string p;

        if (  ( double(zone) / double(zones) ) < seq_percent_read ) { rw = 100%; p = "read_"; }
        else                                                        { rw = 0%;   p = "write_"; }

        [CreateWorkload] p + "zone" + string(zone)
                    [iogenerator] "sequential"
                    [parameters] "VolumeCoverageFractionStart=" + string(start)
                                + ",VolumeCoverageFractionEnd=" + string(end)
                                + ",IOPS=max, blocksize=64KiB, maxTags=1"
                                + ", fractionRead=" + string(rw);
}

[Go] "stepname=read_and_write_zones, measure_seconds = 60";
```

Sequential workloads and `maxTags`

HITACHI
Inspire the Next

- The default `maxTags` value is 1.
- ivy iogenerators (I/O sequencers) generate a sequence of I/Os in scheduled start time order.
- For `IOPS=max`, the scheduled start time for each I/O is zero.
- For all iogenerators, if you specify, for example, `maxTags=4`, this means "keep issuing I/Os when it's the scheduled time, except wait to start the next I/O if there are already 4 I/Os running".
  - For a sequential workload, `IOPS=max, maxTags=4` means "issue I/Os for 4 consecutive blocks at once, and then when one of these completes, keep issuing more to try to keep 4 running at all times."

Statements - [CreateWorkload]

HITACHI
Inspire the Next

- [CreateWorkload] "r_steady"
  [select] "LDEV" is "00:04"
  [iogenerator] "random_steady"
  [parameters] "fractionRead = 75%"

- Apply a [select] filter matching against "available test LUN" attribute values.

- On each selected LUN, create an identical workload thread with the specified workload name, running the specified [iogenerator] plug-in, and supplying the [iogenerator] with a [parameters] text string that the iogenerator will parse and apply.
  - Each type of iogenerator has its own set of valid parameter names.

[CreateWorkload]

HITACHI
Inspire the Next

- Attributes for selection are those of the underlying LUN plus several special built-int attributes
  - `workload`, set to the specified workload name, and
  - `host` which is an alias for `ivyscript_hostname`, the name used on the `[hosts]` statement which might be an alias or a dotted quad.
- The newly created workload threads will be in "waiting for command" state.

.ivyscript dedupe syntax

HITACHI
Inspire the Next

```
[CreateWorkload]  "owl"
     [select]      "port" is "1A"
     [iogenerator] "random_steady"
     [parameters]  "IOPS=max,fraction_read=\"50%\",
                    blocksize = \"4KiB\"
                    dedupe = 1.5 ";
```

- The dedupe parameter (default `dedupe = 1.0`) controls the average number of copies of a generated pattern that is written across the set of workload threads each mapped to a LUN on a test host with the same workload name (e.g. "`owl`")
  - `dedupe` must set to a value greater than or equal to 1.0
  - It is an error if some workload threads are set to a different `dedupe` value than other workload threads with the same name.
  - The `dedupe` parameter is ignored for `fraction_read = 100%`.

.ivyscript compressibility syntax

HITACHI
Inspire the Next

```
[CreateWorkload]  "owl"
     [select]      "port" is "1A"
     [iogenerator] "random_steady"
     [parameters]  "IOPS=max,fraction_read=\"50%\",
                    blocksize = \"4KiB\"
                    pattern = random ";
```

- The `pattern` parameter selects a pattern generator to fill the contents of a block before it is written to the LUN.
- The default is `pattern = random`.

.ivyscript pattern parameter

HITACHI
Inspire the Next

- `pattern = random`
  - Random binary noise. Not compressible. This is the ivy default
- `pattern = trailing_zeros, compressibility = 50%`
  - Each block has an incompressible section and a section with repeated zeros.
  - `compressibility` specifies the % of the block that is repeating zeros.
- `pattern = ascii`
  - Random ascii characters. Fixed degree of compressibility
- `pattern = gobbledegook`
  - Pseudo-English text generated by randomly selecting words from a dictionary.
  - Fixed degree of compressibility.

Using the first 32 Ki Words appearing in the 1913 public domain edition of Webster's dictionary.

pattern=random

HITACHI
Inspire the Next

```
offset 0x0000     0 "..G.n....:%.~wuH/...f.cM......6." (ffd44709 6ecaf4c6 fb3a25be 7e777548 2fa3c79b 66a2634d 9b04101f 1dab36ef)
offset 0x0020    32 "......N.\QL..I......?.....~.j.U4" (e816f69a 7f0a4e1b 5c514ce4 e549c1a4 90dd0cc5 3f13ba91 [illegible]
offset 0x0040    64 "~.p...!<.Z.=..#;P....7...|>*..Q." (7e89701e bf84213c 1d5ae83d e184233b 50e6aeff 0f370bec [illegible]
offset 0x0060    96 "..}..Z/.;}q.../.$..2...p.....JE." (b81d7d87 135a2fc7 3b7d71fc 10e62fce 24b60532 96d58370 [illegible]
offset 0x0080   128 ".4.I3.M....h].....I0...;..V$".l" (8434e949 33d64d02 dbed9d68 5df512ee 97a51949 3098881e [illegible]
offset 0x00a0   160 ".......h.#l[..G.n....<....be.:.e" (d7e40db2 9b0fc868 1a236c5b e29147d5 6e9ba8c3 ef3c10aa [illegible]
offset 0x00c0   192 ".../......w.....y~W%;Ao..[x..Y.A" (d2c9a32f 17c099c2 b092779a e9c2f8af 797e5725 3b416f15 [illegible]
offset 0x00e0   224 "...?\..{.H/mw.}F.....G.P.Y.v+.\." (f4e1913f 5cd1147b c1482f6d 77b27d46 8d97a484 ba478d50 [illegible]
offset 0x0100   256 "..+M..Q.+..n~.t...v..i...s.^=k.G" (92152b4d 95d251dc 2b10b76e 7ea87409 ede576ed 0869cfa9 [illegible]
offset 0x0120   288 "...b.8r`P{9....a.\]@@.*P%....$.=" (94aff162 9a387260 507b390c ee80e961 1e5c5d40 408c2a50 [illegible]
offset 0x0140   320 "..||>.........MV.J.&..S._7..f._." (ad1d7c7c 3eeb14ef f6aae09b 0f1c4d56 e84ae126 1ed65319 [illegible] 66c45ff1)
offset 0x0160   352 "..H..z.AB...,...$......=.CZ....|" (890948a3 d07aee41 42b91aed 2c9de494 2497ccc2 1da2e[illegible] 82435aa2 cdd08f7c)
offset 0x0180   384 ".\.\.......& ...O.&#....+s.?..0'" (125c925c e59fe31c f60fc926 20f0be09 4f932623 [illegible] 48687d6 2b73113f b4e43027)
offset 0x01a0   416 "....^.R..|......stq$....|......." (a9a6d3d3 5e8c527f 817cc7e8 d88c92be 73747124 c19ae1b3 7ce3d6fe f00d98a5)
offset 0x01c0   448 ".1t...I.A[S[.c.....\........l_.x" (8a3174a6 0ddd49e6 415b535b 9b638f05 d1f08d5c 8d97a9db 85181698 6c5fae78)
offset 0x01e0   480 ".Y.P.<...w.hb>.n&..d.IE}e.w..g.b" (f0590f50 cc3ceffe bb77e368 623eb26e 2612f664 db49457d 65ae77a2 9467f462)
offset 0x0200   512 "W..g...L&V#....+.q}..,3........." (57e49d67 b18cd74c 26562304 8094ba2b 06717dea d72c33cc ddf5b4c1 098f0b1e)
offset 0x0220   544 "..[.m....dN..lw...5o....f..C...." (d0c85ba9 6de71307 9c644ead fe6c7704 c8ac356f ba12d7d3 66b6c543 84e717e7)
offset 0x0240   576 "....*u../4.0.q..4.....w...x...m." (c1d71713 2a75c5b6 2f34fa4f 1271e980 34cf9b00 18d377c0 dc1a78be 1ba16dea)
offset 0x0260   608 "#..6.>...i......P..l.giv...|.5.." (23bc1f36 ec3efe10 a169f489 ae0fd68c 50eaad6c b3676976 b310fd7c 8a35e1e3)
offset 0x0280   640 ".....N..Bs......J^{<.k.cp..b..SU" (e6861bf1 a54ee8a1 4273c0c8 d3dffef7 4a5e7b3c fb6be163 70afb362 de955355)
offset 0x02a0   672 ".-B}.}..a<9.....k......y.8.<.7.a" (a32d427d 907d0ca5 613c39f7 a98b90d3 6b17c2f6 19088c79 9e380a3c a237c861)
offset 0x02c0   704 ".........}.&);d...'ki......X...." (f20ed0c9 ee0dacc0 1c7d1a26 293b64da 8b7f276b 69f01aee d68cab58 9fade706)
offset 0x02e0   736 "...p.......u6...5}..U....~._.b.." (d10770ca 0c86f111 b0fe7536 ca99bc35 7db9ee55 8fc7db8b 7ef8065f 8e62c6a5)
offset 0x0300   768 "...3.MsP.4N.u.Y..*..@...C#....g." (0fed0433 d34d7350 bc344e0f 75be5918 e12afdd5 40dd99f8 4323a5fc a6086717)
offset 0x0320   800 ".e..jbd..E.........q..>K..%.9.`." (8f65f999 6a6264e7 e845f916 97dbceea 9ea58371 cdab3e4b e4e725da 390a60b9)
offset 0x0340   832 "o..cM........c...s..5.E.ro..71Z." (6fb2ac63 4dc3b00c eb1713b5 e36397dd 0973cdfc 350645b6 726f0587 37315ac2)
offset 0x0360   864 "c..."...B.r.x,.e.o..No.K...L..[." (63939ac6 22c5b295 42e672b9 782cb865 986feef2 4e6ff34b bde6e04c 11c45bd0)
offset 0x0380   896 ".(h..8\.Vw..\.Tc.?_`.Q.........." (062868d4 fc385c19 5677e9c6 5ca75463 ab3f5f60 95517fe4 dd8ae4e5 f884d519)
offset 0x03a0   928 ".$........Q,-.......C,..&........" (b1249cfa 88a1cbbf 07fc512c 2d93a7e1 09ebe643 2cdcb026 960bc381 c6d19b04)
offset 0x03c0   960 "....7hF.9.l.O8.... .._.`....3j5." (ccdb8f07 3768461e 39ed6cf2 4f38e3ec f4e0209d a15fe560 a8fe1cdd 336a359c)
offset 0x03e0   992 "..6...P7.[.;.....)..K......oi..." (edd6367f e1e45037 ae5b923b d78ef3b2 e729b1fe 4b1fa79f 05e7f96f 69dcef9a)
```

Random binary data (incompressible)

pattern=trailing_zeros,compressibility=50%

HITACHI
Inspire the Next

```
offset 0x0000     0 "6.8.`Qe..n,x...v....'.-.;......." (360738cd 605165ed a06e2c78 bcdbd676 0b8cac94 27db2dd3 3be90396 1a120e9b)
offset 0x0020    32 ".S..3.........I.....Z.t..`~....H" (aa53d8bf 33c79f05 89ffd187 14bf49c5 0989acaf 5a9974f9 be607e8e 8790b
offset 0x0040    64 ".E..f..<...I`....B..~...<0.-...." (a645ac8d 6610993c 7ff51749 6083fbfc b342d9c6 7eade19f 3c30c02d 1498
offset 0x0060    96 "......-B. .......d.A.=.....yT..-3" (118cbca3 ae2d42cd 20111afe 05d5e216 64dd4106 3d9fb48e f5de7954 c3b1
offset 0x0080   128 "..Nr.vu....L..r......3.S^.^R...n" (afe14e72 be767584 aab4084c 8e87721d 1397022e ef33b553 5e145e52 cb10
offset 0x00a0   160 ".Dxb..q<......a..]c..3.7..n..R.." (8f447862 f99a713c 90b2ec06 0e1c611e 815d63b2 1133b337 05c36e9a ac52
offset 0x00c0   192 "._...3.D...y...Wq.Z..\.*,_....." (b1ba5ffa d51933b6 44faaf02 79dbb481 5771ec5a be865c13 2a2c5fd0 c5
offset 0x00e0   224 "..H.....K=":.Q.w.-d............" (9bef4812 89baabf6 4b3d223a cd519b77 9e2d649b 188188b1 928ba59
offset 0x0100   256 "...f...\Pw.U. wI1....c.0.?U..8.." (1bb49566 1c06955c 50778f55 a9207749 31b80bfe b563a030 903
offset 0x0120   288 ".([.._...{.K.*P"R>1G.....LsA.V.." (84285b90 bf5fe9ad 1e7bae4b 092a5022 523e3147 9b10c58b 4c7341 b856d9
offset 0x0140   320 "rKbp{.....|..n...~...p.e......D\" (724b6270 7bd5cf07 8b177cb5 e26e8694 e77e8fef ca70bd65 8880e1d2 80ea445c)
offset 0x0160   352 "o7.......dR9...m...Sl.S.Fq......" (6f371d12 e2bc9b1d 88645239 b4f2bb6d ebb90353 6ca853cd 4671f813 a7a3cbbb)
offset 0x0180   384 "....B.......&./.Q...h...6.B&Ly.." (04e2d180 4299dc8b 86acb4f4 26922fbe 51b8ceb2 68d61008 36044226 4c79eb17)
offset 0x01a0   416 "q.m..0Bq...uW..\.m..Q.2.......1." (71876ddc f7304271 15dede75 57dc065c 106dc40d 51ee3288 c60caf95 c5af31ac)
offset 0x01c0   448 ".?....P)^....?..9..@............" (be3fa2ba bbfd5029 5ef9dfdb c93fe41c 39e41640 c685d6bc ddf7aa83 b4cbd0e4)
offset 0x01e0   480 "...~rJ.q.y7#....i.Xme...{;.i...." (a58e897e 724a1671 e5793723 b3f71cf2 6999586d 65011ecd 7b3b8069 c2c9bde9)
offset 0x0200   512 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0220   544 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0240   576 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0260   608 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0280   640 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x02a0   672 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x02c0   704 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x02e0   736 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0300   768 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0320   800 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0340   832 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0360   864 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x0380   896 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x03a0   928 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x03c0   960 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
offset 0x03e0   992 "................................" (00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000)
```

Leading part of block is random binary data (incompressible)

compressibility = 50% means 50% trailing binary zeros

pattern=ascii

HITACHI
Inspire the Next

```
offset 0x0000     0 "aPJYuQcQ4yaW>Oa}LGc7;z[[9PFOj{\4" (61504a59 75516351 34796157 3e4f617d 4c476337 3b7a5b5b 3950464f 6a7b5c34)
offset 0x0020    32 "b|n=Vp%]iWLB^JIYRkXiH-$b%qBjz(b!" (627c6e3d 5670255d 69574c42 5e4a4959 526b5869 482d2462 2571426a 7a286221)
offset 0x0040    64 "A+>/bf.%nu^GV(t_ZFT7 AlZ76Y)Ran " (412b3e2f 62662e25 6e755e47 5628745f 5a465437 20416c5a 37365929 52616e20)
offset 0x0060    96 "ngxq??<Clv?'?M&w !po=ouj3fI/cnR=" (6e677871 3f3f3c43 6c763f27 3f4d2677 2021706f 3d6f756a 3366492f 636e523d)
offset 0x0080   128 "DI<:l?*([:Znm %5i*olN<'%!4"EQjVt" (44493c3a 6c3f2a28 5b3a5a6e 6d202535 692
offset 0x00a0   160 "UE]pNzawt<![XfUaB$`;~yQ/t8,O)*HE" (55455d70 4e7a6177 743c215b 58665561 422
offset 0x00c0   192 ",/~IB|VO2w ym`XZ`xRIn{S#mpe_RQ_)" (2c2f7e49 427c564f 32772079 6d60585a 607
offset 0x00e0   224 "U/>LPMdpl>FzJZY(3K=FYo}DD+eM:wU/" (552f3e4c 504d6470 6c3e467a 4a5a5928 334
offset 0x0100   256 "NQjVxX{YQ50]NMQ`SYG7t|3k<u@H1hsR" (4e516a56 78587b59 5135305d 4e4d5160 535
offset 0x0120   288 "AKphT_Y?pq+_xs#xo8)-ExmlwSrw:%]H" (414b7068 545f593f 70712b5f 78732378 65
offset 0x0140   320 "8D,f`,9/FE%HV7#$[. `1^MB[7_m!nZ(" (38442c66 602c392f 46452548 563
offset 0x0160   352 "O\7afoGRP/iX_;8?<{)E a:[L|?4_K{)" (4f5c3761 666f4752 503
offset 0x0180   384 "zKqHz8%+W/oca,%J*(C%`Z+]o9Is7qO$" (7a4b7148 7a38                  612c254a 2a2
offset 0x01a0   416 "CVz/Z%cqy3|@"R.<Tsc7=/!SEyD:(2lq" (4356                  79337c40 22522e3c 54736537 5d2f2155 4579445a 28526c71)
offset 0x01c0   448 " OevIIX31`1?wP}n]WOK['T}YbU@S8h"  (204f6576 49495833 3160313f 77507d6e 5d574f4b 5b27547d 59625540 5338683f)
offset 0x01e0   480 "7XVR9[sD"k^Xt%u/B>(wAwZv91[Qsk~S" (37585652 395b7344 226b5e58 7425752f 423e2877 41775a76 39315b51 736b7e53)
offset 0x0200   512 "q{m*=uU8WD6xXHvBzO9hKgBHTCib+]Sf" (717b6d2a 3d755538 57443678 58487642 7a4f3968 4b674248 54436962 2b5d5366)
offset 0x0220   544 "E`.z}^!6I-LX)X9kDW}%VT(1ixGC6`)K" (45602e7a 7d5e2136 492d4c58 2958396b 44577d25 56542831 69784743 3660294b)
offset 0x0240   576 "JkN^pNd@/}e+,3,HNA9W*-k\mzD~4i*V" (4a6b4e5e 704e6440 2f7d652b 2c332c48 4e413957 2a2d6b5c 6d7a447e 34692a56)
offset 0x0260   608 "FFb+sEQW&JE`kLL))9S1J\Y^-fV3":)}" (4646622b 73455157 264a4560 6b4c4c29 29395331 4a5c595e 2d665633 223a297d)
offset 0x0280   640 "{uIM9 >\9XUHp&UoOJH~.iT9H}{f'<io" (7b75494d 39203e5c 39585548 7026556f 4f4a487e 2e695439 487d7b66 273c696f)
offset 0x02a0   672 "1Ubl)&\RcmQ~,%2"XB!tOds#=+s%VGlm" (3155626c 29265c52 636d517e 2c253222 58422174 4f647323 3d2b7325 56476c6d)
offset 0x02c0   704 "v{=S\\hEJ7}&'qJqK{1)>BF,HW`/C(?P" (767b3d53 5c5c6845 4a377d26 27714a71 4b7b3129 3e42462c 4857602f 43283f50)
offset 0x02e0   736 "*(Bp.p">t=5XQBdU48@'\+._Z8$!1cv?" (2a284270 2e70223e 743d3558 51426455 34384027 5c2b2e5f 5a382421 3163763f)
offset 0x0300   768 "|n}SR;^->+n5f}taf5U=$f=po|FBG/L%" (7c6e7d53 523b5e2d 3e2b6e35 667d7461 6635553d 24663d70 6f7c4642 472f4c25)
offset 0x0320   800 ""<8*"7/yNR |DW|Sgamv\k]3i1bVO-Zl" (223c382a 22372f79 4e52207c 44577c53 67616d76 5c6b5d33 69316256 4f2d5a6c)
offset 0x0340   832 "dBogZ&1 7L6f6s.F"VNCO8..mD1*?4OV" (64426f67 5a263120 374c3666 36732e46 22564e43 4f382e2e 6d44312a 3f344f56)
offset 0x0360   864 "w%?ODQ&imX}W:e}c(9\]tN+`p+66Wi!l" (77253f4f 44512669 6d587d57 3a657d63 28395c5d 744e2b60 702b3636 5769216c)
offset 0x0380   896 "Su, cHdCRXx>.O/QQK$mFgy6~b#.}?B2" (53752c20 63486443 5258783e 2e4f2f51 514b246d 46677936 7e62232e 7d3f4232)
offset 0x03a0   928 "QnYH&:.l&xvoho58(I`h*U~`;|rt`+W}" (516e5948 263a2e6c 2678766f 686f3538 28496068 2a557e60 3b7c7274 602b577d)
offset 0x03c0   960 "6J**<hVYXQ0(]wa<bV7JwP[-U{yd F<A" (364a2a2a 3c685659 58513028 5d77613c 6256374a 77505b2d 557b7964 20463c41)
offset 0x03e0   992 "I=FaNDTK7FP-P{qJMjYfkbnnVH-_J3qn" (493d4661 4e44544b 3746502d 507b714a 4d6a5966 6b626e6e 56482d5f 4a33716e)
```

Randomly selected printable ASCII characters (fixed degree of compressibility)

pattern=gobbledegook

HITACHI
Inspire the Next

```
offset 0x0000     0 "agens fever APPLICATORY indignan" (6167656e 73206665 76657220 4150504c 49434154 4f525920 696e6469 676e616e)
offset 0x0020    32 "t bacoro anamnestic ADVERTENCE W" (74206261 636f726f 20616e61 6d6e6573 74696320 41445645 5254454e 43452057)
offset 0x0040    64 "EBSTER weighed DELEGATE ail radi" (45425354 45522077 65696768 65642044 454c4547 41544520 61696c20 72616469)
offset 0x0060    96 "ant ANGLOMANIAC publicity fixus " (616e7420 414e474c 4f4d414e 49414320 [illegible]
offset 0x0080   128 "alouer reality arma satiety aumo" (616c6f75 65722072 65616c69 74792[illegible]
offset 0x00a0   160 "snier isotropic Seeley actinism " (736e6965 72206973 6f74726f 70696[illegible]
offset 0x00c0   192 "hatched addicere extrinsically p" (68617463 68656420 61646469 63657[illegible]
offset 0x00e0   224 "arlance fell aquiline passed ant" (61726c61 6e636520 66656c6c 20617[illegible]
offset 0x0100   256 "iquarian rot AGGRANDIZER AFTERBI" (69717561 7269616e 20726f74 204[illegible]
offset 0x0120   288 "RTH ANALYTICS yearly occultation" (52544820 414e414c 595449[illegible]
offset 0x0140   320 " nathra Notwithstanding discrimi" (206e6174 68726120 [illegible]
offset 0x0160   352 "nate CHAMBER mongst tacitly prom" (6e617465 204[illegible] 424552 206d6f6e 67737420 74616369 746c7920 70726f6d)
offset 0x0180   384 "inences ambulacral designs avail" (696e656[illegible] 7320 616d6275 6c616372 616c2064 65736967 6e732061 7661696c)
offset 0x01a0   416 " aberrating Argillaceous making " (20[illegible] 72726174 696e6720 41726769 6c6c6163 656f7573 206d616b 696e6720)
offset 0x01c0   448 "prose apostele tro ALLODIARY bo" (70726f73 65206170 6f737465 6c652074 726f2041 4c4c4f44 49415259 20626f77)
offset 0x01e0   480 "lines malonyl exists Per bases a" (6c696e65 73206d61 6c6f6e79 6c206578 69737473 20506572 20626173 65732061)
offset 0x0200   512 "cuminate ciere legumes necessary" (63756d69 6e617465 20636965 7265206c 6567756d 6573206e 65636573 73617279)
offset 0x0220   544 " onward Bombax APPLY exploit sym" (206f6e77 61726420 426f6d62 61782041 50504c59 20657870 6c6f6974 2073796d)
offset 0x0240   576 "pathy ANGELICALNESS ALGAROBA gai" (70617468 7920414e 47454c49 43414c4e 45535320 414c4741 524f4241 20676169)
offset 0x0260   608 "ning Alpes ACTIVITY buoyancy wit" (6e696e67 20416c70 65732041 43544956 49545920 62756f79 616e6379 20776974)
offset 0x0280   640 "her filaments Blackfeet opponent" (68657220 66696c61 6d656e74 7320426c 61636b66 65657420 6f70706f 6e656e74)
offset 0x02a0   672 "s footing cannon anai APTLY arge" (7320666f 6f74696e 67206361 6e6e6f6e 20616e61 69204150 544c5920 61726765)
offset 0x02c0   704 "ntic ALLUSION appropinquatus apo" (6e746963 20414c4c 5553494f 4e206170 70726f70 696e7175 61747573 2061706f)
offset 0x02e0   736 "stel ARCHIEPISCOPATE AGGLOMERATE" (7374656c 20415243 48494550 4953434f 50415445 20414747 4c4f4d45 52415445)
offset 0x0300   768 "D nightingale aphol ACCUSTOMABLE" (44206e69 67687469 6e67616c 65206170 686f6c20 41434355 53544f4d 41424c45)
offset 0x0320   800 " law centripetal AMIABLE rin AMN" (206c6177 2063656e 74726970 6574616c 20414d49 41424c45 2072696e 20414d4e)
offset 0x0340   832 "ESIA yardarm deserving jure argu" (45534941 20796172 6461726d 20646573 65727669 6e67206a 75726520 61726775)
offset 0x0360   864 "mentatio Num flushed Abas APRICA" (6d656e74 6174696f 204e756d 20666c75 73686564 20416261 73204150 52494341)
offset 0x0380   896 "TION AFFLUENTLY old affected Hud" (54494f4e 20414646 4c55454e 544c5920 6f6c6420 61666665 63746564 20487564)
offset 0x03a0   928 "ibras ACROTISM ARAEOSYSTYLE Gall" (69627261 73204143 524f5449 534d2041 5241454f 53595354 594c4520 47616c6c)
offset 0x03c0   960 "icism PEAR ACHIEVEMENT reclining" (69636973 6d205045 41522041 43484945 56454d45 4e542072 65636c69 6e696e67)
offset 0x03e0   992 " mechanism Amalgamating cooperat" (206d6563 68616e69 736d2041 6d616c67 616d6174 696e6720 636f6f70 65726174)
```

Randomly selected words from Webster's 1913 dictionary.
(fixed degree of compressibility)

Statements - [DeleteWorkload]

HITACHI
Inspire the Next

- `[DeleteWorkload] "r_steady" [select] "LDEV" is "00:04";`
- `[DeleteWorkload] "r_steady" ;`
  - Deletes all instances of the `r_steady` workload on all test hosts / all LUNS.
- `[DeleteWorkload] ;`
  - Deletes all workloads.

Statements – [CreateRollup]

HITACHI
Inspire the Next

- `[CreateRollup] "Serial_Number+Port"`
- `[CreateRollup] "host"`
  `[nocsv] [quantity] 8 [MaxDroop] "25%";`
- A rollup is a partition of all workload threads.
- Every workload thread belongs to exactly one instance of each rollup.
- There is always an "`all`" workload which only has one instance "`all`".
- `[nocsv],[quantity],[MaxDroop]` are optional, but if they appear they must be in that order
- To get individual data for each workload thread, say "`workloadID` " which is comprised of "`host+LUN_name+workload`".

You make rollups for four reasons

HITACHI
Inspire the Next

1. To get an output csv file with a csv folder by rollup type (e.g. Port+CLPR) and csv files by rollup instance (e.g. Port+CLPR = 1A+CLPR0)
   - This is how you get custom "sliced & diced" data.
2. To perform dynamic feedback control (`dfc=PID`) at the granularity of the rollup instance.
3. To identify a valid measurement period at the granularity of the rollup instance using `measure=on`.
4. To validate the test configuration as operating correctly
   - E.g. test that the number of ports reporting was what you expected
   - E.g. validate that no one port had an IOPS too far below the highest IOPS seen on any port.

Rollups are key to how the ivy engine works

HITACHI
Inspire the Next

- **`[CreateRollup] "Serial_Number+Port"`** (no spaces are permitted around the + sign)
  - Both `Serial_Number` and `Port` must be valid LUN-lister column header attributes, or built-in layers on top of those attributes
- Then for all existing WorkloadIDs, we build a data structure that looks like this

```
- "Serial_Number+Port"
    - "410123+1A"
        - "sun159+/dev/sdd+workload_name", "cb28+/dev/sdd+workload_name",
    - "410321+1A"
        - "sun159+/dev/sdf+workload_name", "cb28+/dev/sdf+workload_name"
- "host"
    - "sun159"
        - "sun159+/dev/sdd+workload_name", "sun159+/dev/sdf+workload_name"
    - "cb28"
        - "cb28+/dev/sdd+workload_name", "cb28+/dev/sdf+workload_name"
```

The rollup type

The rollup instance. It's the rollup instance that has the rolled up data.

List of WorkloadIDs that "landed" on this rollup instance

- Every WorkloadID appears exactly once in each rollup type

[CreateRollup] combines LUN attribute names

HITACHI
Inspire the Next

- If you would like to see a rollup instance for each unique LDEV across two or more subsystems, make "serial_number+LDEV".
- [nocsv] – prevents csv files from being created.
- The [quantity] <int expression> clause can enforce that the right number of distinct rollup instances are reporting in this rollup.
  - If not, even if the DFC reports "success" and designates a subinterval subsequence representing a successful measurement, no measurement rollup csv data will be produced – instead error msg.
  - Make a rollup by "port" or "host+scsi_bus_number__hba_" and use the [quantity] rollup to validate you have the number of paths you think you have.
- [MaxDroop] <double expression>
  - "25%" means invalidate test if any one rollup instance has an average IOPS more then 25% below the highest average IOPS over all rollup instances.

[DeleteRollup]

HITACHI
Inspire the Next

- `[DeleteRollup] "serial_number+Port";`
- `[DeleteRollup] ;`
  - Deletes all rollups except the "all" rollup.

[EditRollup]

HITACHI
Inspire the Next

- `[EditRollup]`
  `"serial_number+Port = { 410123+1A, 410123+2A }"`
  `[parameters] "fractionRead = 100%";`

- The `[EditRollup]` statement operates in between test steps while the workload threads are in "waiting for command" state.

- It gives you access to the same mechanism used by a Dynamic Feedback Controller to send out real time parameter updates to running workload threads.

- You specify a set of rollup instances to send to, such as "`all=all`", or "`Port={1A,3A,5A,7A}`" and you specify the text `[parameters]` string to send to the remote iogenerator to parse and apply.

[EditRollup]

HITACHI
Inspire the Next

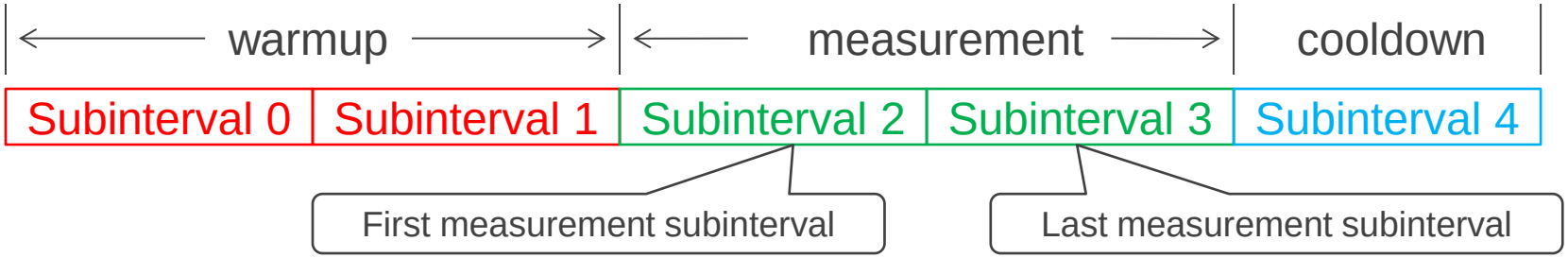
- `[EditRollup]` is typically used at the top of a do-loop, to change whatever parameters vary by loop pass.
- Use `[EditRollup] "all=all"` to send to all workload threads.
- There is a special parameter name that is only recognized by [EditRollup] - `total_IOPS` - where the numeric value you specify is divided by the number of workload threads comprising a workload instance, before being sent out as `IOPS=`.
  - `[EditRollup] "all=all" [parameters] "total_IOPS = 1000000";`

Statement – [Go]

HITACHI
Inspire the Next

- `[Go] "stepname=random4K, subinterval_seconds=5, …"`
- The `[Go]` statement starts the workload threads running a "test step", which is a sequence of "subintervals" each of a duration specified in the `subinterval_seconds` parameter, defaulting to 5 seconds.
  - If you have a case for using ivy to measure a restricted set of things much more frequently, we can talk about putting in support.
  - Most of the time 5 seconds is plenty short and if you are going to be doing any tests that will run for hours you may want to consider a longer subinterval just to mercifully cut down on the size of the csv files by subinterval.
  - Sometimes when you say you want an answer to +/- 1% and the behaviour is a bit noisy, it can take time to see enough to say you are sufficiently confident statistically. (Did you say you wanted "valid" data?)

Test step = warmup, measure, cooldown



- There must be at least one warmup, one measurement, and one cooldown subinterval.
- Parameter defaults
  - `warmup_seconds = 5` - this number is divided by subinterval_seconds, and rounded up to get the (minimum) number of warmup subintervals.
  - `measure_seconds = 60` - also rounded up to the minimum number of measurement subintervals.
  - `cooldown_by_wp = on` - If a command device is available for the subsystem under test, the cooldown period is extended until write pending is empty.

For each test step you get:

HITACHI
Inspire the Next

- A subfolder of the overall test output folder that contains the csv files with one line for each subinterval in that test step.
  - Nested subfolders for each workload data rollup
    - Containing a csv file for each rollup instance, with one line per subinterval.
  - A nested subfolder with raw RAID_subsystem RMLIB API data.
    - Collected time-synchronized "just before" the end of each subinterval.
- A single line in the overall test results "summary.csv" files.
  - In ivy terminology, this is called a "measurement" line, which represents the rollup from the first to last measurement subintervals.
    - Unless "`measure=on`" with specified accuracy timed out – then you get an error message line

cooldown_by_wp

HITACHI
Inspire the Next

- Default: `cooldown_by_wp = on`
- Set `cooldown_by_wp = off`
  - When it is valid to carry forward dirty data in cache (Write Pending) from one test step to the next.
  - This can speed up the next test step tremendously if
    - the next step doesn't stabilize until WP is full,
    - AND if both steps place the SAME things into WP.

The default `[Go]` statement

HITACHI
Inspire the Next

- [go];
  - Default `warmup_seconds = 5`
  - Default `measure_seconds = 60`
  - Default `subinterval_seconds = 5`
  - Default `cooldown_by_wp = "on"`
    - Runs at least one cooldown subinterval
    - If you have a command device and the proprietary command device connector software, continuing more cooldown subintervals until WP is empty.
  - Useful when you are developing an ivyscript workflow and you just want to see quick sample csv files.

stepname

HITACHI
Inspire the Next

- On the [Go] statement to start a test step, you can optionally specify `"stepname="`, which defaults to "step" followed by a four digit step number starting with 0000, so the default name for the first step is `step0000`.

- Giving a test step a meaningful name is useful when looking at overall measurement summary csv files, where you get one csv line for each test step.

- Those labels are handy when making Excel charts, as you can use the stepname column as the series name on a chart.

[Go] "focus metric"

HITACHI
Inspire the Next

- If you want to run a fixed workload for a fixed number of subintervals, all you need is `warmup_seconds` and `measure_seconds`.

- Otherwise, we need to specify the "focus metric".

  1. The focus metric is what we are making a valid measurement of using "`measure=on`", the "seen enough and stop" feature.
     - Measure the focus metric to a required plus/minus accuracy with a specified % confidence level.
  2. When dynamically adjusting `total_IOPS` using the PID loop dynamic feedback controller (`dfc=pid`), the focus metric is the "feedback" in dynamic feedback control.

Granularity of the "focus metric"

HITACHI
Inspire the Next

- When `measure=on` and/or `dfc = pid` are used, measurement or PID loop DFC is performed at the granularity of each instance of the `focus_rollup`.
- For the default, `focus_rollup = all`, the measurement or DFC is at the overall level.
- When a `focus_rollup` is used that has multiple rollup instances,
  - With `measure=on`, a successful measurement identifies a subsequence of subintervals where for every rollup instance within the `focus_rollup`, the measurement is valid for that rollup instance.
  - When `dfc=pid` is used, dynamic feedback control is performed independently for each rollup instance in the `focus_rollup`.

`source` of the `focus_metric`

HITACHI
Inspire the Next

- `source = workload`
  - Specifies that we are selecting a focus metric from data collected by ivy workload threads on test hosts.
  - We always have rollup data from test host workload threads (*more next page*)
- `source = RAID_subsystem`
  - Requires the proprietary command device connector that is not part of the ivy open source project.
  - Specifies that we are selecting the focus metric from real time performance data collected from a command device.
  - There's a small list of subsystem metrics specified in an ivy source code table that are filtered and rolled up from the raw bulk RMLIB data by rollup instance, and from which you select the focus metric. (*more even later after we explain* `source=workload`)

Selecting a "source=workload" metric

HITACHI
Inspire the Next

- category =
  - overall, read, write, random, sequential, random_read, random_write, sequential_read, sequential_write
- accumulator_type =
  - bytes_transferred, service_time, response_time
- accessor =
  - avg, count, min, max, sum, variance, standardDeviation
- It will be easier to explain first accessor then category, then accumulator_type

Accumulators and "accessor"

HITACHI
Inspire the Next

- An accumulator is an object that you push numbers into in order to be able to compute summary values.
- Every time that an I/O completes, ivy posts the service time into one accumulator, the bytes transferred into another accumulator, and if we are not running IOPS=max, it posts the response time into another accumulator.
- The selectable values for "`accessor`" are the names of the methods that you can use to retrieve something from an accumulator
  - `avg, count, min, max, sum, variance, standardDeviation`
  - `avg` gives you the average of the numbers that were pushed in the accumulator
  - `count` gives you how many numbers were pushed in.
  - Et cetera.

Attributes of individual I/Os:

HITACHI
Inspire the Next

- read vs. write
- blocksize
- LBA
  - Logical Block Address = sector number from 0 within LUN
- service_time (in seconds)
  - The duration from when ivy launched an I/O until ivy received the notification that the I/O was complete.
- response_time* (in seconds) (analogue to application-level response time)
  - The duration from the scheduled start time of an I/O until the time the I/O is complete.
  - An I/O may not be started at the scheduled time if there are no idle asynchronous I/O "slots" (~tags) available.
  - ***only I/Os with a non-zero scheduled start time will have a response_time attribute.**
  - When running iops=max, all I/Os have a scheduled start time of zero, meaning you don't get response_time

Ivy uses the Linux nanosecond resolution clock for all timing

How ivy posts results of each I/O

HITACHI
Inspire the Next

- Based on the attributes of each I/O, an accumulator category is selected.
  - Then the I/O is posted into the selected category "bucket" (into two or three accumulators in that bucket – more in a moment.)
- Currently, the breakdown for the array of categories for which there are accumulators are
  - read vs. write
  - random vs. sequential (The I/O sequencer tells you if it's a random or sequential sequencer.)
  - For each of those 4 there is a further breakdown as a histogram by service time and by response time
    - You see the histograms in the csv files.
    - Ivy doesn't currently expose the histogram in the PID loop, but if there is interest it can be added.

Other category breakdowns could be defined

HITACHI
Inspire the Next

- The rollup mechanism operates on a view of the categories as an array, and is blind to the significance of each position in the array.
  - It is easy to define a different mapping from the attributes of an individual I/O to the category bucket the I/O will be recorded in.

- Future:
  - We could just as easily define a histogram of a 100 buckets by LBA range - we could break out the data by each 1% of the LBA range across the volume.
    - If we had an I/O sequencer that was playing back a customer I/O trace, we could show if workload characteristics were different in different areas of the LUN.
    - If we simply run sequential transfers across the LUN, we could see the sustained data rate "staircase" showing the zones in underlying HDDs.

During rollups, the categories are preserved

HITACHI
Inspire the Next

- For the `all=all` instance, you still have all the category breakdowns.
- Then in addition to the category bucket array, there are virtual categories, implemented as functions, which rollup underlying category buckets.
  - `overall` – sum over all categories in the bucket array
  - `read, write`
  - `random, sequential`
  - `random_read, random_write, sequential_read, sequential_write`
- You can see these virtual category rollups in column groups in ivy csv files.

source=workload available category values

HITACHI
Inspire the Next

- overall
  read, write
  random, sequential
  random_read, random_write, sequential_read, sequential_write

- These are actually the virtual categories, representing the rollup over the underlying service time / response time bucket arrays (histograms).
  - If there is a need, we could provide access to the more fine-grained underlying category bucket array, or we could define other virtual categories as aggregations of the buckets.

source=workload - selecting accumulator

HITACHI
Inspire the Next

- Category buckets have 3 accumulators
- accumulator_type = bytes_transferred
  - For every I/O, the blocksize is posted to bytes_transferred.
  - Use sum attribute and divide by elapsed seconds to get bytes per second. Use count instead and get IOPS.
- accumulator_type = service_time
  - For every I/O the duration from when ivy started it to when it completed.
  - service_time and response_time values for I/Os are posted in units of seconds, with nanosecond resolution.
  - Use "avg" and multiply by 1000 to get average service time in ms.
- accumulator_type = response_time (~ application response time)
  - Only posted for those I/Os that have a non-zero "scheduled time".
  - Duration from scheduled time to I/O completion time.
  - The I/O sequencer computes the scheduled time, and when that time is reached, the I/O is started if there is an idle Asynchronous I/O "slot" (~tag) available. If not, it waits.
  - For IOPS=max, I/Os have a scheduled time of 0 (zero), so then you don't get any response_time events.

Summary: source=workload

HITACHI
Inspire the Next

- category =
  - overall, read, write, random, sequential, random_read, random_write, sequential_read, sequential_write
- accumulator_type =
  - bytes_transferred, service_time, response_time
- accessor =
  - avg, count, min, max, sum, variance, standardDeviation

source = RAID_subsystem

HITACHI
Inspire the Next

- Subsystem performance data is collected from a command device, and for each subsystem with a command device, there is a subfolder within the test step folder, where each csv file has one line per subinterval within that test step.
  - You cannot select the focus metric from this raw, bulk subsystem performance data.

- A small subset of metrics are extracted from the bulk subsystem data, and filtered and summarized by rollup instance
  1. To serve as candidates for selection as the focus metric
  2. To be printed as columns in rollup instance csv files side-by-side with the columns of host-workload data.

- This is controlled by a table in ivy source code, which has two levels that you pick from
  - `subsystem_element`, and within that, `element_metric`.

- For each metric in the table, you can optionally set a flag to have the value inserted a column side by side with the normal workload data for each rollup instance.

Subsystem metrics by rollup instance 2015-11-19

HITACHI
Inspire the Next

- MP_core
  - busy_percent, io_buffers
- CLPR
  - WP_percent
- PG
  - busy_percent,
    random_read_busy_percent, random_write_busy_percent, seq_read_busy_percent,
    seq_write_busy_percent
- LDEV
  - read_service_time_ms, write_service_time_ms,
    random_blocksize_KiB, sequential_blocksize_KiB,
    random_read_IOPS, random_read_decimal_MB_per_second , random_read_blocksize_KiB,
    random_read_hit_percent,
    random_write_IOPS, random_write_decimal_MB_per_second, random_write_blocksize_KiB,
    sequential_read_IOPS, sequential_read_decimal_MB_per_second, sequential_read_blocksize_KiB,
    sequential_write_IOPS, sequential_write_decimal_MB_per_second,
    sequential_write_blocksize_KiB,

Subsystem data filtered by rollup instance

HITACHI
Inspire the Next

- The way this works is via a "config filter" that is prepared in advance before a subinterval sequence starts.
- For each thing you get data for, such as PG, or LDEV, or MPU, etc., the config filter has the set of instances of PG or LDEV or MPU names that were either
  - directly observed as a SCSI Inquiry attribute of the LUNs underlying the workloads in the rollup instance, or
  - observed as an attribute of an underlying LDEV obtained via the RMLIB API, or
  - which were inferred from static tables of relationships for the particular subsystem model.

Subsystem data by rollup instance – csv columns

HITACHI
Inspire the Next

We know how many drives underlie the each workload rollup

Shows you if the OS / device driver are breaking up your large block application-level I/O into smaller pieces

Matching subsystem vs. application data validates that both host-workload rollups and subsystem data rollups are working correctly

Shows you if there is delay between when the application issues the I/O and when the device driver issues the I/O.

Shows you the amount of that delay in ms.

subsystem avg LDEV sequential_blocksize_KiB	subsystem avg LDEV read_service_time_ms	subsystem avg LDEV write_service_time_ms	host IOPS per drive	host decimal MB/s per drive	Subsystem IOPS as % of application IOPS	Subsystem MB/s as % of application MB/s	Subsystem service time as % of application service time	Path latency = application minus subsystem service time (ms)	Overall IOPS	Overall Decimal MB/s	Overall Average Blocksize (KiB)	Overall Little's Law Avg Q
0	6.82666	0	1.90	-	98.22%	98.22%	98.69%	0.091	30	0.12288	4	0.207528
4	0	60.5501	98.30	0.40	99.99%	99.99%	99.08%	0.563	1572.9	6.4426	4	96.1246
4	50.0142	49.0471	120.70	0.50	100.43%	100.43%	99.27%	0.364	1931.98	7.91339	4	95.9223
4	41.4719	32.018	162.00	0.70	99.99%	99.99%	99.24%	0.283	2592.74	10.6199	4	95.9846
4	39.3228	3.90259	194.90	0.80	99.68%	99.68%	99.02%	0.303	3118.22	12.7722	4	96.0234
0	20.9364	0	283.10	1.20	100.51%	100.51%	98.79%	0.257	4529.76	18.5539	4	95.9988

RMLIB API candidates flagged to display

HITACHI
Inspire the Next

maxTags	IOPS input parameter setting	fractionRead	test host cores	test host avg core CPU % busy	subsystem MP_core count	subsystem avg MP_core busy_percent	subsystem CLPR count	subsystem avg CLPR WP_percent	subsystem PG count	subsystem avg PG busy_percent	subsystem LDEV count	subsystem avg LDEV random_read_IOPS	subsystem avg LDEV random_write_IOPS	subsystem avg LDEV random_blocksize_KiB
16	5	1	16	0.10%	12	2.52%	1	0%	4	1.12%	6	4.91	-	4.00
16	max	0	16	0.00%	12	5.39%	1	68.78%	4	99.83%	6	-	261.42	4.00
16	max	0.25	16	0.00%	12	5.63%	1	68.84%	4	99.82%	6	80.11	242.82	4.00
16	max	0.5	16	0.00%	12	6.06%	1	68.27%	4	99.85%	6	215.46	216.43	4.00
16	max	0.75	16	0.10%	12	6.23%	1	62.59%	4	99.80%	6	388.87	128.99	4.00
16	max	1	16	0.10%	12	4.34%	1	0.20%	4	95.16%	6	758.77	-	4.00

- The “subsystem” columns are automatically generated according to the focus metric RMLIB API candidate table.
- As raw data comes in for each MP_core, CLPR, PG, LDEV, etc., ivy filters the data to aggregate for each rollup only the data for the MP_cores, etc. that map to LDEVs/LUNs underlying workloads in that rollup.
- Make a rollup by MPU, and each MPU rollup instance will show data for 4 MP_cores.

Examples of data for each rollup – drive / PG type

HITACHI
Inspire the Next

Rollup Type	Rollup Instance	drive type	drive quantity	RAID level	PG layout	iogenerator type	blocksize	maxTags	IOPS input parameter setting	fraction Read
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	5	1
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0.25
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0.5
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	0.75
all	all	DKR2E-H4R0SS+DKR5D-J600SS	8+8=16	RAID-5	3+1	random_steady	4 KiB	16	max	1

- Information comes from RMLIB API configuration data, filtered / aggregated for each rollup instance.
- (There are also dedicated csv folders that contain detailed RMLIB API subsystem configuration and performance data csv files.)

Now that we know how to specify the focus metric

HITACHI
Inspire the Next

- We will look at
  - The [`go`] statement `measure=on` option and its subparameters
    - Specifying `measure=on` on a Go statement means "watch the focus metric and when you have seen enough to make a measurement of the specified accuracy, stop. Timeout if it takes too long."
  - The [`go`] statement `dfc=pid` option and its subparameters
    - If you don't specify a `dfc`, the workload settings remain constant through the test.
    - If you specify a `dfc` (dynamic feedback controller), it gets called at the end of every subinterval once all the rollups are done.
    - The DFC looks at what has happened so far, looking at all workload data and all subsystem data, and then may use the ivy engine real time edit rollup mechanism to send out parameter updates to rollup instances (to the workload threads belonging to the rollup instance).

measure = on

HITACHI
Inspire the Next

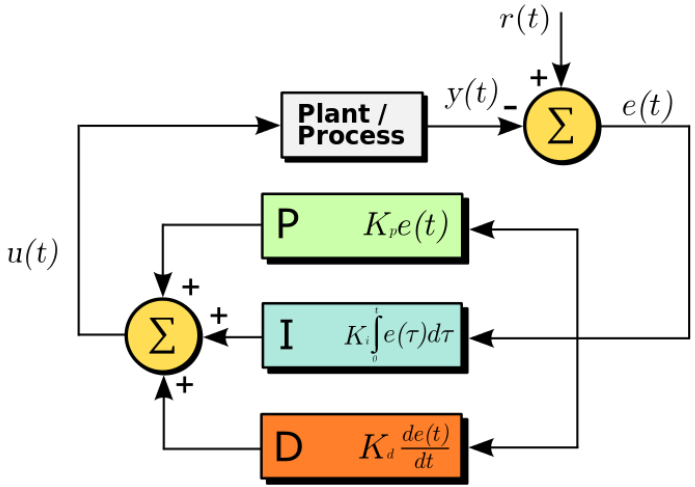
- `accuracy_plus_minus = 2%`
  - Any numeric value with an optional trailing % sign maybe specified.
- `confidence = 95%`
  - How confident you need to be that your measurement falls within the specified plus or minus range around the long term average that you would get measuring forever.
  - Default is 95%
  - Ivy has a menu of 11 specific pre-loaded confidence values that you pick from.
    - 50%, 60%, 70%, 80%, 90%, 95%, 98%, 99%, 99.5%, 99.8%, and 99.9%
    - http://en.wikipedia.org/wiki/Student%27s_t-distribution

`measure` Write Pending-based stability criteria

HITACHI
Inspire the Next

- `max_wp = 2%` - default `100%`
  - A subinterval sequence will be rejected if WP is above the limit at any point in the sequence.
  - Set this to "1%' or so for read tests to ensure WP is empty during the test.
- `min_wp = 67%` - default `0%`
  - A subinterval sequence will be rejected if WP is below the limit at any point in the sequence.
  - Use this for write tests to ensure WP is full during the test.
- `max_wp_change = 3%` - default `3%`
  - A subinterval sequence will be rejected if WP varies up and down by more than the specified (absolute) amount at any point in the sequence. `max_wp_range="3%"` matches from 0% to 3% Write Pending, as well as from 67% to 70% Write Pending. (not a percent OF the WP value)
  - Use this in general all the time so you reject periods with major movement in Write Pending.

`dfc=pid` dynamically adjusts total IOPS



- General purpose DFC – see http://en.wikipedia.org/wiki/PID_controller
- The feedback is the value of the focus metric
  1. `source=workload`
     - E.g. host-view service time, response time
  2. `source=RAID_subsystem`
     - e.g. `subsystem_element="PG", subsystem_metric="busy_percent".`
- User specifies “p”, “i”, and “d” constants.

PID loop basics

HITACHI
Inspire the Next

- See "PID loop" on wikipedia https://en.wikipedia.org/wiki/PID_controller
- ivy's PID loop dynamically adjusts IOPS up and down to hit a target value for the focus metric.
- The "error signal" is the difference between the measured focus metric value and the target value.

PID loop – computing new IOPS setting

HITACHI
Inspire the Next

- The user provides 3 multiplier factor constants: p, i, d.
- The new total_IOPS setting is
  - "p" times the error signal (**p**roportional factor)
  + "i" times the sum of the error signal since the start of the test (**i**ntegral factor)
  + "d" time the rate of change of the error signal (**d**erivative factor)
- The ivy engine "edit rollup" mechanism sends out the new `total_IOPS` setting to the focus metric's rollup instance (usually "`all=all`"), where it takes effect in real time.

[Go] parameter summary

HITACHI
Inspire the Next

- Overall
  - stepname = stepNNNN
  - subintervalseconds = 5
  - warmup_count = 1
  - measure_count = 1
  - cooldown_by_wp = on
- For dfc = pid
  - p = 0
  - i = 0
  - d = 0
  - target_value = 0
- For measure = on
  - accuracy_plus_minus = 2%
  - confidence = 95%
    - 50%, 60%, 70%, 80%, 90%, 95%, 98%, 99%, 99.5%, 99.8%, or 99.9%
  - max_wp = 100%
  - min_wp = 0%
  - max_wp_change = 3%
- Focus metric
  - focus_rollup = all
  - source = ""
    - or workload / RAID_subsystem
  - subsystem_element = ""
  - element_metric = ""
  - category = overall
    - or read, write, random, sequential, random_read, random_write, sequential_read, sequential_write
  - accumulator_type = ""
    - or bytes_transferred, service_time, response_time
  - accessor = ""
    - avg, count, min, max, sum, variance, standardDeviation

HITACHI
Inspire the Next

<end>
Thank You

Hitachi Data Systems